Estima
1560 Sherman Ave., Suite 1029
Evanston, IL 60201

Sales, Orders  847–864–8772
Technical Support  847–864–1910
Fax:  847–864–6221

Web: www.estima.com
Sales: sales@estima.com
Technical Support: support@estima.com
Preface

Section 1 of the Reference Manual documents all of the instructions available in RATS, in alphabetical order.

Section 2 describes RATS expressions. Section 3 documents some algorithms common to several instructions and Section 4 provides information on programming features that are common to multiple instructions.

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. Note that almost all the information in the Reference Guide is now included in the on-line Help.

A list of the changes and new features in Version 10 is included at the beginning of the User’s Guide.

# Table of Contents

## Preface

<table>
<thead>
<tr>
<th>Section 1: The RATS Instructions</th>
<th>RM–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCUMULATE — Computing Partial Sums</td>
<td>RM–2</td>
</tr>
<tr>
<td>ALLOCATE — Preparing for Work with Data Series</td>
<td>RM–4</td>
</tr>
<tr>
<td>AR1 — Regression with Autocorrelation Correction</td>
<td>RM–6</td>
</tr>
<tr>
<td>ASSOCIATE: Assigning Coefficients to Equations</td>
<td>RM–11</td>
</tr>
<tr>
<td>BOOT — Randomization for Bootstrapping</td>
<td>RM–13</td>
</tr>
<tr>
<td>BOXJENK — ARIMA, Transfer Function, Intervention Models</td>
<td>RM–15</td>
</tr>
<tr>
<td>BREAK — Breaking Control Out of Loops</td>
<td>RM–24</td>
</tr>
<tr>
<td>CACCUMULATE — Partial Sums of Complex Series</td>
<td>RM–25</td>
</tr>
<tr>
<td>CALENDAR — Setting the Date Information</td>
<td>RM–26</td>
</tr>
<tr>
<td>CATALOG — Listing the Contents of a RATS Data File</td>
<td>RM–32</td>
</tr>
<tr>
<td>CDF — Computing Marginal Significance Levels</td>
<td>RM–33</td>
</tr>
<tr>
<td>CHANGE — Switching I/O Units</td>
<td>RM–34</td>
</tr>
<tr>
<td>CLABELS: Labeling Complex Series</td>
<td>RM–36</td>
</tr>
<tr>
<td>CLEAR — Clearing a Data Series</td>
<td>RM–37</td>
</tr>
<tr>
<td>CLN, CEXP and Related Instructions</td>
<td>RM–38</td>
</tr>
<tr>
<td>CLOSE — Closing an Open I/O Unit</td>
<td>RM–39</td>
</tr>
<tr>
<td>CMASK: Creating Seasonal Masks</td>
<td>RM–40</td>
</tr>
<tr>
<td>CMOMENT — Cross-Moment and Correlation Matrices</td>
<td>RM–41</td>
</tr>
<tr>
<td>CMULTIPLY and related instructions</td>
<td>RM–45</td>
</tr>
<tr>
<td>COMPUTE — Evaluating Scalar and Matrix Expressions</td>
<td>RM–47</td>
</tr>
<tr>
<td>COPY — Flexible Data Output</td>
<td>RM–51</td>
</tr>
<tr>
<td>CORRELATE — Autocorrelation and Related Functions</td>
<td>RM–55</td>
</tr>
<tr>
<td>CPRINT: Printing Complex Series</td>
<td>RM–59</td>
</tr>
<tr>
<td>CROSS — Cross-Correlations</td>
<td>RM–61</td>
</tr>
<tr>
<td>CSAMPLE: Reducing Frequencies</td>
<td>RM–65</td>
</tr>
<tr>
<td>CSET: General Transformations</td>
<td>RM–66</td>
</tr>
<tr>
<td>CTOR: Data From Complex to Real</td>
<td>RM–68</td>
</tr>
<tr>
<td>CVMODEL — Covariance Matrix Modelling</td>
<td>RM–70</td>
</tr>
<tr>
<td>CXTREMUM: Extreme Values for Complex Series</td>
<td>RM–74</td>
</tr>
<tr>
<td>DATA — Reading Data Series</td>
<td>RM–75</td>
</tr>
<tr>
<td>DBOX — Creating User-Defined Dialog Boxes</td>
<td>RM–80</td>
</tr>
<tr>
<td>DDV — Discrete Dependent Variables</td>
<td>RM–89</td>
</tr>
<tr>
<td>DEBUG — Debugging Tools</td>
<td>RM–95</td>
</tr>
<tr>
<td>DECLARE — Setting Data Types</td>
<td>RM–96</td>
</tr>
<tr>
<td>DEDIT — Initiating RATS Data File Editing</td>
<td>RM–98</td>
</tr>
<tr>
<td>DENSITY — Estimate Density Function</td>
<td>RM–100</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DETERMINISTIC</td>
<td>Listing the “Other” Variables in a VAR</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>Time Series Differencing</td>
</tr>
<tr>
<td>DIMENSION</td>
<td>Setting Array Sizes</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Displays Computed Values</td>
</tr>
<tr>
<td>DLM</td>
<td>Dynamic Linear Models</td>
</tr>
<tr>
<td>DO</td>
<td>Simple Loops Over an Index</td>
</tr>
<tr>
<td>DOFOR</td>
<td>Looping Over a List</td>
</tr>
<tr>
<td>DSGE</td>
<td>Dynamic Stochastic General Equilibrium Models</td>
</tr>
<tr>
<td>DUMMY</td>
<td>Generating Dummy and Related Variables</td>
</tr>
<tr>
<td>ECT</td>
<td>Error Correction Terms</td>
</tr>
<tr>
<td>EIGEN</td>
<td>Eigen Decomposition of Matrices</td>
</tr>
<tr>
<td>ENCODE</td>
<td>Restricted Regressions</td>
</tr>
<tr>
<td>END</td>
<td>Ending a Program</td>
</tr>
<tr>
<td>ENTER</td>
<td>Defining Your Own Supplementary Cards</td>
</tr>
<tr>
<td>ENVIRONMENT</td>
<td>Error Modes and Other Options</td>
</tr>
<tr>
<td>EQUATION</td>
<td>Defining Linear Equations</td>
</tr>
<tr>
<td>EQV</td>
<td>Attaching Names to Numbered Series</td>
</tr>
<tr>
<td>ERRORS</td>
<td>Decomposition of Forecast Variance</td>
</tr>
<tr>
<td>ESMOOTH</td>
<td>Exponential Smoothing</td>
</tr>
<tr>
<td>ESTIMATE</td>
<td>Estimating a VAR System</td>
</tr>
<tr>
<td>EWISE</td>
<td>Elementwise Operations on Matrices</td>
</tr>
<tr>
<td>EXCLUDE</td>
<td>Testing Exclusion Restrictions</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Executing a PROCEDURE</td>
</tr>
<tr>
<td>EXP, SQRT, MOVE</td>
<td>Convenience Transformations</td>
</tr>
<tr>
<td>EXTREMINUM</td>
<td>Extreme Values</td>
</tr>
<tr>
<td>FFT, IFT</td>
<td>Fourier Transforms</td>
</tr>
<tr>
<td>FILTER</td>
<td>General Linear Filtering</td>
</tr>
<tr>
<td>FIND</td>
<td>General Optimization</td>
</tr>
<tr>
<td>FIXED</td>
<td>Setting Values of Arrays in PROCS and FUNCTIONS</td>
</tr>
<tr>
<td>FMATRIX</td>
<td>Matrix from a Filter</td>
</tr>
<tr>
<td>FOLD</td>
<td>Folding a Spectrum</td>
</tr>
<tr>
<td>FORECAST</td>
<td>Dynamic and Static Forecasts</td>
</tr>
<tr>
<td>FREQUENCY</td>
<td>Initializing Frequency Domain Analysis</td>
</tr>
<tr>
<td>FRML</td>
<td>Creating Formulas</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>User-Defined Functions</td>
</tr>
<tr>
<td>GARCH</td>
<td>ARCH and GARCH Models</td>
</tr>
<tr>
<td>GBOX</td>
<td>Graphing Box Plots</td>
</tr>
<tr>
<td>GCONTOUR</td>
<td>Contour Plots</td>
</tr>
<tr>
<td>GOTO</td>
<td>Go to a Specific Instruction</td>
</tr>
<tr>
<td>GRAPH</td>
<td>High-Resolution Time Series Graphs</td>
</tr>
<tr>
<td>GROUP</td>
<td>Building General Simultaneous Models</td>
</tr>
<tr>
<td>GRPARM</td>
<td>Graphics Parameters</td>
</tr>
<tr>
<td>GRTEXT</td>
<td>Adding Text to Graphs</td>
</tr>
<tr>
<td>GSAVE</td>
<td>Saving Graphs</td>
</tr>
<tr>
<td>GSET</td>
<td>Setting Series of Arrays</td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>PRINT — Display Data Series ................................................................. RM–383</td>
<td></td>
</tr>
<tr>
<td>PRJ — Fitted Values/Normal Distribution Statistics ................................ RM–385</td>
<td></td>
</tr>
<tr>
<td>PROCEDURE — User-Defined Procedures .................................................. RM–389</td>
<td></td>
</tr>
<tr>
<td>PRTDATA — Printing Data File Series .................................................... RM–391</td>
<td></td>
</tr>
<tr>
<td>PSTATS — Analysis of Variance for Panel Data ........................................ RM–393</td>
<td></td>
</tr>
<tr>
<td>QUERY — Requesting Input from the User ............................................... RM–395</td>
<td></td>
</tr>
<tr>
<td>QUIT — Aborting Data Editing ................................................................. RM–397</td>
<td></td>
</tr>
<tr>
<td>QZ Instruction — QZ Decomposition ...................................................... RM–398</td>
<td></td>
</tr>
<tr>
<td>RATIO — Tests with Multiple Equation Systems ...................................... RM–399</td>
<td></td>
</tr>
<tr>
<td>READ — General Information Input ......................................................... RM–401</td>
<td></td>
</tr>
<tr>
<td>RELEASE — Releasing Segments of Memory ............................................ RM–404</td>
<td></td>
</tr>
<tr>
<td>REPORT — Report Generation ................................................................. RM–405</td>
<td></td>
</tr>
<tr>
<td>RESTRICT — Testing or Imposing General Linear Restrictions ................... RM–411</td>
<td></td>
</tr>
<tr>
<td>RETURN — Returning from a PROCEDURE or FUNCTION ........................ RM–416</td>
<td></td>
</tr>
<tr>
<td>REWIND — Rewinding a RATS I/O Unit .................................................... RM–417</td>
<td></td>
</tr>
<tr>
<td>RLS — Recursive Least Squares .............................................................. RM–418</td>
<td></td>
</tr>
<tr>
<td>RREG — Robust Regression .................................................................... RM–422</td>
<td></td>
</tr>
<tr>
<td>RTOC: Transferring from Real to Complex ............................................. RM–426</td>
<td></td>
</tr>
<tr>
<td>SAMPLE — Extracting Data and Changing Frequencies ............................ RM–428</td>
<td></td>
</tr>
<tr>
<td>SAVE — Saving an Edited RATS Format File ............................................ RM–430</td>
<td></td>
</tr>
<tr>
<td>SCATTER — High-Resolution X-Y Scatter Plots ........................................ RM–431</td>
<td></td>
</tr>
<tr>
<td>SEASONAL — Creating Seasonal Dummies ............................................... RM–439</td>
<td></td>
</tr>
<tr>
<td>SEED — Initializing the Random Number Generator ................................... RM–442</td>
<td></td>
</tr>
<tr>
<td>SELECT — User Selection from a List ...................................................... RM–444</td>
<td></td>
</tr>
<tr>
<td>SET — General Data Transformations ..................................................... RM–446</td>
<td></td>
</tr>
<tr>
<td>SHOW — Run-Time Information ............................................................... RM–449</td>
<td></td>
</tr>
<tr>
<td>SIMULATE — Random Simulations of a Model ......................................... RM–450</td>
<td></td>
</tr>
<tr>
<td>SMPL — Setting the Default Entry Range ................................................ RM–455</td>
<td></td>
</tr>
<tr>
<td>SOURCE — Secondary Input Files ............................................................ RM–457</td>
<td></td>
</tr>
<tr>
<td>SPECIFY — Bayesian Priors for Vector Autoregressions .......................... RM–458</td>
<td></td>
</tr>
<tr>
<td>SPGRAPH — Special Graphs .................................................................... RM–462</td>
<td></td>
</tr>
<tr>
<td>SSStats — Series Statistics ..................................................................... RM–467</td>
<td></td>
</tr>
<tr>
<td>STATISTICS — Sample Statistics ............................................................ RM–469</td>
<td></td>
</tr>
<tr>
<td>STEPS — Static Forecasts ........................................................................ RM–473</td>
<td></td>
</tr>
<tr>
<td>STORE — Saving Data on a RATS Data File ............................................. RM–475</td>
<td></td>
</tr>
<tr>
<td>STWISE — Stepwise Regressions ............................................................. RM–479</td>
<td></td>
</tr>
<tr>
<td>SUMMARIZE — Sums of Coefficients ....................................................... RM–483</td>
<td></td>
</tr>
<tr>
<td>SUR — Linear Systems Estimation ........................................................... RM–487</td>
<td></td>
</tr>
<tr>
<td>Restriction Across Equations: SUR with EQUATE .................................. RM–493</td>
<td></td>
</tr>
<tr>
<td>SWEEP — Sweep Regressions .................................................................. RM–494</td>
<td></td>
</tr>
<tr>
<td>SYSTEM — Setting Up a VAR System ...................................................... RM–497</td>
<td></td>
</tr>
<tr>
<td>TABLE — Basic Statistics on Series ........................................................ RM–499</td>
<td></td>
</tr>
<tr>
<td>TAPER: Tapering Data .............................................................................. RM–502</td>
<td></td>
</tr>
<tr>
<td>TEST — Testing Specific Coefficient Values .......................................... RM–504</td>
<td></td>
</tr>
<tr>
<td>THEIL — Computing Forecast Performance Statistics ............................. RM–507</td>
<td></td>
</tr>
</tbody>
</table>
Contents

TRFUNCTION: Transfer Functions .................................................................................. RM–513
TVARYING — Time-Varying Coefficients .................................................................. RM–515
TYPE — Setting Procedure Parameter Types ............................................................. RM–516
UFORECAST — Univariate Forecasting ..................................................................... RM–517
UNTIL — Conditional Loops ...................................................................................... RM–520
USERMENU — Defining Pull-Down Menus ................................................................. RM–521
VADD — Adding a Variable to An Equation ............................................................... RM–524
VARIABLES — Specifying the Variables in a VAR .................................................. RM–525
VCV — Computing a Residual Covariance Matrix .................................................. RM–526
VREPLACE — Substituting Variables Within an Equation ........................................ RM–529
WHILE — Conditional Looping ................................................................................. RM–531
WINDOW: Smoothing Spectral Estimates ................................................................... RM–532
WRITE — Printing Matrices ....................................................................................... RM–535
X11 — X11 Seasonal Adjustment ............................................................................... RM–537

Section 2: Expression Syntax and Operators RM–541

Section 3: Algorithms RM–545

  Autocorrelations ..................................................................................................... RM–546
  Long-Run Variance/Robust Covariance Calculations ........................................... RM–548

Section 4: Common Elements RM–551

  Date Notation .......................................................................................................... RM–552
  Picture Codes .......................................................................................................... RM–553
  SMPL Option ........................................................................................................... RM–554
  SPREAD Option ...................................................................................................... RM–555
  SHUFFLE Option .................................................................................................... RM–556
  WEIGHT option ....................................................................................................... RM–557

Bibliography RM–559
Section 1: The RATS Instructions

Section 1 of the *Reference Manual* provides a detailed description of every RATS instruction. 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**.
- **Parameter names** are always in italicized Courier font. For example, *start, end, series*.
- **Option names** 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.

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

Wizard

You can use the Data/Graphics—Transformations wizard to set up this instruction. 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 some older versions.)

Missing Values

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

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

produces the following output
Accumulate

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>TREND</th>
<th>TRENDSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>6</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>44</td>
</tr>
</tbody>
</table>

Examples

```plaintext
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.

```plaintext
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

`ACCUMULATE` is sometimes used when all that is needed is the final value, not the whole series of sums. For that, it’s better to use `SSTATS`:

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

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

`ACCUMULATE` is (almost) the inverse of `DIFFERENCE`. `DIFFERENCE` followed by `ACCUMULATE` loses the information about the level included in the first value of the original series. The `SET` instruction with the `FIRST` option can be used to do roughly the same thing as `ACCUMULATE`, but with a non-zero “pre-sample” value:

```plaintext
uforecast(equation=diffeq) dxfore fstart fend
set(first=x{1}+dxfore) xfore fstart fend = xfore{1}+dxfore
```
Allocate

ALLOCATE — Preparing for Work with Data Series

The ALLOCATE instruction sets the standard workspace length for your program. This information is used by instructions, such as DATA and SET, that work with series.

The ALLOCATE instruction is now optional. If you prefer, you can omit the ALLOCATE and set the workspace length using the start and end parameters on your first DATA or SET 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/Graphics menu, automatically determine the appropriate range when the data are read from a single data file.

Parameters

length

This sets the standard workspace 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.
**Allocate**

### Examples

- `allocate 1200`  
  1200 observation data set
- `calendar(d) 1986:1:8`  
  Daily, Jan. 8, 1986 through Dec. 31, 2017
- `allocate 2017:12:31`
- `cal(m) 2008:1`  
- `all 24`
- `cal(m,panelobs=12) 1998:1`  
  Monthly panel data, 12 months per individual
- `all 50//12` (or `50//1998:12`)  
  and 50 individuals

### 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 page UG–483 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.
- **UG Section 15.3**  
  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 2.4 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 is in the Statistics—Linear Regressions wizard. Select either “AR(1)—Regression” or “AR(1)—Instrumental Variables” from the “Method” list.

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.

Options

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

This chooses the method for estimation. See “Technical Details” for details:

- **CORC** is (iterated) 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.

AR1 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 ρ rather than having it estimated.
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.

**cvcrit**=convergence criterion for $\rho$ [.0001]

These are the same as for LINREG.

**print**/[noprint]

**vcv**/[novcv]

**smpl**=SMPL Series or formula (“SMPL Option” on page RM–546)

**dfc**=Degrees of Freedom Correction (Additional Topics PDF, Section 1.4)

**unravel**/[nounravel] (Section 2.10)

**equation**=equation to estimate

**entries**=number of supplementary card entries to process [all]

**title**="title to identify estimation method"

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 apply to panel data sets (User’s Guide, Section 12.5). If you use HETEROGENOUS, AR1 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. With HETEROGENOUS, you can use PRHOS to save the series of estimated rho values for each individual.
Technical Details

For the following model with first order serially correlated errors:

\[ y_i = X_i \beta + u_i, \quad u_i = \rho u_{i-1} + \varepsilon_i \]

the (log) likelihood function, assuming Normality, is

\[
\frac{-T}{2} \log(2\pi) - \frac{T}{2} \log(\sigma^2) + \frac{1}{2} \log(1 - \rho^2) - \frac{1}{2\sigma^2} (y_1 - X_1 \beta)^2 - \frac{1}{2\sigma^2} \left\{ \sum_{t=2}^{T} (y_t - \rho y_{t-1} - (X_t - \rho X_{t-1}) \beta)^2 \right\}
\]

- 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 \((1 - \rho^2)(y_1 - X_1 \beta)^2\)

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 last squares of \(y_t - \rho y_{t-1}\) on \(X_t - \rho X_{t-1}\). If you choose METHOD=HILU, AR1 uses a search procedure to minimize (3) over \(\rho\). If you use METHOD=CORC, given \(u=y-X\beta\), \(\rho\) is estimated by

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

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

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

Both of these are to adjust for the fact that

\[
u_t \mid u_{t-s} \sim N \left( \rho^s u_{t-s}, \sigma^2 \left( 1 + \rho^2 + \rho^4 + \ldots + \rho^{2(s-1)} \right) \right)
\]

**Examples**

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

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

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

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

**Variables Defined by AR1**

AR1 defines the standard regression variables (see page RM–288 in 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 Hildreth-Lu Search
Dependent Variable RATE
Monthly Data From 1959:04 To 1996:02
Usable Observations 443
Degrees of Freedom 438
Centered R^2 0.9679799
R-Bar^2 0.9676875
Uncentered R^2 0.9945027
Mean of Dependent Variable 6.0806546275
Std Error of Dependent Variable 2.7714419161
Standard Error of Estimate 0.4981857000
Sum of Squared Residuals 108.70677837
Regression F(4,438) 3310.2261
Significance Level of F 0.0000000
Log Likelihood -317.4010
Durbin-Watson Statistic 1.6508
Q(36-1) 144.4029
Significance Level of Q 0.0000000

<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>-53.04561425</td>
<td>30.83708607</td>
<td>-1.72019</td>
<td>0.08610443</td>
</tr>
<tr>
<td>2. IP</td>
<td>0.28415004</td>
<td>0.05371758</td>
<td>5.28970</td>
<td>0.00000000</td>
</tr>
<tr>
<td>3. GRM2</td>
<td>-65.27152869</td>
<td>9.73912096</td>
<td>-6.70199</td>
<td>0.00000000</td>
</tr>
<tr>
<td>4. GRFPI{1}</td>
<td>6.45196308</td>
<td>2.95281378</td>
<td>2.18502</td>
<td>0.02941644</td>
</tr>
<tr>
<td>5. RHO</td>
<td>0.99916389</td>
<td>0.00522081</td>
<td>191.38118</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. AR1 computes the standard errors and covariance matrix from a linearization of the objective function.

If you use the HETEROGENOUS option with panel data, AR1 omits the output for $\%RHO$, and computes the standard errors from the second stage regression on the quasi-differenced data.

Higher-order Autocorrelation Corrections
To estimate a model with corrections for higher-order serial correlation, the simplest choice is BOXJENK with the GLS option. For instance,

```
boxjenk(gls,ar=2) y
# constant x1 x2
```

estimates a model with second order autoregressive noise.
ASSOCIATE: Assigning Coefficients to Equations

**ASSOCIATE** sets the coefficients (and possibly other aspects) of an equation. This is an older instruction, and you should prefer to use the functions %EQNSETCOEFFS, %EQNSETVARIANCE, or %MODELSETCOEFFS, or the COEFFS option on EQUATION.

```plaintext
associate( options )  equation  coeffVECTOR
# coefficients   (under some circumstances)
```

**Description**

When you estimate an equation within RATS, the estimating instruction automatically sets the coefficients and residual variance. So, you usually 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 copy them from a VECTOR by using the `coeffVECTOR` parameter.
3. It can read them from a file (using the options BINARY or FREE).

This is rarely used in any modern programs, but you may come across it in code written from RATS version 4 or earlier. If you need to set or reset the coefficients of a Vector Autoregression (such as when doing Monte Carlo integration and similar types of simulations), use the functions %MODELGETCOEFFS and %MODELSETCOEFFS.

Similarly, the functions %EQNSETCOEFFS and %EQNSETVARIANCE provide alternative methods of setting the coefficient values and variance of an individual equation.

**Parameters**

- **equation**
  
  **ASSOCIATE** sets the coefficients of this equation.

- **coeffVECTOR**
  
  (Optional) This is a VECTOR which holds the coefficients.

**Options**

- **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.

- **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. The residuals are usually saved when the equation is estimated.
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 if you are also using the coeffVECTOR 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. In either case, omit the supplementary card.

Examples

There are several ways to input the relationship

\[ y_t = 5.0 + 1.3y_{t-1} - 0.4y_{t-2} + u_t; \quad Var(u_t) = 0.01 \]

This one creates an EQUATION named YEQ:

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

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

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

You can also directly create a FRML:

```plaintext
frml(variance=1.0) yfrml y = 5.0+1.3*y{1}-.4*y{2}
```

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 it 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 16.9 of the User’s Guide for detailed information.

```
boot( option )  BOOTseries  start  end  lower  upper
```

**Parameters**

- **BOOTseries**
  The SERIES of INTEGERS created by **BOOT**.

- **start**  **end**
  The range of entries in **BOOT series** set by **BOOT**. By default, the standard workspace.

- **lower**  **upper**
  The lower and upper bounds (inclusive on both ends) 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. With **NOREPLACE**, once a value is drawn, it won’t be drawn again for a different element of **BOOTseries**. 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]**
  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. **METHOD=CIRCULAR** is similar to **METHOD=OVERLAP** except that any point in [**lower**, **upper**] can be selected as the start of a block—if the block hits **upper** it wraps to continue with **lower**.
panel/[nopanel]
Use PANEL if you want to randomize the individuals in a (balanced) panel data set. Within each individual, the original time order is maintained. Use the PANEL option on any subsequent SET instructions you use to get the shuffled data.

Examples

```plaintext
calendar(q) 1980:1
allocate 2017:4
boot entries / 1980:1 2003:4
```

The dates of lower and upper correspond to the 1st and the 96th entry numbers, so BOOT fills the ENTRIES series with random integers ranging from 1 to 96.

```plaintext
@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
```

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

```plaintext
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. A similar calculation can be done with 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
%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 6 of the *User’s Guide* for more information.

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

**Wizard**

The *Time Series—Box Jenkins (ARIMA) Models* wizard 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

\[
\text{span=seasonal span for seasonal ARMA [CALENDAR seasonal—see below]}
\]
\[
\text{difs=number of regular differencings [0]}
\]
\[
\text{sdiffs=number of seasonal differencings [0]}
\]
\[
\text{ar=list of autoregressive lags [0]}
\]
\[
\text{ma=list of moving average lags [0]}
\]
\[
\text{sar=list of seasonal autoregressive lags [0]}
\]
\[
\text{sma=list of seasonal moving average lags [0]}
\]

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

\[
\text{[print]/noprint}
\]
\[
\text{vcv/[novcv]}
\]
\[
\text{dfc=Degrees of Freedom Correction (User's Guide, Section 5.15)}
\]

These are the same as for LINREG. Note that there are no WEIGHT or SPREAD options.

\[
\text{smpl=SMPL series or formula (“SMPL Option” on page RM–546)}
\]

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.

\[
\text{maxl/[nomaxl]}
\]

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.

\[
\text{method=[gauss]/bfgs/simplex/genetic/annealing/ga/initial/evaluate}
\]
\[
\text{iterations=iteration limit [100]}
\]
\[
\text{subiterations=subiteration limit [30]}
\]
\[
\text{cvcrit=convergence limit [.00001]}
\]
\[
\text{trace/[notrace]}
\]

BOXJENK uses non-linear optimization. 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).

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 4 in the User’s Guide for more details.

\[ \text{pmethod=} \text{gauss/bfgs/simplex/genetic/annealing/ga/initial} \]
\[ \text{piter} = \text{number of PMETHOD iterations to perform [none]} \]

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 estimation methods—RATS will automatically switch to the METHOD choice after completing the preliminary iterations requested using PMETHOD and PITER.

\[ \text{initial=} \text{VECTOR of initial guesses} \]

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. See ““Coefficient Order”” for the proper placement.

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.

\[ \text{define=} \text{equation to define from result} \]

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:

\[ y_t = \frac{10}{1-.3L} x_t + \frac{1+.4L}{1-.5L} u_t \]

into the equation

\[ (1-.3L)(1-.5L)y_t = 10(1-.5L)x_t + (1-.3L)(1+.4L)u_t \], or

\[ y_t = .8y_{t-1} - .15y_{t-2} + 10x_t - 5x_{t-1} + u_t + .1u_{t-1} - .12u_{t-2} \]

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.
Boxjenk

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 “Supplementary Cards”. 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.

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.

sv=input variance of residuals [standard variance calculations]
This can be used (combined with INITIAL and METHOD=EVALUATE) to get the log likelihood of an ARIMA model at a specific value of the residual variance. This can be used as a step in a Gibbs sampling scheme. The point estimates of the coefficients themselves aren’t affected by the residual variance, so this changes only the log likelihood from what you would get by using the standard concentrated variance estimate. This was added with version 9.1.

Options for “RegARIMA” Models
regressors/[noregressors]
gls/[nogls]
Use REGRESSORS or GLS if you want to include additional non-ARIMA variables using standard regression format, rather than the intervention/transfer function style inputs available using the INPUTS option. While similar, the two have a different emphasis. With GLS, it’s the mean equation represented by the explanatory variables which is the focus of the estimation; while with REGRESSORS, it’s the ARIMA model itself. With GLS, the output is switched around so the explanatory variables are listed first. GLS forces use of maximum likelihood (i.e. same as the MAXL option) and also includes the behavior of the APPLYDIFFERENCES option.

meaneq=equation to define from the mean model only
This saves the equation created by the regressors only

outlier=[none]/ao/ls/tc/standard/all
critical=critical (t-statistic) value [based on # of observations]
With any of the choices for OUTLIER other than NONE, BOXJENK does an automatic procedure for detecting and removing outliers. This can be used with or without the GLS option. If used without GLS, it operates like GLS with an empty set of
base regressors, that is, it estimates dummy shifts to the mean of the dependent variable, using maximum likelihood. CRITICAL allows you to set the $t$-statistic value used for the automatic outlier detection threshold. See “Outliers” on page RM–21 for more details.

**adjust**=series of RegARIMA adjustments [not used]
This is a series which has the combined effects on the mean of all the regression coefficients, including input regressors and outliers, leaving out only the CONSTANT (if it's included in your original set of regressors). You can input this (or a transformation of it) into **X11** as a set of preliminary adjustment factors.

### 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:

$$\left(\omega_0 + \omega_1 L + \ldots + \omega_n L^n\right) \div \left(1 - \delta_1 L - \ldots - \delta_m L^m\right) X_{t-d}$$

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 in the model.

**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:

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

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
- \( \theta \) 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 coefficient 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 software.

If you are using the INPUTS option, polynomial terms of the form shown in “Supplementary Cards” 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. The simplex and genetic methods are also available with the METHOD and PMETHOD options. These can be helpful in improving initial parameter guesses for models that prove 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.
Outliers

The OUTLIER and related options implement an automatic procedure for dealing with outliers in the data.

The choices for OUTLIER are NONE, AO, LS, TC, STANDARD, and ALL. AO locates additive outliers. For an outlier at entry $t_0$, the resulting dummy would be 1 only at $t_0$. LS detects level shifts, generating a dummy with $-1$'s from the beginning of the sample until $t_0 - 1$. TC detects temporary changes. For a temporary change starting at $t_0$, the dummy takes the value 1 at $t_0$, then declines exponentially for data points beyond that. OUTLIER=A0, OUTLIER=LS and OUTLIER=TC select scans for only the indicated type of outlier. OUTLIER=STANDARD scans for AO and LS, OUTLIER=ALL does all three.

The following procedure is repeated until no further outliers are detected:

Beginning with the last RegARIMA model (including previously accepted outliers), LM tests are performed for each of the requested types of outliers at all data points. If the largest $t$-stat exceeds the critical value, that shift dummy is added to the model, which is then re-estimated.

When there are no further outliers to be added, the list is then pruned by examining the $t$-stats from the full estimation using the same critical value.

Note that the first step uses a “robust” estimate of the standard error of the residuals, based upon the median absolute value. There are several ways to compute maximum likelihood estimates; RATS uses Kalman filtering, Census X12–ARIMA uses optimal backcasting. The two lead to identical values for the likelihood function, identical values for the sum of squares of the residuals, but not to identical sets of estimated residuals. As a result, there can be slight differences between this robust estimate of the standard error. In some cases, they can be large enough to cause the two programs to differ on whether a marginal $t$-stat is above or below the limit. (X12–ARIMA tends to give a lower value for the standard error, and hence higher $t$-statistics). This tends to correct itself in the backwards pruning steps.

Coefficient Order

For a standard ARIMA/Transfer function model, the coefficients are in the model in the following order. This is used in the INITIAL option or for doing hypothesis tests.

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

For RegARIMA models, the regressors are considered to be of principal interest, so they are moved to the start:
Boxjenk

1. Regressors in order
2. Autoregressive
3. Seasonal autoregressive
4. Moving average
5. Seasonal moving average

Hypothesis Tests

You can use TEST and RESTRICT to test hypotheses on the coefficients. The coefficient order is provided in the description of the INITIAL option. It’s probably easiest to use the Regression Tests Wizard to set those up.

Examples

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

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

boxjenk(diffs=1,ar=1,inputs=1,apply,define=saleseq) sales
# ads 0 1 0
boxjenk(diffs=1,ar=||2,4||,define=adseq) ads

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

set change = t>=1974:3
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 ASSIST is ARIMA (0,1,0)×(0,1,1). The intervention is a change in the level of the series from 1974:3 on.

dec vect[series] days(7)
do i=1,7
  set days(i) = %tradeday(t,j)
end do
boxjenk(gls,sdiffs=1,ma=1) logsales
# days

The code above estimates a RegARIMA model with trading day counts as the exogenous variables.
Variables Defined

In addition to the standard regression variables (see page RM–288), **BOXJENK** defines:

- %NARMA: number of ARMA parameters (useful for degrees of freedom correction when computing a $Q$ statistic) (**INTEGER**)
- %NFREE: number of free parameters (**INTEGER**)
- %FUNCVAL: final value of the estimated function (**REAL**)
- %ITERS: iterations completed (**INTEGER**)
- %CONVERGED: 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, which will show the correction to degrees of freedom for the estimated ARMA parameters (below 8 correlations minus 1 lost degree for the one MA parameter). 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 $N_{nnn...n}$ and $D_{nnn...n}$. The $R^2$ is computed based upon the original dependent variable before differencing, if any is used.

This instruction below produces the output shown:

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

**Box-Jenkins - Estimation by ML Gauss-Newton**

<table>
<thead>
<tr>
<th>Dependent Variable</th>
<th>Usable Observations</th>
<th>Degrees of Freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES</td>
<td>35</td>
<td>30</td>
</tr>
</tbody>
</table>

Centered $R^2$: 0.7705281

R-Bar$^2$: 0.7399319

Uncentered $R^2$: 0.9878646

Mean of Dependent Variable: 24.602857143

Std Error of Dependent Variable: 5.898478653

Standard Error of Estimate: 3.008039840

Sum of Squared Residuals: 271.44911031

Log Likelihood: -85.8798

Durbin-Watson Statistic: 1.9789

$Q(8-1)$: 2.7203

Significance Level of $Q$: 0.9096124

<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>14.510838600</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>0.159772291</td>
<td>1.78084</td>
<td>0.08506872</td>
</tr>
</tbody>
</table>
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 GOTO, rather than BREAK, to exit a more complex loop structure.

Example

This does iterated weighted least squares, allowing for up to 40 iterations, but breaking out of the loop if the slope coefficient changes by less than .00001 from one iteration to the next.

```
set resids = %resids
*  
* Iterated. We'll allow up to forty iterations. Each time,
* we'll check to see if the new value of c2 is very close to
* the last one. Once the (absolute value) of the gap is small
* enough, we break the loop. Inside the loop, we suppress the
* regression output. We show the result with the last value once
* we exit the loop.
* 
do iters=1,40
  set logusq = log(resids^2)
  linreg(noprint) logusq
  # constant lf
  if abs(\%beta(2)-c2)<.00001
    break
    compute c2=%beta(2)
    prj logvar
    linreg(equation=base,spread=exp(logvar),noprint)
    set resids = %resids
  end do iters
  disp "Convergence achieved after" iters "iterations"
disp "Estimated c2 is" c2
```
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 \textit{CACCUM} is the Durbin cumulated periodogram test (see Durbin, 1969, and \textit{User’s Guide}, page UG–82).

\textbf{caccumulate(}option\textbf{)} \textit{cseries start end newcseries newstart}

**Parameters**

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

**Options**

- \textit{standardize/[nostandardize]} If you use \textit{STANDARDIZE}, \textit{CACCUMULATE} normalizes the resulting series so the final entry has the value one. (This was called \textit{SCALE} before version 7).

**Example**

This is the business end of a Durbin test. The \textit{CACCUMULATE} instruction cumulates the periodogram (squared Fourier transform) over frequencies 0 to \(\pi\) and standardizes 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 \textit{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
cacc(standardize) 1 1 half
cunits 2
\hline
\end{verbatim}

\textit{Periodogram of resids}

\begin{verbatim}
cacc(standardize) 2 1 half
csubtract 2 1 1 half 3
cxt(part=absval) 3 1 half
\hline
\end{verbatim}

\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 introduced 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 gives 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(julian=SERIES of julian dates for entries)`
  This is used for an irregular time series (for instance, daily data with skipped entries due to holidays) where information is available to map entries to dates. The dates are coded using a “Julian” (day count) mapping.
- `calendar(irregular)`
  For erratic time series—indicates that you have time series data, but without a specific date scheme.
Wizards

The Data/Graphics—Calendar wizard allows you to set the **CALENDAR** using a dialog box. Both of the Data wizards set the **CALENDAR** as part of the process of reading in data.

Other Options

**RECALL**= saved VECT[INTEGER] to be used for calendar scheme
**SAVE**= VECT[INTEGER] into which calendar scheme is saved

Note that **SAVE** does not change the working calendar scheme—this allows you to define a calendar scheme to be used for processing a data file which has no date information but which follows a different arrangement than the main workspace.

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 (“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 Field

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.

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.
Calendar

Daily Data (5, 7, or other number of periods per week)
Use the \texttt{D} option for a daily calendar with five days per week (Saturdays and Sundays are omitted), or the \texttt{7} option for a daily calendar with entries for all seven days.

For a less standard periods per week, use the \texttt{PPW} option. For example, if you have six day per week data starting on June 5, 2000, use: \texttt{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 \texttt{PPD} option to set the periods per day, combined with one of the other options (usually \texttt{D} or \texttt{7}) to describe the day-to-day arrangement of the data set.

You can use any number of periods with \texttt{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 \texttt{PPD=72}.

Other Periodic Schemes
Use the \texttt{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 \texttt{M} or \texttt{Q} options). Use \texttt{DPP} (days per period) for frequencies specified as a specific number of days.

Data Recorded Less Than Once a Year
For data which are equally spaced, but more than one year apart, use the \texttt{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 \texttt{JULIAN=SERIES} with Julian dates if you have irregular data where the dates are known. Otherwise, use \texttt{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 \texttt{CAL(IRREGULAR)} is to inform RATS that the data set does have time series properties. In all other ways, there is no difference between \texttt{CALENDAR(IRREGULAR)} and omitting \texttt{CALENDAR} entirely.

Panel Data
Include the option \texttt{PANELOBS=periods per individual} together with other options to describe the date scheme in the time direction. \texttt{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 \texttt{cannot} use \texttt{PANELOBS} with the \texttt{PPD} option. See page \texttt{UG–393} of the \textit{User’s Guide} for more on panel data.
Examples

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

```
calendar(m) 1950:1
allocate 2017:12
```

```
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.
```

```
cal(7) 1980:1:1
open data daily.rat
data(format=rats) * 1999:12:31
    Daily data with seven days per week, beginning on January 1, 1980, and set the default ending period (December 31, 1999) using a **DATA** instruction.
```

```
cal(ppd=8,d) 2000:7:1
allocate 2003:8:31//8
    Eight time periods per business day, starting July 1, 2000, ending on August 31, 2003 (with a full eight periods on the last day).
```

```
cal(panelobs=52,w) 1990:1:5
allocate 30//52
    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 observations for holidays? There are four ways to deal with this:

1. You can 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 approaches.

2. You can use the Julian option to map entries to coded date values. This will give you a data file without gaps, so (for instance) Monday will be the day after Thursday if the Friday is missing.

3. 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).

4. The final option is to read in the data with gaps as in (1), 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.

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 Introduction, Section 2.5 for details). Thus, if you have a mixture of monthly and quarterly data, you can work with it all at either quarterly or 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. You can then use the @Disaggregate procedure to do more complex interpolations and distributions on the expanded data.

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)

You can also save a calendar scheme without resetting the workspace date scheme by using the SAVE option. For instance, the following sets up a quarterly workspace starting in 1954:1, then reads a data file which has monthly data from 1948:1, compacting by quarterly averages. Since the data file is free format, there is no identifying date information on it, thus the need for the CALENDAR option on the DATA instruction.

cal(q) 1954:1
cal(m,save=mdata) 1948:1
data(format=free,compact=average,calendar=mdata) $
 1954:1 2016:4 ffunds

Deprecated CALENDAR Formats

The following older formats for specifying the CALENDAR instruction are still supported, 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

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.

\[
\text{catalog} (\text{options}) \quad \text{list of data file series} \quad \text{(optional)}
\]

Wizard

If you do use \textit{File–Open} to open a RATS format file, you’ll get the same information, but in a scrolling list. From this list, you can view/edit data, export data, and more.

Parameters

\text{list of series} \quad \text{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

\text{full}/[\text{nofull}] \quad \text{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 \textit{list} parameter.}

\text{unit=}[\text{output}]/\text{copy}/\text{Other} \quad \text{Choose where you want output to go.}

\text{like=\text{"template string for variables to list"}} \quad \text{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

\text{catalog} \quad \text{Lists the names of all the series on the file.}

\text{catalog(like="gdp*",full,unit=copy)} \quad \text{Lists information on all series whose names start with GDP to the current COPY unit.}

\text{catalog \hspace{1em} real\_gnp \hspace{1em} nom\_gnp} \quad \text{Lists full information on the series REAL\_GNP and NOM\_GNP}

See Also . . .

\text{DEDIT} \quad \text{Opens a RATS data file for editing.}
\text{PRTDATA} \quad \text{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 \%ZTEST\( (x) \) compute the same significance levels as **CDF**, but **CDF** can display the results as well.

<table>
<thead>
<tr>
<th>syntax</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cdf( option )</code></td>
<td><code>distribution statistic degree1 degree2</code></td>
</tr>
</tbody>
</table>

### 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**
- **print** 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 page UG–95 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

<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>
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

 CHANGE 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:

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

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 “...” or label expressions 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
  # "Crosspec"  "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) or to zero. This provides an easy way to create or initialize series, and to prevent old data from being included in statistical analyses later on.

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.

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

Parameters

`list of series`  The list of the series you want to clear. You can list an array of series to initialize all the series in the array. Use the ALL option to clear all existing series in memory.

If you include a new series name on the list, that series will be created.

Options

`all/[noall]`  If you use ALL, RATS clears all existing series.

`zeros/[nozeros]`  Use ZEROS if you want to set the values of the series to zero rather than to the missing value code.

`length=number of entries [standard workspace length]`  Use this to initialize the series over the specified number of entries, rather than through the default ending period.

Examples

```
clear resids
```

This creates RESIDS (if it doesn’t already exist), and sets its entries to the missing value code.

```
dec vec[series] simul(4)
clear(length=%regend()+12) simul
```

This creates a VECTOR of SERIES called SIMUL that will be used to hold some out-of-sample simulations. SIMUL is initialized out to 12 periods past the end of the most recent regression. This will allow elements of SIMUL to be set using COMPUTE instructions in a later loop.
CLn

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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>cln</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td>cexp</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td>csqrt</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td>conjugate</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td>cmve</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td>cscale (option)</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
</tbody>
</table>

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 to produce newcseries.

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 create.
- **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: \(\frac{\text{seasonalwidth}}{6}\).
- **bandcenter**: The center (entry number) of the first seasonal band. By default, entry one (zero frequency). If this is \(\text{seasonalwidth}+1\), CMASK will leave the low frequencies unaffected.

Description

CMASK creates a mask for seasonal frequencies. It sets entries \(\text{start}\) to \(\text{end}\) of \(\text{cseries}\) to the value 1.0. Then it sets to zero a band of width \(\text{bandwidth}\) about every \(\text{seasonalwidth}\) entry, beginning with \(\text{bandcenter}\). 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 \(\text{MASK}='\text{masking series'}\) on \(\text{WINDOW}\), 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 — 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 ZXMATRIX option)
```

**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.

**Supplementary Cards**

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 plan to base on this `CMOMENT`. Omit this if using the `EQUATION`, `LASTREG`, `MODEL`, or `INSTRUMENTS` option.

The second card is used only with `ZXMATRIX`.

**Description**

Cross-moment and correlation matrices are important in statistical work. However, the main uses of `CMOMENT` come from the fact that a cross moment matrix which includes both the explanatory variables and the dependent variable(s) of a linear regression have all the sample information required to compute regression coefficients and sums of squared residuals. This can be used in several ways within RATS:

- The `LINREG` instruction with the `CMOMENT` option pulls that information out of the computed cross product matrix. This can cut down on the computation time required for doing large numbers of similar regressions, and also ensures that all those regressions use a common sample. See, for example, the information criteria searches in Section 2.9 in the *User’s Guide*.

- The `%RANMVPOSTCMOM, %RSSCMOM` (for univariate regressions) and `%RANMVKRONCMOM` and `%SIGMACMOM` (for multivariate regressions) functions simulate coefficients from the posterior in a linear model (the `%RAN...` functions) or compute the sum of squared residuals or covariance matrix (the other two) using the statistics included in the cross-moment matrix. This is a huge time-saver in simulations as the only calculation that depends upon the data gets computed just once, rather than with every draw.

- The `%SWEEP` function can be used to do a sequence of “regressions” using a common sample range if all you need are the coefficients and sum of squared residuals.
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 page Int–77 of the Introduction).

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.

smpl=SMPL series or formula ("SMPL Option" on page RM–546)
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=standard SPREAD option ("SPREAD Option" on page RM–547)
This lets you provide a series with residual variances for weighted least squares.

weight=series of entry weights ("WEIGHT option" on page RM–549)
Use this option if you want to provide different weights for each observation.

shuffle=SERIES[INTEGER] for remapping ("SHUFFLE Option" on page RM–548)

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(s) 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 as the \texttt{start} and \texttt{end} parameters of the \texttt{CMOM} instruction.

\texttt{zxmatrix=RECTANGULAR \texttt{Z'}X matrix [unused]}
Takes two sets of supplementary cards, or some combination of \texttt{LASTREG} or \texttt{EQUATION} and \texttt{INSTRUMENT} options and possibly a supplementary card. It computes \texttt{Z'X} where \texttt{Z} is the first list of variables and \texttt{X} the second. If you use \texttt{LASTREG} or \texttt{EQUATION}, that will always be the first list; \texttt{INSTRUMENT} will be first if you use it but not \texttt{LASTREG} or \texttt{EQUATION}. If you only use one of these options, use a single supplementary card to supply the other list of variables. \textit{You cannot use a \texttt{LINREG(CMOMET)} instruction after \texttt{CMOMENT} with the \texttt{ZXMATRIX} option.}

\texttt{matrix=SYMMETRIC array for computed matrix [%CMOM]}
This saves the computed cross-moment matrix in the \texttt{SYMMETRIC} array. By default, the result goes into the array \texttt{%CMOM}, which you can access. \textit{You cannot use a \texttt{LINREG(CMOMET)} instruction after a \texttt{CMOM} using the \texttt{MATRIX} option.}

\texttt{setup/[nosetup]}
\texttt{SETUP} creates and dimensions the \texttt{%CMOM} array and prepares for future \texttt{LINREG(CMOM)} instructions, \textit{but does not actually compute the matrix}. Instead, you can fill the array yourself with subsequent instructions, such as \texttt{COMPUTE}.

Notes
Correlations computed by \texttt{CMOM(CORR)} may be different from \textit{pairwise} correlations computed with the function \texttt{\%CORR} or with \texttt{CROSS. CMOMENT} does the computations over a single set of entries appropriate for \textit{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 \textit{any} of the series in the cross-moment matrix is missing is dropped from the sample.

Variables Defined by \texttt{CMOMENT}

\begin{tabular}{|l|l|}
\hline
\texttt{\%CMOM} & Cross-moment matrix (\texttt{SYMMETRIC}) \\
\texttt{\%MEANV} & Vector of means of the variables (\texttt{VECTOR}) \\
\texttt{\%NCMOM} & Number of Variables (\texttt{INTEGER}) \\
\texttt{\%NOBS} & Number of Observations (\texttt{INTEGER}) \\
\hline
\end{tabular}
Examples

```plaintext
cmom
  # constant m1{0 to 12} gnp
do lag=1,12
    linreg(cmoment) gnp
    # constant m1{0 to lag}
end do lag
```

computes a set of distributed lag regressions. This is similar to what you would get with LINREGs 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.

```plaintext
cmom(corr,print)
  # rate30 rate60 rate90 rate120 rate1yr rate2yr
```

computes and prints the correlation of six interest rate series.

```plaintext
cmom 1974:2 *
  # uscpi{1} exrat{1} itcpi{1} uscpi exrat italcpi
  compute s=%sweeptop(%cmom,3)
```

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

```plaintext
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).

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 and related instructions

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

![cmul (option) cseries1 cseries2 start end newcseries newstart cmadd cseries1 cseries2 start end newcseries newstart cssubtract cseries1 cseries2 start end newcseries newstart cdivide cseries1 cseries2 start end newcseries newstart](image)

**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 \textit{cseries1} and \textit{cseries2}.
- **newcseries** Complex series for the result. By default, same as \textit{cseries1}.
- **newstart** Start of the result series. By default, same as \textit{start}.

**Option (only for CMULTIPLY)**

- **scale=**real expression to scale result  
  Use this option to multiply all entries of \textit{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 \texttt{SCALE=1./(2*PI*N)}, where \textit{N} is the number of actual data points and not the padded length. If you use a taper (see **TAPER**), you should use the \texttt{%SCALETAP} variable defined by **TAPER** instead of \textit{N}.

**Examples**

- \texttt{cmult(scale=1./(2*%pi*nobs)) 1 1}
- \texttt{cmult 1 2 / 3}

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

```
compute variable = 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 instructions like **LINREG** and **NLLS**.
- You can set integer values for starting and ending observations, then use these variables as parameters on 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 or C and often much more efficiently, since RATS can work with entire matrices. And it has similar capabilities to well-known “math” packages.

Parameters

**variable**

This can be almost any type of variable supported by RATS, except **FRML** (formula) which must be set using the instruction **FRML**. It can be an element of a series, but not an entire series itself—use **SET**, **CSET** or **GSET** (depending upon the type of the series) 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 1.4 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 computing 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:

```plaintext
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:

```plaintext
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.

```plaintext
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:

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

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

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

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

**In-Line Matrices**

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

```plaintext
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

```plaintext
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).

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

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.

```plaintext
compute sales(2009:1) = %na
```

sets entry 2009:1 of series `SALES` to the missing value code.

```plaintext
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 to the vector B the positive entries of the vector A. 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 1.1      Scalar calculations
UG, Section 1.7      Matrix calculations


Copy

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.

```plaintext
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 View—Series Window to display a list of all the series in memory, selecting the series you wish to export, and doing File—Export. You can choose the file format using the “Save As Type” field.

Parameters

- **start end**: Range of entries to be printed or copied to a file. 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

- **format**=[free]/binary/cdf/dbf/dif/dif/fame/html/portable/prn/rats/rtf/tex/tsd/wks/xls/xlsx/"(FORTRAN format)"

  The format to be used for the output. See Chapter 2 of the Introduction and Data Formats on the help 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.

- **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 COPY still supports the choices OBSERVATION (equivalent to COLUMNS) and VARIABLE (equivalent to ROWS) used in older versions. The ORG option is ignored for most formats, since only text files and spreadsheets allow for the different arrangements.
unit=\[\text{copy}\]/\text{output}/\text{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.

dates/[\text{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.

picture=\text{"picture code formatting string"}
This provides a simple means of formatting the numbers. A picture code 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 "text" formats FREE, PRN, RTF, DIF, CDF, TSD, TEX and HTML. Formats which produce a spreadsheet or database, such as XLSX or WKS, will always write the full double precision value. See "Picture Codes" on page RM–545 for more information.

smpl=\text{SMPL series or formula} ("SMPL Option" on page RM–546)
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.

header=\text{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.

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 commands. 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 ORG=ROWS) or observation (with ORG=COLS), FORMAT=FREE uses multiple lines if necessary to keep the width of the output from getting too large. For instance, this is a segment of the output for a single quarterly data series (this was done with options FORMAT=FREE, ORG=ROWS, DATES):

<table>
<thead>
<tr>
<th>Year</th>
<th>Data 1</th>
<th>Data 2</th>
<th>Data 3</th>
<th>Data 4</th>
<th>Data 5</th>
<th>Data 6</th>
<th>Data 7</th>
<th>Data 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1947:01</td>
<td>224.9000000</td>
<td>229.1000000</td>
<td>233.3000000</td>
<td>243.6000000</td>
<td>249.6000000</td>
<td>257.1000000</td>
<td>264.0000000</td>
<td>265.5000000</td>
</tr>
<tr>
<td>1948:01</td>
<td>249.6000000</td>
<td>257.1000000</td>
<td>264.0000000</td>
<td>265.5000000</td>
<td>276.9000000</td>
<td>294.5000000</td>
<td>305.9000000</td>
<td>305.9000000</td>
</tr>
<tr>
<td>1949:01</td>
<td>260.1000000</td>
<td>256.6000000</td>
<td>258.6000000</td>
<td>265.5000000</td>
<td>276.9000000</td>
<td>294.5000000</td>
<td>305.9000000</td>
<td>305.9000000</td>
</tr>
<tr>
<td>1950:01</td>
<td>267.4000000</td>
<td>276.9000000</td>
<td>294.5000000</td>
<td>305.9000000</td>
<td>319.9000000</td>
<td>327.7000000</td>
<td>334.4000000</td>
<td>338.5000000</td>
</tr>
<tr>
<td>1951:01</td>
<td>319.9000000</td>
<td>327.7000000</td>
<td>334.4000000</td>
<td>338.5000000</td>
<td>341.1000000</td>
<td>341.3000000</td>
<td>347.0000000</td>
<td>359.2000000</td>
</tr>
</tbody>
</table>

FORTRAN Format

FORTRAN format (Additional Topics PDF, Section 3.9) is very similar to free-format with a picture code. 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 Aw format, where 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 ORG=ROWS with DATES, there is a special option ACROSS which you must set correctly:

ACROSS=number of entries per line [4]

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 FORMAT="(A8,8F8.1)", ACROSS=8, DATES.

<table>
<thead>
<tr>
<th>Year</th>
<th>Data 1</th>
<th>Data 2</th>
<th>Data 3</th>
<th>Data 4</th>
<th>Data 5</th>
<th>Data 6</th>
<th>Data 7</th>
<th>Data 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1947:01</td>
<td>224.9</td>
<td>229.1</td>
<td>233.3</td>
<td>243.6</td>
<td>249.6</td>
<td>257.1</td>
<td>264.0</td>
<td>265.5</td>
</tr>
<tr>
<td>1949:01</td>
<td>260.1</td>
<td>256.6</td>
<td>258.6</td>
<td>256.5</td>
<td>267.4</td>
<td>276.9</td>
<td>294.5</td>
<td>305.9</td>
</tr>
<tr>
<td>1951:01</td>
<td>319.9</td>
<td>327.7</td>
<td>334.4</td>
<td>338.5</td>
<td>341.1</td>
<td>341.3</td>
<td>347.0</td>
<td>359.2</td>
</tr>
<tr>
<td>1953:01</td>
<td>365.4</td>
<td>368.8</td>
<td>367.8</td>
<td>362.6</td>
<td>362.0</td>
<td>361.8</td>
<td>366.2</td>
<td>375.0</td>
</tr>
</tbody>
</table>
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=free, org=cols)

writes the same series to a text file using free 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 oeecdg7.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
Prints 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 Autocorrelations on the Time Series menu.

**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** \[\min\left(\frac{T}{4},2\sqrt{T}\right)\]
  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 some other software). See “"Yule–Walker vs Burg Methods" on page RM–538.

- **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, respectively. These are all saved in entries 1 through number+1, with the 0 lag value in entry one. (The RESULTS option was called CORRELATIONS before version 7.)
**stderrs=series for standard errors [none]**

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

**[center]/nocenter**

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

**qstats/[noqstats]**

**span=width of test intervals [all]**

**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.

**[print]/noprprint**

**picture=picture code for output**

**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 "Picture Codes" on page RM–545 for more information. Use the TITLE option if you want to provide your own title for the output. By default, this will be “Autocorrelations (covariances) of Series xxxx”

**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.

**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.

**Notes**

All the output series (RESULTS, PARTIALS, INVERSE and STDERRS) have the “0” lag in entry 1. While this has value 1 for the first three of these (and zero for STDERRS) unless you use the COVARIANCE option, it makes it easier to do graphs since the difference between (for instance) the theoretical pattern of an AR(1) and ARMA(1,1) depends upon the shape when including the 0 lag.
Correlate

There are other instructions you can use, directly or indirectly, to compute autocorrelations. In general, these will produce somewhat different results than CORRELATE, which uses methods specifically designed for autocorrelations. For instance 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 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.

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.

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

$$\Phi(L)y_i = \Theta(L)u_i,$$

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

$$\Theta(L)z_i = \Phi(L)u_i,$$

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.

Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations (<strong>INTEGER</strong>)</td>
</tr>
<tr>
<td>%QSTAT</td>
<td>The test statistic for the (final) $Q$ test (<strong>REAL</strong>)</td>
</tr>
<tr>
<td>%QSIGNIF</td>
<td>The significance level of the (final) $Q$ test (<strong>REAL</strong>)</td>
</tr>
<tr>
<td>%NDFO</td>
<td>degrees of freedom for (final) $Q$ test (<strong>INTEGER</strong>)</td>
</tr>
</tbody>
</table>
Correlate

Examples

It is usually easiest to examine correlations by graphing them. The quickest way to do that is to use the @BJIDENT procedure included with RATS. 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 MAX=1.0, MIN=-1.0 and NUMBER=0 on your GRAPH. The NUMBER=0 option causes the “time” axis to be labeled with 0 to number. Below is a simple example. See the code in BJIDENT.SRC and other procedures for more complex examples.

\begin{verbatim}
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
\end{verbatim}

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.

\begin{verbatim}
correlate(corrs=corrd,partial=partd,qstats,span=4,org=column) dgdp
\end{verbatim}

\begin{verbatim}
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>...</td>
<td></td>
<td></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>0.012290</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>
\end{verbatim}
CPRINT: Printing Complex Series

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

```plaintext
    cprint( options ) start end cseries list
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Range of entries to print. By default, the range required to display all defined entries of the series printed.</td>
</tr>
<tr>
<td>end</td>
<td>List of complex series to print. If you omit the list, RATS prints all of the complex series stored in memory.</td>
</tr>
<tr>
<td>cseries list</td>
<td>List of complex series to print. If you omit the list, RATS prints all of the complex series stored in memory.</td>
</tr>
</tbody>
</table>

Description

CPRINT 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.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval=</td>
<td>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.</td>
</tr>
<tr>
<td>lc=</td>
<td>LC (Left Column) indicates how you want to label the entries. CPRINT assumes entry one is frequency zero when it computes the PERIODS and FREQUENCIES.</td>
</tr>
<tr>
<td>length=</td>
<td>Sets the number of ordinates upon which the frequencies are based for computing the left column labeling. The base number should be the Fourier transform length if you haven’t already reduced the frequency count using CSAMPLE, or the transform length divided by the sample rate if you did use CSAMPLE.</td>
</tr>
<tr>
<td>number=</td>
<td>If you use NUMBER, CPRINT labels the first entry it prints with the indicated number, and then numbers the other entries successively.</td>
</tr>
</tbody>
</table>

---

RM–58  
RATS Reference Manual
**picture=picture code [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 code 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 "Picture Codes" on page RM–545 for more information.

**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**

```plaintext
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>

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.

```plaintext
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 code limiting the display to four digits to the right of the decimal.
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 Cross Correlations on the Time Series menu to do these computations.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>series1</td>
<td>series2</td>
</tr>
<tr>
<td>start</td>
<td>end</td>
</tr>
</tbody>
</table>

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 (or, optionally, the cross-covariances) of series1 and series2. Note that the “lag” refers to the number of periods by which series2 lags series1, that is, the correlation at lag k is the correlation between series1(t) and series2(t-k).

The @CROSSCORR procedure uses the CROSS instruction to compute and graph cross-correlations. You’ll probably find it better to use that than CROSS itself.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from=lowest lag number</td>
<td>These specify the range of lags for which CROSS will compute the cross correlations. Represent leads as negative lags. FROM defaults to –n and TO defaults to n, where n is (\min\left(T/4, 2\sqrt{T}\right)).</td>
</tr>
<tr>
<td>to=highest lag number</td>
<td></td>
</tr>
<tr>
<td>covariances/[nocovariances]</td>
<td>If you use the option COVARIANCES, CROSS computes the cross-covariances rather than the cross-correlations of the series.</td>
</tr>
<tr>
<td>results=series for computed cross correlations/covariances [none]</td>
<td>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 FROM=-10 and TO=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 before version 7).</td>
</tr>
</tbody>
</table>
qstats/[noqstats]
span=width of test intervals [all]
dfc=degrees of freedom correction for test [0]

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 width (subject to your choices of FROM and TO), and increasing the test span by width with each set. Use DFC to correct the degrees of freedom of the $Q$ statistic.

[center]/nocenter
Use NOCENTER if you want the correlations to be computed without subtracting means from the data.

[print]/noprint
picture="picture code for output"
title="Title for output" ["Cross-correlations of xx and yy"]

Unless you use NOPRINT, CROSS displays the computed statistics. PICTURE allows you to control the formatting of the numbers in the output table—see "Picture Codes" on page RM–545. 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”

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.

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. 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

%NOBS Number of observations (integer)
%QSTAT The test statistic for the (final) $Q$ test.
%QSIGNIF The significance level of the (final) $Q$ test.
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 \texttt{SALES} and \texttt{LEAD}. It also performs $Q$ tests on the correlations.

Output

\begin{tabular}{ll}
\textbf{Lag} & \textbf{Cross ACF} \\
-8 & 0.04854 \\
-7 & 0.14119 \\
-6 & 0.04364 \\
-5 & 0.10842 \\
-4 & 0.10449 \\
-3 & 0.72007 \\
-2 & -0.38029 \\
-1 & 0.07092 \\
0 & -0.00317 \\
1 & 0.09698 \\
2 & -0.05844 \\
3 & 0.05464 \\
4 & -0.02955 \\
5 & 0.06766 \\
6 & -0.10622 \\
7 & 0.00208 \\
8 & 0.09510 \\
\end{tabular}

\begin{tabular}{llll}
\textbf{Ljung-Box Q-Statistics} & \\
\textbf{Lag Range} & \textbf{Statistic} & \textbf{Signif Lvl} \\
1 to 8 & 6.482 & 0.593389 \\
-8 to -1 & 110.167 & 0.000000 \\
-8 to 8 & 116.651 & 0.000000 \\
\end{tabular}
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 (CORR)** 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 \( M_1 \) to \( M_2 \) 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 . . .**

- **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

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 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 mainly use special purpose instructions like *CMULTIPLY* and *CACCUMULATE*. *CSET* sets the values of a complex series using a function.

```
cset( options ) cseries start end = 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 start to 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 FREQUENCY. As t runs from 1 to N, this gives the appropriate Fourier transform for the lag operator \( L^t \).
Options

**first=complex expression/FRML**

special value or FRML (if the value of start affects the calculation) for entry start. Use this only if the first entry uses a different calculation from the others. (This is not common in frequency domain calculations).

**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.

**smpl=standard SMPL option** ("SMPL Option" on page RM–546) [not used]

**startup=expression/FRML evaluated at period <<start>>**

provides an expression which is computed only for the first entry of the range, and before function(T) is computed. This can be a FRML or expression of any type.

Examples

cset 3 = %z(t,3)*%z(t,4)
cset 3 = %z(t,3)*%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*%pi*(t-1)/400.))
cset 3 = %cmplx(cos(2*%pi*(t-1)/400.),sin(2*%pi*(t-1)/400.))

Any of these makes entry t of series 3 = exp\(2\pi i(t-1)/400\)

cset 5 = c=%cmplx(1.0,0.0),%do(i,1,4,c=c+b(i)*%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(scratch) 3 = %z(t,3)/%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

UTOR 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.

\[
\text{ctor ( options ) start end newstart}
\]

# list of complex series
# list of real series

**Parameters**

- **start**  **end**
  Range of entries to transfer. This defaults to the (possibly different) 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.
Examples

```r
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.

```r
compute nords=2*%freqsize(2018:6)
freq 3 nords
rtoc 1960:1 2018:6
#  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 2018:6
#  1  2
#  xadj yadj
```

See Also...

```r
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 7.5, for most of the technical details.

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

\[ \begin{align*}
    Au_i &= Bv_i \quad & (5) & \quad E(v_i v_i') = D \quad & D \text{ diagonal} \\
    E(u_i u_i') &= V \quad & (6)
\end{align*} \]

or the V matrix in

\[ (6) \quad E(u_i u_i') = V \]

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, and V as a FRML[SYMM].

\[
\text{cvmodel(options)} \quad \text{sigma} \quad \text{AFRML} \quad \text{BFRML}
\]

Parameters

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

\[
\begin{align*}
    \text{sigma} & \quad \text{The SYMMETRIC covariance matrix of the residuals u.} \\
    \text{AFRML} & \quad \text{A FRML[RECT] which gives the A matrix in (1). We would recommend using the newer A option instead.} \\
    \text{BFRML} & \quad \text{A FRML[RECT] which gives the B matrix in (1). We would recommend using the newer B option instead.}
\end{align*}
\]

Options

\[
\begin{align*}
    \text{a=} & \quad \text{A formula producing RECTANGULAR array} \\
    \text{b=} & \quad \text{B formula producing RECTANGULAR array} \\
    \text{v=} & \quad \text{V formula producing SYMMETRIC array to model covariance matrix} \\
    \text{factor=} & \quad \text{resulting factoring matrix for sigma}
\end{align*}
\]

Use these to provide formulas for the A and B matrices in (1). If you use these, don’t use the AFRML and BFRML parameters.

Use this to supply a formula that models the covariance matrix itself in (2). Omit the A and B formulas when using this option.

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.
**parmset**=**PARMSET** to use [default internal]

This option selects the parameter set to be estimated (see the User’s Guide, page UG–129 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]
**dvector**=(output) VECTOR of values for diagonal of D matrix

The first three select the specific form for the maximand. The **DMATRIX** option determines the treatment of the “D” matrix in (1), whether it’s concentrated out, set to the identity. You can use the **DVECTOR** option to capture the estimated values for this. The **DFC** option is used when you’re maximizing the posterior density. It gives the value of c in the formulas in Section 7.5.2 of the User’s Guide. The **PDF** option sets d, which is used for the posterior density with D integrated out.

**method**=[**bfgs**]/simplex/genetic/annealing/ga/evaluate
**iterations**=iteration limit [100]
**subiterations**=subiteration limit [30]
**cvcrit**=convergence limit [.00001]
**trace**/[**notrace**]

These control the non-linear optimization. **BFGS** is Broyden, Fletcher, Goldfarb and Shanno and is the only method which computes standard errors. However, it is very sensitive to initial guess values, so you might need to use a **PMETHOD** with one of the other methods first. **EVALUATE** simply evaluates the function given the initial values—by default, no output is displayed

**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 4 for more details.

**pmethod**=**bfgs**/[**simplex**]/genetic/annealing/ga
**piters**=number of **PMETHOD** iterations to perform [none]

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 4.5 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 “Sample Output” is from this example.

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

```
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|$
         ,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 ,mr ,0.0,1.0,mp |$
0.0,pc ,pr ,0.0,0.0,1.0||
compute ur=cr=rx=mc=mr=mp=pc=pr=0.0

cvmodel(method=bfgs,factor=afactor,a=afrml) %sigma
```

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=vf,obs=tdim) rr
```
Sample Output

This is the output from the first example above. 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.

<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>RX</td>
<td>0.096444135</td>
<td>0.036726886</td>
<td>2.62598</td>
<td>0.00863994</td>
</tr>
<tr>
<td>UR</td>
<td>-0.099392208</td>
<td>0.065294651</td>
<td>-1.52221</td>
<td>0.12795633</td>
</tr>
<tr>
<td>CU</td>
<td>-0.504940691</td>
<td>0.099773258</td>
<td>-5.06088</td>
<td>0.00000042</td>
</tr>
<tr>
<td>CR</td>
<td>0.146654165</td>
<td>0.069452874</td>
<td>2.11156</td>
<td>0.03472389</td>
</tr>
<tr>
<td>PC</td>
<td>-0.226894244</td>
<td>0.227920839</td>
<td>-0.99550</td>
<td>0.31949517</td>
</tr>
<tr>
<td>PR</td>
<td>0.306165680</td>
<td>0.167011497</td>
<td>1.83320</td>
<td>0.06677265</td>
</tr>
<tr>
<td>MP</td>
<td>-0.121270417</td>
<td>0.044380899</td>
<td>-2.73249</td>
<td>0.00628573</td>
</tr>
<tr>
<td>MC</td>
<td>0.204903072</td>
<td>0.096649532</td>
<td>2.12006</td>
<td>0.03400075</td>
</tr>
<tr>
<td>MR</td>
<td>0.040662013</td>
<td>0.071883647</td>
<td>0.56566</td>
<td>0.57162200</td>
</tr>
</tbody>
</table>

Notes

These models can be quite difficult to estimate successfully (see Section 7.5.2 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 PMETHOD=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 or log posterior at the final estimates
%NVAR  The number of variables (INTEGER)
%NFREE  The number of free parameters (including diagonal if concentrated/marginalized) (INTEGER)
CXTRENUM: Extreme Values for Complex Series

CXTRENUM 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.

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

Parameters
- **cseries**: Complex series whose extreme values you want to locate.
- **start** and **end**: Range of entries to search.

Options
- **part=[real]/imaginary/absvalue**: The part of the complex numbers (real, imaginary, or absolute value) whose extreme values you want.

- **[PRINT]/NOPRINT**
  - **TITLE="title for output" ["Extreme Values of the yyy Part for Series xxx"]**
  - NOPRINT suppresses the output. Use NOPRINT if you want to use the variables described below, but you don’t need the printed output. If you use the default of PRINT, the TITLE option can be used to provide a more descriptive title.

Variables Defined by CXTRENUM
- **%MAXIMUM**: maximum value (real)
- **%MINIMUM**: minimum value (real)
- **%MAXENT**: entry at which maximum was found (integer)
- **%MINENT**: entry at which minimum was found (integer)

Output

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

| Extreme Values of Absolute Value of Series No Label(3) | Minimum Value is 0.0000000000000 at Entry 128 | Maximum Value is 0.0931383026983 at Entry 80 |
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 *Introduction* 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/Graphics menu) generate the appropriate sequence of `CALENDAR`, `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 standard workspace. If you have not set that, and the length of the data set is clear from the contents of the data file, `DATA` uses that and sets it as the workspace length.

- **`list of series`**  The list of series names or numbers you want `DATA` to read from the file. If you omit the list, `DATA` reads all of the series on the file. You can take advantage of this with any formats except `FREE`, `BINARY` or `FORTRAN` format (as they have no labels).

  You can use `series<<fileseriesname` parameter fields to map a series/column name on an original file to a (valid) RATS name. This can be used to handle source file names which are too long, improperly formatted (for instance, having spaces), conflict with RATS reserved names (such as `INV` or `T`), or simply not easily understood. If fileseriesname has spaces or punctuation, enclose it in quotes (single or double). This was added with RATS 9.2.

Options

- **`format=[free]/binary/cdf/citibase/crsp/dbf/dif/dri/dta/fame/fred/haver/matlab/odbc/portable/prn/rats/tsd/wf1/wks/xls/xlsx/'(FORTRAN format)']`**  This describes the format of your data set. 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 of the *Introduction*.

- **`organization=columns/[rows]/multiline`**  This tells `DATA` whether the data series run down the file in columns (one series per column) or across it in rows. The `ORG` option isn’t needed for most formats, since only text files and spreadsheets allow for the different arrangements.
ORG=MULTILINE can be used to read data for a single series (only) which spans more than one row in a spreadsheet (or similar file format). If you have more than one series on a file blocked this way, use separate DATA instructions with the TOP and BOTTOM options to isolate the information for each series.

Note that the older terminology of ORG=VARIABLE/OBSERVATION is still supported. OBSERVATION corresponds to COLUMNS and VARIABLE corresponds to ROWS.

**unit=[data]/input/other unit**

Chooses the RATS I/O unit from which the data are read. With the default option, DATA reads the data from the external data file. Use UNIT=INPUT when you want to enter data directly into the input file.

**sheet="worksheet or matrix name"**

When reading an Excel file with multiple worksheets or a MATLAB file with several data series on a single matrix, you can use SHEET to identify the which you want to read. DATA reads the first by default.

**skiplines=number of top lines to skip [0]**

**top=top line to process [1]**

**bottom=top line to process [last]**

**left=leftmost column to process [1]**

**right=rightmost column to process [last]**

**[labels]/nolabels**

You can use these if you have comments or other non-data information on a text or spreadsheet file. SKIPLINES or TOP can be used to skip over information at the top of the file, and the others can be used to prevent reading marginal comments at other locations. If you have a spreadsheet file with non-standard (or no) labels, you can use NOLABELS to indicate that the series labels aren’t on the file, but will be provided by the list of series.

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

**select=subperiod**

These options, which work with most dated file formats, set the method DATA will use to convert data from higher frequency to lower, such as monthly to quarterly. See Section 2.5 of the Introduction.

**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 Introduction, Section 2.6, for more information.
blank=[zero]/missing
This applies only with FORTRAN formats. If an area where DATA expected a number is blank, it treats it either as a zero or as a missing value, depending upon your choice for this option.

julian=SERIES to fill with Julian date information from file [not used]
This creates a series from the date information on the file which has the mapping from the dates to the Julian date coding used by RATS. This can be used to create a mapped irregular date scheme.

calendar=Coded CALENDAR data array [not used]
In general, DATA does a fairly good job of detecting the frequency and dates of the information on a data file which has date information. However, if you have a very short data set, or a data set with large gaps, it might fail to determine the date scheme that you intend. The CALENDAR option allows you to force a particular (encoded) date scheme. The %CALENDAR() function can be used to take the current CALENDAR scheme and save the information into a VECT[INTEGER]. The most typical use would be

DATA(CALENDAR=%CALENDAR(),other options) ...

which forces the data to be interpreted as having the CALENDAR scheme of the current workspace.

reverse/[noreverse]
If your data are in time-reversed order (starting with the most recent at the top or left), use REVERSE to have it switched to the standard time series order when read in. Note that if the file has usable dates which run in the reversed order, DATA will automatically detect that and correct the ordering, so this is only necessary if you have a data set with reversed sequence and no usable dates.

verbose/[noverbose]
Use VERBOSE if you want DATA 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 DATA is reading date information on your data file properly—this is particularly important when converting from one frequency to another.

sql="SQL command to read data"
query=input/Other unit
When using FORMAT=ODBC, use one of these options to provide a SQL query to read your data. For a relatively short query (255 characters or fewer), you can use the SQL option, supplying the query as a literal string or as a variable of type STRING. For a longer SQL query, use the QUERY option.

With QUERY=INPUT, RATS reads the SQL commands from the lines following the
**DATA** instruction in the input window (or input file). With **QUERY=unit**, RATS reads the query from the text file associated with the specified I/O unit (opened previously with an **OPEN** instruction). In either case, use a “;” symbol (which tells RATS to begin a new instruction) to signal the end of the SQL string.

See **OPEN** for more on I/O units, and Section 3.15 of the *Additional Topics* for more on using SQL commands to read data.

```plaintext
dateformat="date format string"
This can be used if dates on the file are text strings in a form other than “year (delimiter) month (delimiter) day”.

In the date format string, use \( y \) for positions providing year information, \( m \) for positions providing month information, and \( d \) for positions with day information. Include the delimiters (if any) used on the file. For example:

```plaintext
dateformat="mm/dd/yyyy"
dateformat="yyyyymmdd"
```

### Examples

```plaintext
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.

```plaintext
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.

```plaintext
cal(a) 1920:1
all 1945:1
open data klein.prn
data(org=col,format=prn) 1920:1 1941:1 consumption $
    profit privwage invest klagged production govtwage govtexp taxes
```

reads data for nine series in labelled ASCII (PRN) format over the range 1920:1 to 1941:1.

```plaintext
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.

<table>
<thead>
<tr>
<th>DBox(action=define)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dbox</strong> <em>(options)</em> <em>output value</em></td>
</tr>
<tr>
<td><strong>Dbox(action=run)</strong></td>
</tr>
</tbody>
</table>

**Parameters**

- **output value**

  For most dialog box elements, **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 types of items you can include within a DBOX. Some of these are shown below (see "Item Type Options" 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.

```
dbox(action=define,title="Sample Static Text")
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.

```
declare string result
dbox(action=define,title="Sample Edit Text")
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.

```
declare integer on
dbox(action=define,title="Sample Check Box")
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.

```
declare integer button

dbox(action=define,title="Sample Radio Buttons")
  dbox(radio="Button 1",default=1) button
  dbox(radio="Button 2")
  dbox(radio="Button 3")
  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.

![Sample Popup Box](image)

```plaintext
declare integer seriesnum
dbox(action=define,title="Sample Popup Box")
dbox(stext="A Popup 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.

![Sample Combo Box](image)

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

A list of choices in a scrolling box. If \textit{output value} is an \texttt{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 \textit{output value} is a \texttt{VECT[INTEGER]}, the user can select any number of the choices from zero up to all.

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

\section*{General Options}

\texttt{action=define/modify/run}

As mentioned above, you start with \texttt{ACTION=DEFINE} and end with \texttt{ACTION=RUN}. Since \texttt{ACTION=MODIFY} is the default, you don’t need an \texttt{ACTION} option when defining the items in the dialog box.

\texttt{title="dialog title" (only with \texttt{ACTION=DEFINE}) ["User Dialog"]}

This provides the title at the top of the dialog.

\texttt{status=integer variable set to 0,1 status (only with \texttt{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 \texttt{STATUS} option.
**default**\(^=\) default (integer) value [see below]

_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**\(^=\) number of character positions [see below]

**height**\(^=\) number of items to show [see below]

_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**\(^=\) sameline/gap  [not used]

**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**/[nofocus]

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

**initialize**/[noinit]ialize]

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**/[nooptional]

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**\(^=\) expression to verify data [no check]

**disable**\(^=\) expression to disable an item

**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.
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 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.

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 code 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 "Picture Codes" on page RM–545 for more information.
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.

**mselect=rows/one/byrow/bycol**

**select=VECTOR[INTEGER] of selections**

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

You use **SELECT** to control how selections are made. With **MSELECT=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 **MSELECT=ONE**, it will have dimension 2 with `array(1)=row` and `array(2)=column`. If **MSELECT=BYROW**, the user selects one cell per row. The **SELECT** array will have dimension equal to the number of rows, with `array(i)=column` set equal to the column selected in row i. **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.

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

Wizard

The *Statistics—Limited/Discrete Dependent Variables* operation provides dialog-driven access to most of the features of the **DDV** instruction.

Parameters

- **depvar**: Dependent variable. **DDV** requires numeric coding for this. The coding required will depend upon the model type. See the description under the **TYPE** option.
- **start** and **end**: Estimation range. If you have not set a **SMPL**, this defaults to the maximum common range of all the variables involved.

Options

- **type=[binary]/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 for analyzing differing attributes across choices, with a fixed set of coefficients. See Section 12.2 of the User’s Guide for more information.

  For **BINARY**, one choice is represented (in **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=(output) 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 a K \times (m–1) matrix of coefficients where K is the number of regressors. The first choice is normalized to the zero vector and left out.

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

[print]/noprint
vcv/[novcv]
smpl=SMPL series or formula ("SMPL Option" on page RM–546)
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 4 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 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. 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 1.4 in the Additional Topics PDF for details.

weight=series of entry weights ("WEIGHT option" on page RM–549)
Use this option if you want to provide different weights for each observation.

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

\[ L = \sum_{Y_i=0} \log\left(F(X_i\beta)\right) + \sum_{Y_i=0} \log\left(1 - F(X_i\beta)\right) \]

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

\[ \left( -\frac{\partial^2 L}{\partial \beta^2} \right)^{-1} \text{ evaluated at } \hat{\beta}. \]

**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,...,\( m \). Don’t include \textsc{constant} among your regressors; it would merely shift the cut points. The likelihood elements are generated from:

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

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 \textsc{indiv} and \textsc{modes} options to tell DDV which series they are. The \textit{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 \textit{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 \textsc{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(2010), section 18.2), 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 ma-
trix is reasonable.

Examples

\[
ddv(smpl=(\text{kids}==0)) \text{lfp}
\]

# constant wa agesq faminc we

estimates a binary choice probit for the observations where \text{KIDS} is zero.

\[
ddv(\text{dist=logit}) \text{ union}
\]

# potexp exp2 grade married high constant

prj(\text{dist=logit}, \text{cdf=fitlgt})

estimates a binary choice logit, and uses \text{PRJ} to get the predicted probabilities from
the model.

\[
ddv(\text{type=m}ulti\text{n}omial, \text{smpl=y87}) \text{ status}
\]

# educ exper expersq black constant

estimates a multinomial logit.

\[
ddv(\text{type=count}, \text{robusterrors}) \text{ children / resids}
\]

# educ age agesq evermarr urban electric tv constant

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

Output

\text{TYPE=}\text{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 num-
ber of individuals, \(y_i\) is the observed value):

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

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

\[
\text{Base Likelihood} \quad \log L_c \equiv \sum_{j=1}^{m} N \hat{p}_j \log(\hat{p}_j)
\]

\[
\text{Pseudo-}R^2 \quad 1 - \left( \frac{\log L}{\log L_c} \right)^{(2/N)\log L_c}
\]
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)
- `%NFREE` number of free parameter in model (INTEGER)
- `%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 on page UG–74 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.

```plaintext
d 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 `SYMBOLS` option helps identify variables in procedures and user-defined functions that are defined as global, rather than local, variables.

  Execute the `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 `SOURCE` instruction to read them in.

  RATS will display a list of any global variables found in the procedure or function. You can use `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.

\[
\text{declare datatype list of names (and/or dimension fields)}
\]

Parameters

- **datatype**: 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 `datatype`. 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 zFRML[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,

```plaintext
compute [symmetric] s=tr(a)*a
```

can be used instead of

```plaintext
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.
Data Types Available

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

Aggregate Types:
  VECTOR, RECTANGULAR, SYMMETRIC, PACKED, SERIES, FRML, HASH, LIST

The aggregate types can be layered as deep as you need. You can have a VECTOR of FRMLs (written VECTOR[FRML]), a RECTANGULAR of SERIES of VECTORS (written as RECTANGULAR[SERIES[VECTOR]]), and so on. These are all for real numbers by default, but you create things like a VECTOR of COMPLEX numbers (VECTOR[COMPLEX]) or a FRML of STRINGS (FRML[STRING]). See the User’s Guide, page UG–16 for details.

Examples

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

The statements above 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)

```plaintext
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

While there is no instruction which can “undeclare” a variable, you can manually delete variables from memory by using the View—All Symbols operation, selecting the variable(s), and doing Edit–Delete (or Right+Click then Delete). This can be handy if you are writing a program and get a type wrong.

Because of their special nature, you cannot delete SERIES except by doing File–Clear Memory, which clears everything.
Dedit

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. (Some of the less-used of these instructions are now covered in the *Additional Topics* PDF). Use **SAVE** to save any changes to the file, or **QUIT** to close the file without saving your changes.

```plaintext
dedit( option )    filename
```

**Wizards**

The **Open** and **New** operations on the **File** menu are an alternative way to open or create a RATS format data file. There are then other operations which allow you to view, edit, and create data series, copy data from one file to another, export data, and more. This is now the preferred way to handle RATS format files. See Section 2.7 in the *Introduction*.

**Parameters**

```plaintext
file name
```

Name of the data file you want to edit or create (can include drive and path information). If you simply type **DEDIT**, RATS will prompt you for the file name.

**Option**

```plaintext
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=filename** 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 (or the **Data Wizard (RATS Format)** operation) 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

```plaintext
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 ...

- **Intro. Section 2.7** RATS format data files
- **QUIT** Closes RATS file being edited without saving any changes.
- **SAVE** Saves the changes made to a RATS format file.
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.

\[ \text{density}(\text{options}) \quad \text{series} \quad \text{start} \quad \text{end} \quad \text{grid} \quad \text{density} \]

**Wizard**

The Statistics—(Kernel) Density Estimation operation provides access to the major features of DENSITY.

**Parameters**

- **series**
  - The series for which you want to compute the density function.

- **start 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” on page RM–100.

- **bandwidth=kernel bandwidth**
  - BANDWIDTH specifies the bandwidth for the kernel. The default value is \( 0.79 \, N^{-1/8} \, IQR \)

  - 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 13.1 of the User’s Guide), the grid series is usually a series created from the data.
Density

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.

weight=series of entry weights (“WEIGHT option” on page RM–549)
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 which matches up, point for point, with the grid and density series. It requires TYPE=GAUSSIAN or TYPE=LOGISTIC—the other kernels aren’t differentiable.

smoothing=smoothing scale vector [1]
You can supply a real value (bigger than 0) to adjust the amount of smoothing. Use a value bigger than 1 for more smoothing than the default, values less than 1 for less smoothing.

smpl=SMPL series or formula (“SMPL Option” on page RM–546)
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]
PRINT 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:

\[
\hat{f}(u) = \sum_{i=1}^{T} \frac{w_i K \left( \frac{u-x_i}{h} \right)}{h \sum_{i=1}^{T} w_i}
\]

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 \( K(v) = 0.75(1-v^2) \) if \( |v| \leq 1 \), and 0 otherwise

TRIANGULAR \( K(v) = (1-|v|) \) if \( |v| \leq 1 \), and 0 otherwise

GAUSSIAN \( K(v) = \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{v^2}{2} \right) \)
Density

\[ K(v) = \frac{e^v}{(1+e^v)^2} \]

\[ K(v) = 0.5 \text{ if } |v| \leq 1, \text{ and } 0 \text{ otherwise} \]

\[ K(v) = \frac{4}{3} - 8v^2 + 8|v|^3 \text{ if } |v| \leq 0.5, \frac{8(1-|v|^3)}{3} \text{ if } 0.5 \leq v \leq 1, \text{ and } 0 \text{ otherwise} \]

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 becomes a series of “bins” centered at each grid point. DENSITY counts the number of data points which fall in each bin. If you use COUNTS, these raw counts will be the values returned. Otherwise, the counts are divided by the number of data points times the bin width 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 an Epanechnikov kernel density estimate and a histogram, and graphs these along 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(smoothing=1.5) 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.

```
deterministic  list of deterministic variables
```

Wizard

The Time Series—VAR (Setup/Estimate) wizard provides an easy, dialog-driven interface for defining and estimating VAR models.

Parameters

```
list of ...  Lists the variables that you want to include in the equations.
Use regression format (see page Int–79 of the Introduction).
```

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 GDPDEF and M2 as exogenous variables in the unrestricted model, allowing us to test the hypothesis that they do not enter the equations for GDPQ, UNEMP, and GDPI.

Notes

The ECT instruction is available as a subcommand of SYSTEM to define error correction terms. In some older programs, you might find these put into the system using DETERMINISTIC.

See Also ...

User’s Guide, Ch. 7  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.

\[
\text{difference}\left(\text{options}\right)\quad \text{series start end newseries newstart}
\]

Wizard

The Data/Graphics—Differencing wizard provides an easy, dialog-driven interface to the differencing capabilities of the DIFFERENCE instruction.

Parameters

- **series**: Series to difference.
- **start 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 “Description”, page RM–104]
- **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.

- **fraction=d (fractional differencing parameter)** [unused]
- **padding=padding value for out-of-sample observations** [0.0]

  To do fractional differencing, use the FRACTION option (you can use D for the option name as well) 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 14.10 of the User’s Guide to see how the differenced series is calculated.
**Difference**

`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 \( Y_t = \left( X_t - \bar{X} \right)/s \) to create a series with mean 0 and variance 1. 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.

**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, \( \text{SPAN}=12 \) for monthly data). For frequencies like weekly and daily, where there is no clear definition of a seasonal span, \( \text{SPAN} \) defaults to 1.

**Examples**

```plaintext
data 1959:1 2009:12 mlnsa
set mlnsa = log(mlnsa)
diff mlnsa / m1ldiff
diff(sdiffs=1) mlnsa / m1lsdiff
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.

```plaintext
diff(fraction=d, pad=uupresample) uu / uufilter
```

does fractional differencing on series UU producing UUFILTER. The out-of-sample values are treated as having the value UUPRESAMPLE.

```plaintext
diff(center) sunspots / cspots
```

removes the mean from SUNSPOTS to create CSPOTS.
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)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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**

**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(dims)  (separate multiple arrays by blanks)
```

**Parameters**

`array(dims)`  
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 and the arrays dimensioned do not have to be of the same type.

**Dimension fields**

A dimension field has one of the forms:

- `name(dimension)`  for one-dimensional arrays (`VECTOR` arrays).
- `name(dim1,dim2)` for the two-dimensional arrays: `RECTANGULAR`, `RECenter`, and `SYMMETRIC`, and `PACKED`. This sets up an $dim1 \times dim2$ matrix. ($dim2$ is ignored for `SYMMETRIC` or `PACKED`, since such matrices 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 its previous contents. If you have an array which (possibly) needs to be extended without losing the original information, you might be better off using a `LIST` rather than a `VECTOR`.
DISPLAY — Displays Computed Values

DISPLAY can be used to display the values of scalars, arrays, strings, expressions, and more in the output window, or to a file.

```
display(options) variables, strings, picture/position/tab codes
```

Parameters

**variables**
These can be variables or expressions to be evaluated. You can also display EQUATIONS, MODELS or PARMSETs, and SERIES on (almost) any type, though PRINT is generally much better for working series.

**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—details on the next page. It provides the representation for the next variable printed.

**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.

**delimited=[none]/tab/comma/semicolon**
By default (with the NONE choice), output in a DISPLAY instruction is separated by blank spaces. This option allows you to generate tab, comma, or semicolon-delimited output instead. This works when outputting to the screen, but is primarily useful with the UNIT option to output to a text file.

**store=string variable**
This saves the created line in string variable; it does not display it. You can use this variable, for instance, as a header for a graph, though you can often do the same operation with a COMPUTE instruction.

**width=maximum width of a line [255]**
This controls the width of the content of a line. If the information exceeds that length, one line is finished and a new one started.
hold/[nohold]

If you use **HOLD**, **DISPLAY** creates the string and waits for another **DISPLAY**
instruction to add more to it. This can be useful when the number of objects to
display is not known in advance, but **REPORT** is generally the better choice.

**Picture Codes**

A picture code is a string of characters that provide a template for formatting the
next variable (or variables) displayed. You can use the following characters to con-
struct a picture code:

```
#, *, and .
```

To display an integer value (or a real value rounded to the nearest integer) with a
fixed number of character spaces, use a form like “####”. This tells RATS to display
the number right-justified in a field whose width is the number of # signs.

For real-valued output, use a form like “###.##”. The number of character spaces
used equals the total number of characters in the string, with the decimal at the
indicated location. Use * to the left of the decimal point (“*.##”) if you want RATS to
use only as many places as needed to show the number and no more.

For instance, ###.## would display the value 1.23 with two blanks padding the field
on the left, while *.## will display it using just the four characters.

Using * by itself requests a standard format which generally shows five digits right
of the decimal.

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. %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 ac-
curately 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 field is followed by one blank 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. The instruction below displays 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:

```
display @@.12 *.## xvect
```

Using ?

You can abbreviate the command to simply ? if you aren’t using any options.

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 use a picture code to squeeze out the extra spaces and drop some of the excess digits on the result. This:

```
display "F("+q+","+%ndf +") =" *.### ((rssr-rssu)/q)/(rssu/%ndf)
```

will produce

```
F(13,123) = 3.921
```
Display

```plaintext
display @12 "Method 1" @32 "Method 2"
display @10 ######.#####  t1 @30 t2
```

will display

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.17235</td>
<td>11234.20921</td>
</tr>
</tbody>
</table>

```plaintext
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:

<table>
<thead>
<tr>
<th>Method 1</th>
<th>Method 2</th>
<th>Method 3</th>
<th>Method 4</th>
</tr>
</thead>
</table>

Note that it’s often easier to use `REPORT` rather than complicated `DISPLAY` instructions.

```plaintext
@johmle(lags=6,det=rc,cv=cvector)
# ftbs3 ftb12 fcm7
equation(coeffs=cvector) ecteq *
# ftbs3 ftb12 fcm7 constant
?ecteq
```

estimates a cointegrating vector using `@JOHMLE`, creates an `EQUATION` based upon it, and displays that.

See Also ...

- `REPORT` Organizes output into a table format.
- `WRITE` Displays arrays and other variables. More general, but less flexible, than `DISPLAY`.
- `MESSAGEBOX` Displays information messages requiring user response.
- `INFOBOX` Displays messages, progress bars (no user response required).
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 Chapter 10 in the User’s Guide for more information.

\[ \text{dlm( options ) } \text{ start } \text{ end } \text{ state vectors } \text{ state variances} \]

### Parameters

- **start**  **end**  Range to use in estimation.
- **state vectors**  A SERIES of VECTORS into which the output state vectors are placed. If you call this STATES, for instance, STATES(2014:1) is the complete estimated state vector for 2014:1. To get component \( k \) of this, use STATES(2014:1)(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 consists of two equations:

1. \[ X_t = A_t X_{t-1} + Z_t + F_t W_t, \]
2. \[ Y_t = \mu_t + C_t' X_t + V_t \]

\( X_t \) is an unobserved vector of states. \( Y_t \) is observable, and gives information about \( X_t \) through the measurement equation (2). \( W_t \) 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; often the identity matrix. The \( Z_t \) term in (1) allows for exogenous shifts in the state equation, and \( \mu_t \) in (2) allows for exogenous shifts in the measurement equation.

In these descriptions, \( N \) is the size of \( X \), \( M \) is the size of \( Y \), \( L \) is the size of \( W \).

- **a=** RECTANGULAR or FRML[RECT] [identity matrix]
  This gives the A matrices. The expression should evaluate to an \( N \times N \) matrix.

- **c=** RECTANGULAR or FRML[RECT] [zero matrix]
  This gives the C matrices—it should evaluate to an \( N \times M \) matrix.

- **y=** VECTOR or FRML[VECT] [no measurement equation]
  This gives the Y vectors—it should evaluate to an \( M \) vector.
$z=\text{VECTOR or FRML}[\text{VECT}]$ [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.

$\mu=\text{VECTOR or FRML}[\text{VECT}]$ [zero]
This gives the $\mu$ vectors, which adds a shift term to the measurement equation—it should evaluate to an $M$ vector.

$sw=\text{SYMMETRIC or FRML}[\text{SYMMETRIC}]$ [zero]
This gives the covariance matrices of the $W$’s—it should evaluate to an $L \times L$ symmetric.

$sv=\text{SYMMETRIC or FRML}[\text{SYMMETRIC}]$ [zero]
This gives the covariance matrices of the $V$’s—it should evaluate to an $M \times M$ symmetric.

$f=\text{RECTANGULAR or FRML}[\text{RECTANGULAR}]$ [$N \times N$ identity]
These are the loadings from the $W$’s to the $X$’s—it should evaluate to an $N \times L$ matrix.

discount=discount value [not used]
Multiplies $\Sigma(t-1) \times \text{discount}$. This is an alternative to using $sw$; instead of the change in variance being additive, it will be multiplicative.

type=[filter]/smooth/control/simulate/csimplify
This determines what technique is to be used. FILTER is the Kalman filter, SMOOTH is the Kalman smoother and CONTROL solves the control problem (see “Options for Optimal Control”). SIMULATE does a random (Normal) simulation of the DLM drawing randomly from the presample distribution, the state disturbances and the measurement equation disturbances. CSIMULATE does a conditional simulation, drawing the states from their distribution conditional on the observed $Y$’s.

save=DLM to save inputs [not used]
get=DLM with saved model to be used[not used]
SAVE saves the information about the inputs to the model (basic input options $A$, $C$, $Y$, $F$, $sw$, $sv$, $\mu$, $z$ and less standard ones $sx0$, $x0$, $q$, $r$, $sh0$ and $\text{DISCOUNT}$). You can then use GET on a later DLM instruction to re-use those inputs. Note that this only applies to the inputs, not the outputs, or to $\text{TYPE}$ options or estimation controls.

smpl=SMPL series or formula (“SMPL Option” on page RM–546)
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 on “Missing Values” concerning missing values.
presample=ergodic/[x0]/x1/diffuse
exact/[noexact]
g=matrix (R\times N, R<N) reducing model to stationarity [not used]
sh0=variance of diffuse part of the prior [identity if EXACT]
x0=VECTOR or FRML[VECTOR] [zero vector]
sx0=SYMMETRIC or FRML[SYMMETRIC] [zero]

These options can be used to initialize the pre-sample states of the Kalman filter. See Section 10.6 of the User's Guide for the technical details.

PRESAMPLE=DIFFUSE (the older EXACT option is a synonym) gives a diffuse prior to the entire pre-sample state vector. PRESAMPLE=ERGODIC computes a (possibly mixed) stationary-diffuse prior, as described in Doan (2010) and is generally the recommended option. G and SH0 are older options for this: 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. X0 supplies the initial state mean—it should evaluate to an N VECTOR while SX0 gives the covariance of X0—it should evaluate to an N\times N SYMMETRIC array. If PRESAMPLE=X1, X0 and SX0 are still used, but they provide X_{1|0} and \Sigma_{1|0}, not X_{0|0} and \Sigma_{0|0}.

\textbf{variance}=[\text{known}]/\text{concentrated}/\text{chisquaredinverse}
\textbf{scale}/[\text{noscale}] (old option—use \text{VARIANCE=CONCENTRATED} instead of \text{SCALE})
\text{VARIANCE=KNOWN} assumes all variances are known (or being estimated).
\text{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. \text{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 \text{PDF} and \text{PSCALE} options below.

When you use \text{VARIANCE=CONCENTRATED} or \text{VARIANCE=CHISQUARED}, you will usually peg one of the variances to 1.0, such as with SV=1.0.

\textbf{pdf}=prior degrees of freedom [not used]
\textbf{pscale}=prior scale factor [not used]
\textbf{vdiscount}=discount factor for degrees of freedom [1.0]
These are used only with \text{VARIANCE=CHISQUARED}. PDF and PScale are the prior degrees of freedom and prior scale factor for the scaling variance, respectively. VDISCOUNT is a suggestion of West and Harrison (1997), which downweights the past information about the variance if the value is less than the default of 1.0.
You can use the `START` option to provide an expression which is computed once per function evaluation, before any of the regular formulas are computed. This allows you to do any time-consuming calculations that depend upon the parameters, but not upon time. It can be an expression of any type. `ONLYIF` calculates the expression provided; if it evaluates to a zero value, the function evaluation doesn't continue, and the function is assigned the missing value. `ONLYIF` is examined before doing the `START` option, unlike `REJECT` (below), which is done after the `START`.

If the expression evaluates to a non-zero ("true") value, the function is immediately assigned the missing value. If you have a `START` option, `REJECT` is examined after it, so it can look at the results for anything computed as part of the `START` option.

This can be used to speed calculations if the system matrices are time-invariant and there are no missing values. This is particularly helpful when the number of observable variables is high.

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.
- `yhat=SERIES[VECT] of predicted or simulated values of Y [not used]`
  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 observable errors [not used]
swhat=SERIES[SYMM] for pred. error cov. matrices [not used]

VHAT 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. SVHAT 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 [not used]
swhat=SERIES[SYMM] of state disturbance variances [not used]

With TYPE=SMOOTH, these give the expected values and variances of the shocks to the states given the full data set. Neither is defined for TYPE=FILTER. With TYPE=SIMULATE or TYPE=CSIMULATE, the WHAT series will be the simulated disturbances; SWHAT isn't defined for the simulations.

gain=SERIES[RECT] of Kalman gain matrices [not used]
Saves the series of Kalman gain matrices.

likelihood=series of cumulated log likelihoods
This saves the (cumulated) log likelihoods into a SERIES.

sighistory=series of estimated scaling variances [not used]
dfhistory=series of degrees of freedom [not used]

If you use VARIANCE=CHISQUARED, SIGHISTORY can be used to get the series of estimated scaling variances, while DFHISTORY returns a series of the sequential degrees of freedom.

Options for Estimating Parameters

method=bfgs/simplex/genetic/annealing/ga/gauss/[solve]
iterations=iteration limit [100]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]

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, which need to be defined ahead of time with NONLIN.

BFGS and GAUSS (Gauss-Newton) are the only two which can compute standard errors; the others can get point estimates only and are more often used as “PMETHODS” to improve guess values. See Optimization Methods in the help.

ITERATIONS sets the maximum number of iterations, SUBITERS sets the maximum number of subiterations, VCICRT 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.
pmethod=bfgs/[simplex]/genetic/annealing/ga/gauss
piters=number of PMETHOD iterations to perform [none]
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 estima-
tion methods—RATS will automatically switch to the “METHOD” choice after com-
pleting 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,
page UG–129, 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

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 PRESAMPLE=ERGODIC or PRESAMPLE=DIFFUSE to deal explicitly with the
diffuse initial conditions.

Options for Optimal Control
For optimal control the model takes the a slightly different form, as there are now
control variables $U_t$. $P$ is the size of $U$.

(3) $X_t = A_t X_{t-1} + B_t U_t + F_t W_t$

(4) $Y_t = \mu_t + C_t' X_t + V_t$, with the objective function being

(5) $E \left[ X_0' Q_0 X_0 + \sum_{i=1}^{T} \{ X_i' Q_i X_i + U_i' R_i U_i \} \right]$

b=RECTANGULAR or FRML[RECT] [zero matrix]
This gives the $B$ matrices. The expression should evaluate to an $N\times P$ matrix.

q=SYMMETRIC or FRML[SYMMETRIC] [zero]
This gives the $Q$ matrices—it should evaluate to an $N\times N$ SYMMETRIC.

r=SYMMETRIC or FRML[SYMMETRIC] [zero]
This gives the $R$ matrices—it should evaluate to a $P\times P$ SYMMETRIC.
Options for Adjusting the State or Covariance Matrix

- `fpre=FUNCTION(VECTOR *,SYMMETRIC *)` called before the prediction step
- `fpost=FUNCTION(VECTOR *,SYMMETRIC *)` called after the filter step
- `fspost=FUNCTION(VECTOR *,SYMMETRIC *)` called after smoothing update

These can be used to intervene in the filtering or smoothing process to either do some added calculations (for, for instance, linearization) or to actually change the state vector and/or the state covariance matrix. If you use `fpre`, it will be called before doing the prediction step, so the state and covariance matrix will be the values at the end of the previous period. `fpost` is called after this period’s Kalman update calculation, so will have the $t|t$ information. `fspost` is called after the smoothing calculation for period $t$, and thus will have $t|T$. In all cases, the reserved variable $T$ will be equal to the entry currently being calculated. For instance, the following will adjust the state vector (here, just a single value) to be equal to 1 if the filtered value exceeds 1.

```r
function WithStdSigma xt vt
type vector *xt
type symm *vt
*
if xt(1)>1.0
    compute xt(1)=1.0
end
*
dlm(fpost=WithStdSigma,y=y,c=1.0,sw=sigsw,sv=sigsv,$
    presample=diffuse,type=filter) / xstates_std vstates_std
```

**Missing Values**

A missing data point can be represented by either a $C_i$ or $Y_i$ 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 $1 \times 1$ 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) \[ y_t - \mu = \phi(y_{t-1} - \mu) + \varepsilon_t + \theta \varepsilon_{t-1} \]

can be defined by the following matrices

(7) \[
\begin{bmatrix}
X_t & = & \begin{bmatrix}
y_t - \mu \\
\varepsilon_t
\end{bmatrix}, A = \begin{bmatrix}
\phi & \theta \\
0 & 0
\end{bmatrix}, C' = \begin{bmatrix} 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 1 \\
1
\end{bmatrix}, W_t = [\varepsilon_t]
\end{bmatrix}
\]

If the variance of \( \varepsilon_t \) is set to one, we can use \texttt{VARIANCE=CONCENTRATE} in estimating this model. Assuming that |\( \phi |<1$, this is a stationary model, which can be initialized using \texttt{PRESAMPLE=ERGODIC}. This sets up and estimates the model. It also estimates the same model using \texttt{BOXJENK} with the \texttt{MAXL} option. This state-space model is exactly what \texttt{BOXJENK} with \texttt{MAXL} does internally.

\texttt{nonlin \ mu \ phi \ theta}
\texttt{dec \ frml[rect] \ af}
\texttt{frml \ af = ||phi,theta|0,0||}
\texttt{sstats(mean) / y>>mu}
\texttt{compute \ phi=theta=.10}
\texttt{dlm(var=concentrate,presample=ergodic,method=bfgs,$}
\texttt{a=af,y=y,mu=mu,c=%unitv(2,1),f=||1.0|1.0||,sw=1.0) 1960:1 2009:4}
\texttt{boxjenk(ar=1,ma=1,constant,maxl) y}

(8) \[ y_t = \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-p} + \varepsilon_t \]

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

(9) \[
\begin{bmatrix}
X_t & = & \begin{bmatrix}
y_t \\
y_{t-1} \\
\vdots \\
y_{t-p+1}
\end{bmatrix}, A = \begin{bmatrix}
\varphi_1 & \varphi_2 & \ldots & \varphi_{p-1} & \varphi_p \\
1 & 0 & \ldots & 0 & 0 \\
0 & 1 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & 1
\end{bmatrix}, F = \begin{bmatrix} 1 \\
0 \\
\vdots \\
0
\end{bmatrix}, W_t = [\varepsilon_t]
\end{bmatrix}
\]

If the coefficients are known and aren't being estimated, \( A \) can be set up with an \texttt{EWISE} instruction with an \texttt{%IF} to handle the first row. Assume that the coefficients are in a vector named \texttt{PHI}.

\texttt{dec \ rect \ a(p,p)}
\texttt{ewise \ a(i,j)=%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 \texttt{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 \texttt{START} option. In the \texttt{DLM} instruction below, this calls \texttt{AFUNC} with the current values of the parameters, puts the
result into the matrix $A$, which is then used by the $A$ option.

```plaintext
compute p=5
dec vect phi(p)
nonlin phi
function afunc phi
  type vector phi
  type rect afunc
  local integer i j
dim afunc(%size(phi),%size(phi))
ewise afunc(i,j)=%if(i==1,phi(j),i==j+1)
end afunc
dlm(var=concentrate,presample=ergodic,method=bfgs,$
  start=(a=afunc(phi)),a=a,y=y,c=%unitv(p,1),$
  f=%unitv(p,1),sw=1.0) 1960:1 2017:4
```

**Variables Defined by DLM**

- **%FUNCVAL, %LOGL**  The log likelihood for TYPE=FILTER or TYPE=SMOOTH, or the value of equation (5) for TYPE=CONTROL (real)
- **%VARIANCE**  If VARIANCE=CONCENTRATED, this is the (maximum likelihood) estimate of the variance scale (real)

The following are only defined when estimating coefficients:

- **%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)
- **%NFREE**  Number of free parameters in the parameter set, plus the variance if it’s concentrated out (INTEGER)
**DO — Simple Loops Over an Index**

A DO loop repeats a set of instructions with an integer-valued index changing in a systematic way. DOFOR is a similar instruction which loops over a list of values. See page UG–468 of the User's Guide for a discussion of the various looping instructions available in RATS.

```plaintext
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 (you can use Edit–Indent Lines to do this). 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.

The value of the index when the loop is exited is the value it had on the last pass through, not the value that causes the loop to terminate.

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 function %DO can be used to put “looping” subcalculations into a larger calculation. For instance,

\[ \text{frml archpart } = v=0.0, \%do(i, 1, p, v=v+b(i)*u(i)^2), v \]

will compute (for a value of T) a sum involving P lags of the series U.

Examples
\[ \text{do ar=0,3} \]
\[ \quad \text{do ma=0,3} \]
\[ \quad \quad \text{boxjenk(diffs=1,constant,ar=ar,ma=ma,maxl)} \] $\$
\[ \quad \quad \text{rate 1975:4 2017:4} \]
\[ \quad \text{end do ma} \]
\[ \text{end do ar} \]

estimates 16 ARIMA models, one for each possible combination of p and q between 0 and 3.

\[ \text{dec vect[sym] xxx(10)} \]
\[ \text{do i=1,10} \]
\[ \quad \text{compute xxx(i) } = \%zeros(n,n) \]
\[ \text{end do i} \]

creates an array of 10 SYMMETRIC arrays and makes each of them an N×N 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.

```r
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 “handles”.
- 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.

**Variables Defined**

- `%DOFORPASS` pass number through the list (starting at 1) (`INTEGER`).

**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 numerical “handle” and will convert it back to reference a series if necessary. If you need to refer to the series in a `SET` or `FRML` instruction, however, explicitly tell RATS that you intend the index to represent a series by using `index{0}` rather than `index` alone the way you could with an ordinary series name. See the first two examples below.

### Examples

```r
spgraph(hfields=3,vfields=4,$
    xlabels=||"Levels","1st Difference","12th Difference"||,$
    ylabels=||"Interest Rate","Money","Price","Output"||)
dofor y = ffunds lm lp lo
    set dy   = y{0}-y{1}
    set d12y = y{0}-y{12}
    graph(row=%doforpass,col=1)
        # y
    graph(row=%doforpass,col=2)
        # dy
    graph(row=%doforpass,col=3)
        # d12y
end dofor y
spgraph(done)
```

This loops over four series, creating the 1st and 12th difference for each, then graphing the level, and the two derived series in a 4 x 3 graph matrix, organized by `SPGRAPH`. The `GRAPH` instructions use the `%DOFORPASS` variable to put the graph into the proper slot.

```r
dofor i = gnp ml cpi wage trddol importpr
    compute [label] logs = "log"+%l(i)
    set %s(logs) = log(i{0})
    stats %s(logs)
end dofor i
```

This loops over a set of six series, using the `%L` and `%S` functions to create new series `LOGGNP`, `LOGM1`, etc. which are used to hold the created series for the logs.

```r
declare vector values(5)
ewise values(i)=.10^i
dofor [real] scalefac = values
    ...
end dofor
```

loops over values for `SCALEFAC` of .1, .01, .001, .0001 and .00001. Note the use of the `VECTOR` in place of the list of reals.
DSGE — Dynamic Stochastic General Equilibrium Models

DSGE takes a dynamic model (possibly nonlinear) with expectational elements and solves it for a state space form. For a nonlinear model, this is done by linearizing about some expansion point, such as a steady state.

```plaintext
dsge( options ) list of target series=<initial values>
```

Parameters

- **target series** List of series which are the endogenous variables in the model. The number of series should match the number of equations in the model.

- **initial values** (Optional) If desired, you can use the syntax “series=<value>” to provide an initial guess value for the expansion point for any of the series in the list of targets.

Options

- **model=** MODEL to be solved
  This must take the form shown in “Form for Model”.

- **expand=[none]/linear/loglinear**
  This indicates the type of expansion required. The default (EXPAND=NONE) is used when the model is fully linear. EXPAND=LINEAR does a linear expansion and EXPAND=LOGLINEAR does a log-linear expansion.

- **initial=VECTOR of initial guess values [0’s or 1’s]**
- **iters=iteration limit for solution algorithm [50]**
- **cvcrit=convergence criterion [.00001]**
- **steadystate=VECTOR of final converged values [not used]**
- **solveby=[lu]/svd**
- **trace/[notrace]**
  These apply if the model is non-linear and thus needs expansion. Most are used in solving for a steady state. If you want to input the expansion point (for instance, if the model has no steady state), use the INITIAL option or the initial values parameters and ITERS=0. The default starting values are zeros for all variables if EXPAND=LINEAR, or all ones if EXPAND=LOGLINEAR. You can retrieve the final values with the STEADYSTATE option. The VECTORS for both INITIAL and STEADYSTATE are in the order listed in the list of target series.

  The SOLVEBY option controls the method used for solving the Newton’s method steps. SOLVEBY=LU uses the faster LU-decomposition, but that fails when the model has unit roots, in which case you’ll need to switch to the slower SOLVEBY=SVD.
cutoff=value or formula giving stable/unstable roots cutoff [1.0]
See “Algorithm (Solving for State Space Representation)”.

a=A matrix for state space model
Z=Z matrix for state space model
F=F matrix for state space model
These are the output arrays from DSGE which describe the state space model which solves the (possibly expanded) model. This takes the form

\( X_t = AX_{t-1} + Z + FW_t \)

The first components of X are the target series from the DSGE instruction in order (except with the CONTROLS option below). It may have more components to handle additional lags and auxiliary variables for the expectational terms. W has dimension equal to the number of non-identities in the model—if you have more than one shock, they are ordered based upon the order of listing in the GROUP instruction that creates the model.

roots=VECTOR of (absolute values) of the roots of the model
Saves the roots of the model to a vector.

etz=VECTOR[RECTANGULAR] with components
Use this option to have DSGE create a vector of rectangular arrays with components for computing effect of future exogenous shocks. In Sims (2002), the general form of the solution to the (linearized) DSGE is

\[ y(t) = \Theta_1 y(t-1) + \Theta_c z(t) + \Theta_0 \sum_{s=1}^{\infty} \Theta_{s-1}^z E_t z(t+s) \]

You can retrieve \( \Theta_1 \) using the A option, \( \Theta_c \) with the Z option and \( \Theta_0 \) with the F option. The final term drops out if the Z process is serially uncorrelated. If it isn't, or if you want to predict the effect of (known) future shocks to Z, you can use the ETZ option to obtain the three matrices needed for that. After ETZ=THETA, THETA(1) has \( \Theta_y \), THETA(2) has \( \Theta_f \) and THETA(3) has \( \Theta_z \). Note that the infinite sum in the final term will rarely simplify easily, so the sum will generally have to be approximated with a finite number of terms.

**Options for Multiple Stage Solutions**
These options allow you to do the analysis of the model in stages.

analyze=[full]/output/input
form=[sims]/second/first
components=VECT[RECT] of components
controls=# of controls [0]
ANALYZE=FULL gives the standard behavior of the DSGE instruction. The ANALYZE=OUTPUT option generates the components of the model for the form selected by the FORM option, saving the results in the variable provided on the
COMPONENTS option. ANALYZE=INPUT takes as input the components provided via the COMPONENTS option and solves out the model. For FORM=FIRST and FORM=SECOND, the COMPONENTS vector has the five $\Gamma$ matrices from the following representation in order and with sign conventions shown:

$$
\Gamma_0 y_t = \Gamma_i y_{t-1} + \Gamma_f E_t y_{t+1} + \Gamma_c \varepsilon_t + \Gamma_e
$$

FORM=FIRST is a “first-order” representation, where the states are defined so this can be done with $\Gamma_i$ as zero. In FORM=SIMS, the middle term takes the form $\Gamma_f \eta_t$, where $E_t \eta_{t+1} = 0$.

The CONTROLS option controls the positioning of the states for the final set of series listed on the DSGE instruction. Ordinarily, all augmenting states (leads and extra lags) are included after all the original series. However, with the CONTROLS option, those final series are placed at the end of the state vector so you can easily separate them from the remainder of the states.

**Form for Model**

The model is a set of FRML’s which are to take the value zero at a solution. Expectational terms are handled by using leads of the series involved: a lead represents an expectation given information at time $t$. For instance, a standard condition for optimal consumption in a very simple model is

$$
(2) \quad E_t (\beta R_t C_t / C_{t+1}) = 1
$$

This would be represented by

```
frml(identity) f1 = beta*r*c/c{-1}-1.0
```

where $R$ and $C$ are series and BETA is a real-valued parameter.

It’s very important to declare formulas as identities if they are not subject to time $t$ shocks. Each non-identity is assumed to be subject to a separate time $t$ shock which will be one of the components of $W$.

You may need to make adjustments to your original model to put it into this particular form. See Section 10.8.1 of the User’s Guide for some standard adjustments.

**Algorithm (Solving for Steady State)**

In the steady state, all lags or leads (thus expectations) of each dependent variable are assumed to be represented by a single common value. If $x$ represents the vector of steady state values, then the solution is the vector which solves $F(x) = 0$. Newton’s method updates using:

$$
(3) \quad x_{n+1} = x_n - F'(x_n)^{-1} F(x_n)
$$

until convergence.
If the model is being subject to log-linear expansion, the update is

\[ \log x_{n+1} = \log x_n - \left( \text{diag}(x_n)F'(x_n) \right)^{-1} F(x_n) \]

This is repeated until the maximum of the absolute values of the components of the adjustment vectors is less than the convergence criterion.

The default initial guess values are a vector of zeros if a linear expansion is used, and a vector of ones if a log-linear expansion is used. There’s a good chance that you’ll need to provide a better set of values than these. That can be done either with the << fields on the DSGE instruction or with the INITIAL option.

Algorithm (Solving for State Space Representation)

This applies the algorithm described in Sims (2002), based upon the generalized Schur (QZ) decomposition. If the model is non-linear, this uses the linearized or log-linearized version. Derivatives are computed analytically. The CUTOFF option is used to control which roots will be considered stable (and solved backwards) and which will be unstable (and solved forwards). The default for the CUTOFF option is actually 1 + a small value to allow for possible roundoff error in computing roots that should be exactly one.

Examples

The example file CASSKOOPMANS.RPF shows a simple Cass-Koopmans growth model. The model is deterministic and the conditions are linear. What DSGE does here is to suppress the unstable root. The state-space representation is solved using DLM for several different sets of initial conditions, one of which we show here. Because the model is linear, there is no need to find the steady-state in order to expand it; here, we compute it in order to know how to set the initial conditions to be above or below the steady-state.

```r
declare series c lambda k
declare real u0 u1 f0 beta
frml(identity) f1 = u0-u1*c-lambda
frml(identity) f2 = f0*lambda-1.0/beta*lambda{1}
frml(identity) f3 = f0*k{1}-k-c{1}
compute beta=.95,f0=1.3,u0=1.0,u1=0.2
group casskoopmans f1 f2 f3
dsge(expand=linear,steadystate=ss,a=a,z=z,model=casskoopmans) $
c k lambda
dlm(x0=||3.0,17.0,u0-u1*3.0||,a=a,z=z,presample=x1) 1 20 xstates
set c 1 20 = xstates(t)(1)
set k 1 20 = xstates(t)(2)
spgraph(vfields=2,footer="Initial consumption below steady state")
  graph(hlabel="Consumption")
  # c
  graph(hlabel="Capital")
  # k
spgraph(done)
```
These are the first order conditions for a simple RBC model with inelastically supplied labor, log utility function and an AR(1) productivity shock.

\[ 1 = E_t \beta R_{t+1} c_t / c_{t+1} \]
\[ R_t = \alpha k_{t-1}^\alpha \theta_t + (1 - \delta) \]
\[ y_t = k_{t-1}^\alpha \theta_t \]
\[ y_t = k_t - (1 - \delta) k_{t-1} + c_t \]
\[ \log \theta_t = \rho \log \theta_{t-1} + (1 - \rho) \mu + \varepsilon_t \]

Part of the example file SIMPLERBC.RPF is provided below which analyzes this. The model has five equations in five endogenous variables, with one fundamental shock. Because the conditions are non-linear, this needs to be (log) linearized, which is why the DSGE instruction includes the EXPAND=LOGLIN option. We need to save the steady-state solution (using the STEADY option) to transform log-linearized simulations back to levels.

```
dec real beta rho mu alpha delta
dec series r c k y theta

frml(identity) f1 = 1 - (beta*r{-1}*c/c{-1})
frml(identity) f2 = r - (theta*alpha*k{1}^(alpha-1)+1-delta)
frml(identity) f3 = y - (theta*k{1}^alpha)
frml(identity) f4 = y - (c + k - (1-delta)*k{1})
frml           f5 = log(theta) - (rho*log(theta{1})+(1-rho)*mu)

*group simplerbc f1 f2 f3 f4 f5
compute alpha=.3,delta=.15,rho=.9,beta=.95,mu=4.0
dsge(model=simplerbc,expand=loglin,steady=ss,a=adsge,f=fdsge) $
   y  c  k  r  theta
```

Simulate 40 periods of the model with productivity shocks that have a standard deviation of .02 (2%, since theta is in log form).

```
dlm(a=adsge,f=fdsge,presample=ergodic,sw=.02^2,type=simulate) $
  1 40 xsims
set y = exp(xsims(t)(1))*ss(1)
set c = exp(xsims(t)(2))*ss(2)
graph(header="Simulated Economy",key=below) 2
  # y
  # c
```

Variables Defined

- **%CONVERGED**: 1 or 0. Takes the value 1 if the solution for the steady state converged, 0 otherwise.
- **%CVCRT**: Final convergence criterion (if steady state solution is needed).
DUMMY — Generating Dummy and Related Variables

**DUMMY** provides an easy way to generate standard dummies and other constructed variables.

```plaintext
dummy( options )  series  start  end
```

**Parameters**

- **series**
  - series to define
- **start**  **end**
  - range to set. These default to the current sample period.

**Options**

- **from**=
  - starting period for treatment [defaults to start period]
- **to**=
  - ending period for treatment [defaults to end period]

  FROM sets the starting period for the dummy treatment, while TO sets the ending period. Used alone or together, these define a shift dummy which is 0 outside the FROM, TO range, and 1 inside it.

  FROM and TO are also used with the RAMP option and may be used with the LS option.

- **ao**=
  - period for additive outlier [not used]

  AO=$t_0$ defines a dummy which is zero except for 1 at $t_0$.

- **ls**=
  - period for level shift dummy [not used]

  LS=$t_0$ defines a dummy with a level shift at $t_0$. This will be $-1$ on $[start , t_0)$ and 0 on $[t_0 , end]$. This form is chosen so the shift dummy is zero at the end of the sample. If you also use the FROM option, the dummy will be 0 on $[start , FROM)$, $-1$ on $[FROM , t_0)$, and 0 on $[t_0 , end]$.

- **tc**=
  - starting period for temporary change dummy [not used]

- **rate**=
  - rate of exponential decline [.7 per year]

  TC=$t_0$ defines a dummy which is zero for $t < t_0$, 1 for $t = t_0$, and declines exponentially from $t_0$ until end at the rate given by the RATE option.

- **ramp**/[noramp]

  Defines a “ramp”, which is a temporary change in a linear trend (increasing at a rate of 1 per entry). The FROM and TO option define the limits of the trend shift. The dummy is (FROM–TO) from start until FROM, (T–TO) from FROM until TO and 0 after that. Again, this is designed to make the dummy zero at the end of the sample.
Dummy

dfrom=||month,day|| or ||month,day of week,count||
dto=||month,day|| or ||month,day of week,count||
wfrom=number of days before DFROM at which effect starts
to=number of days before DTO at which effect ends

These can be used to define a dummy whose effect is split across two (or more periods) depending upon how many days fall into each period. The dummy sums to one across those periods, but the value for each month is the share of the covered days which lies within that.

DFROM and DTO provide the base date for the effect.

The ||month,day|| form is for a specific day of the month, for instance, ||12,25|| for Christmas. Use ||month,day of week,count|| for “floating” dates, given as the n’th occurrence in a given month of the specified day of the week. Days of the week is coded as 1=Monday,…,7=Sunday. So the second Sunday in May is ||5,7,2|| (5th month is May, 7th day is Sunday, 2 gives the 2nd occurrence in the month).

Use a negative value for count to count back from the end of the month, with –1 giving the last occurrence, –2 the next to last occurrence, etc.

Examples

dummy(ao=1987:10) temp
dummy(ls=1987:10) perm

defines TEMP as a single period dummy for 1987:10, and PERM as a level shift which is –1 through 1987:10 and 0 afterwards.

set trend = t
dummy(ramp,to=1981:1) tbreak
linreg y
# constant trend tbreak

estimates a regression with a broken trend, where the rate (but not the level) changes at 1981:1.

dummy(df=||9,1,1||,w=10,d=||9,1,1||,w=0) statefair

defines a split dummy in which the months of August and September in each year get the fraction of the 11 days up to and including U.S. Labor Day (first Monday in September) that fall in that month.

Checking Your Work

You can use PRINT or GRAPH to display your dummy variable as a way to verify that you’ve created it properly.
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 page UG–247 of the User’s Guide.

```
  ect  list of equations describing the error correction terms
```

Wizard

The Time Series—VAR (Setup/Estimate) wizard includes support for defining and estimating error-correction models.

Parameters

```
equations
```

list 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.RPF, which reproduces Table 5 from King, Plosser, Stock and Watson (1991).

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 *
EIGEN — Eigen Decomposition of Matrices

**EIGEN** computes the eigen decomposition of a **SYMMETRIC** or **RECTANGULAR** matrix, or can solve the generalized eigenvalue problem \( |A - AB| = 0 \). The function %EIGDECOMP can also do eigen decompositions, but without some of the options provided by the **EIGEN** instruction.

```
eigen(options)    matrix   eigenvalues   eigenvectors
```

### Parameters

**matrix**
- Matrix for which the eigen decomposition is computed. 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**.

**eigenvalues**
- (Optional output) For an \( N \times N \) matrix, **EIGEN** saves the (real parts of the) \( N \) eigenvalues in this **VECTOR**. Use the **CVALUES** option if you are expecting complex-valued eigenvalues. Use * for this parameter if you want **eigenvectors** but not **eigenvalues**.

**eigenvectors**
- (Optional output) **EIGEN** saves the (real parts of the) eigenvectors in this \( N \times N \) **RECTANGULAR**. Use the **CVECTORS** option instead if you are expecting complex-valued eigenvectors.

### Sorting and Normalization

For a **SYMMETRIC** matrix, the eigenvalues (and their corresponding eigenvectors) are ordered from largest to smallest; for a **RECTANGULAR** matrix, 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 correspond to the elements in the eigenvalue vector in the sorted order. For a **SYMMETRIC**, **EIGEN** normalizes each eigenvector based upon your choice for the **DMATRIX** option. For a general **RECTANGULAR**, the columns will have unit length if the eigenvalue is real.

### Options

**dmatrix=[eigenvalues]/identity**
- For a **SYMMETRIC** matrix, 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 **matrix** is **SYMMETRIC** and positive definite. For the default **DMATRIX=EIGENVALUES**, each column of \( P \) has unit
length and \( \mathbf{D} \) has the eigenvalues on the diagonal. With \( \text{DMATRIX=IDENTITY} \), \( \text{EIGEN} \) produces as the eigenvectors a matrix \( \mathbf{F} \) such that \( \mathbf{A} = \mathbf{F} \mathbf{F}' \). You can use this on the \text{FACTOR} option for \text{IMPULSE} and \text{ERRORS}. For instance,

\[
\text{eigen(dmatrix=identity) } \%\sigma \ast \text{eigdec}
\]

\[
\text{impulse(model=rmpy,factor=eigdec,steps=24)}
\]

\text{explain=RECTANGULAR of fractions explained}

When \text{matrix} is a cross moment or covariance matrix, the \((i,j)\) element of the matrix set by the \text{EXPLAIN} option is the fraction of the variance in variable \(i\) that is explained by principal component \(j\). Technically, if \text{matrix} is called \( \mathbf{V} \), \text{EXPLAIN} produces

\[
\mathbf{E}(i,j) = \frac{\lambda_j e_{ij}^2}{V_{ii}}
\]

where \( \lambda_j \) is the \(j\)th eigenvalue, \( e_{ij} \) is the \(i\)th element of eigenvector \(j\) and \( V_{ii} \) is the \(i\)th diagonal element of the \text{matrix}. This can only be used if \text{matrix} is \text{SYMMETRIC} and \text{positive definite}.

\text{general=B matrix for generalized eigenvalues [not used]}

For computing generalized eigenvalues of the form \( |\mathbf{A} - \lambda \mathbf{B}| = 0 \), use \text{GENERAL} to supply the \( \mathbf{B} \) matrix, while the \text{matrix} parameter supplies the \( \mathbf{A} \) matrix. \( \mathbf{A} \) and \( \mathbf{B} \) must be symmetric. See the \text{QZ} instruction for computing generalized decompositions with non-symmetric matrices.

\text{cvalues=VECTOR[COMPLEX] of complex eigenvalues}
\text{cvectors=RECTANGULAR[COMPLEX] of complex eigenvectors}
\text{sort=real/imag/[absval]}

Use \text{CVALUES} if you want the complex eigenvalues of a \text{RECTANGULAR} matrix, rather than just the real part available with the \text{eigenvalues} parameter. Use \text{CVECTORS} if you want the \text{complex} eigenvectors. They are sorted according to the \text{SORT} option. Sorting is always from highest to lowest. \text{SORT=REAL} sorts on the real part, \text{SORT=IMAG} on the imaginary part and \text{SORT=ABSVAL} on the absolute value.

\text{Examples}

\text{eigen } \%\text{cmom eigenval}
\text{compute condition=}%\text{maxvalue(eigenval)/%minvalue(eigenval)}

computes the condition number for the (positive-definite) \text{SYMMETRIC} matrix \%\text{CMOM}.

\text{eigen(cvalues=cxvalues,sort=absval) a}
\text{dec vector absval}\left(\%\text{rows(a)}\right)
\text{ewise absval(i) = %real(%cabs(cxvalues(i)))}

computes the absolute values of the eigenvalues of \text{A}, producing the \text{VECTOR} \text{ABSVAL}.
**ENCODER — Restricted Regressions**

`ENCODER`, combined with the `UNRAVEL` option on estimation instructions such as `LINREG`, provides one of two methods available in RATS for computing restricted regressions; see also *User’s Guide*, Section 2.10 and `RESTRICT` and `MRESTRICT` here in the *Reference Manual*.

```
encode ( options )  codings  start  end  list of new series
# list of original variables in regression format
```

**Parameters**

- **codings**
  A RECTANGULAR matrix of dimension number of new series × number of original variables. Each row of `codings` provides the coefficients of a different linear combination of the variables.

- **start end**
  This is the range of entries over which `ENCODER` will compute the transformation. If you have not set a `SMPL`, this defaults to the maximum range allowed by the variables being transformed.

- **new series**
  List of series that `ENCODER` will construct from the linear combinations of regressors. There should be one series per row of `codings`. We recommend using the newer `RESULTS` option instead.

**Options**

- `[clear]/noclear`
  Unless you use `NOCLEAR`, `ENCODER` erases any information from previous `ENCODER` instructions. *If a regression requires more than one ENCODER, the ENCODERs after the first must use NOCLEAR.*

- `results=VECTOR[SERIES] of constructed variables`
  The ENCODERd variables have no particular use other than as a group in a regression. `RESULTS` stores the constructed variables into a `VECTOR` of `SERIES`. To do the restricted regression, just use the name of the `RESULTS` array in your regressions (for example, on the supplementary card of a `LINREG`).

**Description**

`ENCODER` 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. **ENCOD**E and **UNRAVEL** don’t change this procedure, they just ensure that the regression is reported using the original regressors. Basically, **ENCOD**E 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**

The code below is taken from the example **PDL.RPF**. It computes a 3rd order polynomial distributed lag first using **ENCOD**E and **UNRAVEL**, then with the **@PDL** procedure.

```
declare rect r
* PDL with no end constraints
dim r(4,25)
ewise r(i,j)=j^(i-1)
encode(results=enc) r
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc
* Same estimation using the PDL procedure
@pdl(graph) longrate
# shortrate 0 24 3
* PDL with far constraint
dim r(3,25)
ewise r(i,j)=(j-26)*j^(i-1)
encode(results=enc) r
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc
* Same estimation using the PDL procedure
@pdl(constrain=far,graph) longrate
# shortrate 0 24 3
```

**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
# shortrate{0 to 12}
```

will actually have just 4 numerator degrees of freedom, not 13. **RATS** will automatically adjust the degrees of freedom accordingly, and issue a warning like:

**X13. Redundant Restrictions. Using 4 Degrees, not 13**

**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 primarily 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.

<table>
<thead>
<tr>
<th>end (option)</th>
</tr>
</thead>
</table>

**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 Memory* 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.

In recent versions of RATS, the new regression list functions, such as %RLADDLAG and %RLCONCAT take up part of the role that ENTER used to play.

```
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=(output) 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 in an INTEGER the number of entries processed.

sequence=sequence number in LIST sequence

This allows you to use a list defined by LIST and CARDS instructions rather than a supplementary card to provide the input. If you use the SEQUENCE option, RATS evaluates the subsequent CARDS instruction using the value supplied by the option as the sequence number in the sequence defined by the LIST instruction.

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 or PACKED, by rows, lower triangle only.

Examples

```plaintext
procedure constrain  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
```

CONSTRAIN will have NCONSTR supplementary cards, each with two integer values and one real. The first integer should be a series name or number. For example:

```plaintext
@constrain 3 r
# tbill  2019:1  2.2
# tbill  2019:2  2.3
# tbill  2019:3  2.4
```

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:

```plaintext
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 page UG–482 in the User’s Guide for more on the regressor list functions.

Notes

INPUT, READ, COMPUTE and FIXED 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.

<table>
<thead>
<tr>
<th>Environment</th>
<th>environment strings (separated with blanks)</th>
</tr>
</thead>
</table>

**Environment Strings**

The following are the available environment strings. *Note that blanks are significant.*

- `[showgraphs]/[noshowgraphs]`                  
  - `showgraphs`/`noshowgraphs` suppresses the displaying of graphs on the screen (they will be still be saved to disk if you use `OPEN PLOT` or `ENV GSAVE` instructions). `PRINTGRAPHS` causes RATS to automatically print each graph as it is generated—graphs are printed even if you use `NOSHOW` as well.

- `printgraphs/[noprintgraphs]`                
  - `printgraphs` causes RATS to automatically print each graph as it is generated—graphs are printed even if you use `NOSHOW` as well.

- `procedure=procedure library file`           
  - If you have a suite of procedures you use regularly, `ENV PROCEDURE` provides 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 process the library file anytime you do something that clears the memory (File—Clear Memory 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 procedures. Another way to do this are to use `File–Preferences` to have RATS load a procedure library each time the program starts. Or, if running in batch mode, you can use the `/procedure=libraryfile` switch on the command line.

- `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`. 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. (An alternative to this is to use `File–Open` and select the file).

- `data=data file name`                       
- `copy=copy file name`                       
- `plot=plot file name`                       
  - These are equivalent to `OPEN DATA`, `OPEN COPY` and `OPEN PLOT` instructions.
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.

Note: The ENV GSAVE/GFORMAT operation described below has been superseded by the GSAVE instruction.

gsave=filename template for saving graphs
gformat=rgf/portrait/landscape/wmf/pict

Use GSAVE=template to automatically save graphs to disk. If include an asterisk (*) somewhere in the template, RATS will save each subsequent graph using that template, replacing the * with a sequence number. For example, to save graphs as MYGRAF1.RGF, MYGRAF2.RGF, etc., do:

```
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).

**Examples**

```
environment gsave="basics*.eps" gformat=portrait
graph(key=upleft) 3
# rate
# ip
```

This saves a graph to disk in PostScript 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

**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).

```
 equation(options) equation depvar ARlags MAlags
 # list of explanatory variables in regression format (omit if using EMPTY or LASTREG)
```

**Wizard**

You can use the *Statistics—Equation/FRML Definition* wizard 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 *AR* option (or *ARlags*) and (MA option) (or *MAlags*) 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.

  The AR and MA options are the preferred way to input ARMA lag information.

- **MAlags**
  (Optional) Moving Average coefficients. These work the same way as the *ARlags* parameter.

**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 besides 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]

LASTREG defines the equation using the variables and estimated coefficients from the most recent regression. Omit the supplementary card if you use this option.

ar=list of autoregressive lags [0]
ma=list of moving average lags [0]

You can use these as a (more readable) 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.

constant/noconstant

For an ARMA equation, EQUATION 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 CONSTANT on the supplementary card along with the other explanatory variables.

regrassors/[noregressors]

You can use this option 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 versions of RATS before 7, and that’s still acceptable.)

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

\textbf{variance= residual variance [not used]}
Residual variance for this equation. You only need to supply this if you are going to use \texttt{SIMULATE, IMPULSE} or \texttt{ERRORS}. It is usually set when the equation is estimated.

\textbf{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.

\textbf{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 1.4 in the \textit{Additional Topics} PDF for details.

\textbf{Example Using Standard Equations}
This sets up two of the equations for Klein’s model I. The second one is the identity
\[ Y = C + I + G. \]

\begin{verbatim}
equation wageeq privwage
# constant prdct{0 1} trend
equation(identity, coeffs=||1.0,1.0,1.0||) prdcteq prdct
# consmptn invest govtexp
\end{verbatim}

\textbf{ARMA Equations}
The RATS forecasting instructions cannot handle directly equations with the multiplicative 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}.

\textbf{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} uses separate “mean” and “noise” models, rather than combining them into a reduced form.
You can estimate simple ARMA equations (with no ARMAX components) by \texttt{INITIAL} followed by \texttt{ITERATE}. \texttt{INITIAL} does not compute coefficients for non-ARMA variables, so you need to do one of the following for ARMAX equations:

- Use \texttt{INITIAL} to set initial guesses for the ARMA parameters, while zeroing out the extra coefficients. Use \texttt{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 \texttt{EQUATION} and \texttt{LINREG(EQUATION=...)} or just \texttt{LINREG(DEFINE=...)} to create the base equation. Then use:

\begin{verbatim}
MODIFY equation
VADD %MVGAVGE list of lags
\end{verbatim}

to add the MA components. This works better when the regression part of the model dominates.

\section*{Examples With ARMA Equations}

An ARIMA(2,1,1) model: \( (1 - \Phi_1 L - \Phi_2 L^2)(1 - L) y_i = (1 + \Theta_1 L) u_i \)

\begin{verbatim}
difference y / ydiff
equation(ar=2,ma=1) ydiffeq ydiff
\end{verbatim}

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 \texttt{MODIFY} and \texttt{VREPLACE}:

\begin{verbatim}
modify ydiffeq yeq
vreplace(print) ydiff with y diff 1
\end{verbatim}

The next example defines an ARIMA(2,0,1) with AR lags 1 and 4 and MA lag 4:

\( (1 - \Phi_1 L - \Phi_2 L^4) y_i = \alpha + (1 + \Theta_1 L^4) u_i \)

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 \texttt{CONSTANT} first, then the AR lags, then the MA lags.

\begin{verbatim}
equation(coeffs=||0.0,0.5,0.4,0.7||,ar=||1,4||,ma=||4||) yeq y
\end{verbatim}

\section*{See Also...}

\begin{itemize}
  \item \texttt{ASSOCIATE} \hspace{2cm} Sets coefficients for an \texttt{EQUATION}
  \item \texttt{FRML} \hspace{2cm} Defines a \texttt{FRML}—a (possibly) non-linear relationship
  \item \texttt{BOXJENK} \hspace{2cm} Estimates ARIMA, transfer function, and intervention models.
  \item \texttt{MODIFY} \hspace{2cm} With \texttt{VREPLACE} and \texttt{VADD}, changes the structure of an equation.
\end{itemize}
EQV

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.

```
eqv  list of series (usually listed by number)
      < text card >  names for series
```

Parameters

list of series  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 list of series. The names on this card must be legal variable names:

- The name must begin with a letter, % or _.
- The only characters are letters, digits, $, % and _.
- The maximum length is sixteen characters.

Description

EQV used to be an important instruction, and you may see it in programs written for RATS version 3 or earlier. Now, if you need an instruction like this, LABELS will probably be the better choice. While EQV assigns series names, which can be used on input and output, LABELS only sets output labels, which offers several advantages:

- Series names (done with EQV) must be unique: you can’t have two series called RESIDS or FORECAST, but any number of series can share a 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 LABEL variables.

Example

```
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
```

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.

```plaintext
errors(options) equations
# equation stderrors newstart column (one card per equation)
```

Wizard

The *Time Series—VAR (Forecast/Analyze)* wizard provides an easy, dialog-driven interface for computing variance decompositions.

Parameters

- **equations**: Number of equations in the system.

Supplementary Cards (if MODEL option isn’t used)

- **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.

Options

With the exception of **MODEL**, **PRINT/NOPRINT**, **STEPS**, and **STDERRORS**, these all apply to **ERRORS only** when used for decomposition of variance.

- **model=model name**: Of the two ways to input the form of the model to be solved (the other is with supplementary cards), this is the more convenient. **MODELS** are usually created by **GROUP** or **SYSTEM**. **FRML’s** (formulas) can be included if they are of a simple linear form. (**ERRORS** requires that the model be fully linear.) If the model includes any identities, those should be last in the model. If you use this, omit the "equa-
tion” supplementary cards.

**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 use the **STEPS** option to supply a value.

**cv=** SYMMETRIC covariance matrix of residuals [%sigma]

**factor=** RECTANGULAR decomposition matrix

Use **CV** if you want the decomposition computed using a Cholesky factorization of the covariance matrix (*User’s Guide*, Section 7.5.1). 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 7.5.2). The form of the factorization does not affect the forecast variance, only the decomposition. (This option was called **DECOMP** in versions before 7. **DECOMP** is still recognized as a synonym for **FACTOR**.)

**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. See “Output” for a sample.

**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 labels for shocks [variable names]

Use the **LABELS** option if you want to supply your own labels for the shocks. Otherwise, **ERRORS** will use the names of the variables.

**stderrs=** VECTOR[SERIES] for standard errors

This option saves the standard errors of forecast into a VECTOR of SERIES.

**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 output.
**Example of Decomposition**

This example computes decompositions for a system of four equations using two orderings.

```r
system(model=canmodel)
variables cpr m1 ppi gdp
lags 1 to 4
det constant
det end(system)
estimate(cvout=v)
*   * The first decomposition is done in the original order:
*   * CPR-M1-PI-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)
```

**Example of Computing Forecast Standard Errors**

```r
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). Note that the combination of **FORECAST** and **ERRORS** can be done with just one **UFORECAST** instruction.
Errors

Technical Information

For a static model such as

\[(1) \quad y_t = X_t \beta + u_t; \quad \text{Var}(u_t) = \sigma^2\]

the variance of the error in using \(X_{t+1} \hat{\beta}\) to forecast \(y_{t+1}\) is

\[(2) \quad \sigma^2 + \sigma^2 X_{t+1} (X'X)^{-1} X'_{t+1}\]

(See, for instance, Greene(2012), p 81). 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:

\[(3) \quad y_t = \alpha y_{t-1} + u_t; \quad \text{Var}(u_t) = \sigma^2\]

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

\[(5) \quad \alpha u_t + u_{t+1} + (\alpha^2 - \hat{\alpha}^2) y_{t-1}\]

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,

\[(6) \quad y_t = \sum_{s=0}^{\infty} \Psi_s u_{t-s} = \sum_{s=0}^{\infty} \Psi_s Fv_{t-s}\]

the error in the \(K\)-step ahead forecast is

\[(7) \quad \sum_{s=0}^{K-1} \Psi_s u_{t-s} = \sum_{s=0}^{K-1} \Psi_s Fv_{t-s}\]

where \(FF'\) 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
\[(8) \sum_{s=0}^{K-1} \Psi_s FF' \Psi'_s = \sum_{s=0}^{K-1} \Psi_s \Sigma \Psi'_s \]

This does not depend upon which factorization of \( \Sigma \) is chosen. However, the decomposition of variance, which breaks this sum down into the contributions of the component of \( \mathbf{v} \), does. See User’s Guide Section 7.6 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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Std Error</th>
<th>USARGDPS</th>
<th>CANUSXSR</th>
<th>CANCD90D</th>
<th>CANM1S</th>
<th>CANRGDPS</th>
<th>CANCPINF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.004688318</td>
<td>13.062</td>
<td>2.172</td>
<td>2.321</td>
<td>0.604</td>
<td>81.842</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>0.007407510</td>
<td>21.727</td>
<td>2.495</td>
<td>1.291</td>
<td>0.943</td>
<td>73.505</td>
<td>0.040</td>
</tr>
<tr>
<td>3</td>
<td>0.008882190</td>
<td>19.386</td>
<td>4.086</td>
<td>0.977</td>
<td>8.445</td>
<td>67.062</td>
<td>0.044</td>
</tr>
<tr>
<td>4</td>
<td>0.010004321</td>
<td>15.284</td>
<td>4.194</td>
<td>3.754</td>
<td>12.355</td>
<td>64.256</td>
<td>0.156</td>
</tr>
<tr>
<td>5</td>
<td>0.010632775</td>
<td>14.403</td>
<td>4.704</td>
<td>6.786</td>
<td>12.090</td>
<td>61.583</td>
<td>0.434</td>
</tr>
<tr>
<td>6</td>
<td>0.011511805</td>
<td>17.409</td>
<td>5.516</td>
<td>13.026</td>
<td>10.328</td>
<td>53.250</td>
<td>0.472</td>
</tr>
<tr>
<td>7</td>
<td>0.013088464</td>
<td>21.646</td>
<td>5.594</td>
<td>21.954</td>
<td>8.939</td>
<td>41.435</td>
<td>0.432</td>
</tr>
<tr>
<td>8</td>
<td>0.014910398</td>
<td>23.338</td>
<td>5.798</td>
<td>29.436</td>
<td>8.936</td>
<td>31.964</td>
<td>0.529</td>
</tr>
<tr>
<td>9</td>
<td>0.016733054</td>
<td>23.104</td>
<td>6.219</td>
<td>35.950</td>
<td>8.549</td>
<td>25.436</td>
<td>0.742</td>
</tr>
<tr>
<td>10</td>
<td>0.018526192</td>
<td>22.059</td>
<td>6.274</td>
<td>41.696</td>
<td>8.106</td>
<td>20.916</td>
<td>0.949</td>
</tr>
</tbody>
</table>

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 16.5) 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 6.3 in the User’s Guide for more on the use of exponential smoothing for forecasting.

```
esmooth( options ) series start end
```

Wizard

The Time Series—Exponential Smoothing wizard provides dialog-driven access to most of the features of the ESMOOTH instruction.

Parameters

- **series**: Series to smooth, seasonally adjust or forecast.
- **start**  **end**: Range to smooth. If you have not set a SMPL, this defaults to the defined range of series.

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.
**ESmooth**

`estimate/[noestimate]`  
`alpha=constant level smoothing parameter [ .3 ]`  
`gamma=trend smoothing parameter [ .3 ]`  
`delta=seasonal smoothing parameter [ .3 ]`  
`constrain/[noconstrain]`  
`initial=[full]/start`  

If you use the `ESTIMATE` option, **ESMOOTH** finds the values for $\alpha$, $\gamma$ and $\delta$ (see "Technical Information") 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 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 `CONSTRAIN` to constrain the estimated parameter values to the range [0,1]. With `INITIAL=START`, 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.

`[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.

**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 "Technical Information" 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**

```r
esmooth(estimate, fore=forecast, steps=12) tbill3mo 1980:1 2009:12
```

This uses data from 1980:1 through 2009:12 to fit a non-trending, non-seasonal model with estimated coefficients, and then uses that model to produce forecasts for 2010:1 through 2010:12.

```r
esmooth(alpha=1.0, gammaw=0.3, trend=exponential, forecasts=ship_f, steps=21) shipment 1994:1 2010:3
```

forecasts 2010:4 to 2011:12 using an exponential trend, non-seasonal model, with assigned parameters.

```r
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 6.3 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>Model Selection</th>
<th>TREND</th>
<th>SEASONAL</th>
<th>SumSquares</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
<td>1919642978.894695</td>
<td>8531.21</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>Additive</td>
<td>387239181.209608</td>
<td>7629.87</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>Multiplicative</td>
<td>258039909.180180</td>
<td>7399.71</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>None</td>
<td>1661329541.805498</td>
<td>8455.61</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>Additive</td>
<td>139617130.682347</td>
<td>7057.79</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>Multiplicative</td>
<td>76249709.192393</td>
<td>6714.82</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>None</td>
<td>1518459266.953026</td>
<td>8404.62</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>Additive</td>
<td>120020798.921642</td>
<td>6972.04</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>Multiplicative</td>
<td>76876443.721707</td>
<td>6719.46</td>
<td></td>
</tr>
</tbody>
</table>

Model with TREND=Linear , SEASONAL=Multiplicative

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha (level)</td>
<td>0.216416</td>
</tr>
<tr>
<td>Gamma (trend)</td>
<td>0.076146</td>
</tr>
<tr>
<td>Delta (seasonal)</td>
<td>0.352740</td>
</tr>
</tbody>
</table>
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></th>
<th>No Seasonal</th>
<th>Additive Seasonal</th>
<th>Multiplicative Seasonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Trend</td>
<td>$S_t = S_{t-1} + \alpha e_t$</td>
<td>$S_t = S_{t-1} + \alpha e_t$</td>
<td>$S_t = S_{t-1} + \alpha e_t / I_{t-p}$</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>
<td>$I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$</td>
</tr>
<tr>
<td>Linear Trend</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>
<td>$S_t = S_{t-1} + T_{t-1} + \alpha e_t / I_{t-p}$</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>
<td>$T_t = T_{t-1} + \alpha \gamma e_t / I_{t-p}$</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>
<td>$I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$</td>
</tr>
<tr>
<td>Exponential Trend</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>
<td>$S_t = S_{t-1}T_{t-1} + \alpha e_t / I_{t-p}$</td>
</tr>
<tr>
<td></td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t / S_{t-1}$</td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t / (I_{t-p}S_{t-1})$</td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t / (I_{t-p}S_{t-1})$</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>
<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 ESMOOTH does not constrain the values to the \([0,1]\) range by default. If you do want to impose the \([0,1]\) constraint, use the \texttt{CONSTRAIN} option.

**Variables Defined**

- \%NOBS \hspace{1cm} Number of observations
- \%RSS \hspace{1cm} Sum of squared errors
- \%ESALPHA \hspace{1cm} \(\alpha\), the level smoothing parameter
- \%ESGAMMA \hspace{1cm} \(\gamma\), the trend smoothing parameter
- \%ESDELTA \hspace{1cm} \(\delta\), the seasonal smoothing parameter

**See Also . . .**

- \textit{UG}, Chapter 6 \hspace{1cm} 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.

```plaintext
estimate(options) start end
```

**Wizard**

The **Time Series—VAR (Setup/Estimate)** wizard provides an easy, dialog-driven interface for defining and estimating VAR models.

**Parameters**

```plaintext
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.

**Options**

- **dfc=** Degrees of freedom correction (*Additional Topics*, Section 1.4)
- **smpl=** SMPL series or formula ("SMPL Option" on page RM–546)
- **spread=** Standard SPREAD option ("SPREAD Option" on page RM–547)
- **weight=** Series of entry weights ("WEIGHT option" on page RM–549)

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 2.1 of the *User’s Guide*. In some older versions, **CVOUT** was called **OUTSIGMA**. **ESTIMATE** will still recognize the old name.
Estimate

**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=SYMmetric X’X 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 7.4). 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 `CVOUT` options and saved as the `%SIGMA` matrix (used for constructing impulse responses, variance decompositions, etc.) uses a divisor of $T$:

$$
\Sigma = \frac{1}{T} \sum_{t=1}^{T} u'_t u_t
$$

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

- `%BETASYS` stacked coefficient `VECTOR`
- `%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: $I - \sum_{s=1}^{p} \Phi_s$ where $\Phi_s$ is the matrix of VAR coefficients for lag $s$. In an ECT VAR, these are computed with the coefficients on the differenced dependent variables.
- `%VECMALPHA` When using `ECT`, this is set to the $\alpha$ matrix (the loadings) for the error correction model ($N \times r$ `RECTANGULAR`).
- `%VECMPI` When using `ECT`, this is set to the $P$ matrix for the error correction model ($N \times N$ `RECTANGULAR`).
- `%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. Note that in a VAR with more than two variables, an insignificant result should not (by itself) be interpreted as a lack of causality from the tested variable to the dependent variable—that requires a more formal block exogeneity test.

Examples

```
system(model=canmodel)
variables usargdps canusxsr cancd90d canm1s canrgdps cancpinf
lags 1 to 4
det constant
end(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.

```
equation checkeq gdp
  # constant  gdp{1 to 4}
system checkeq
end(system)
estimate(cohistory=coh,noprint) 5 9
do time=10,1999:3
  kalman(rtype=recursive,cohistory=coh) resids
end do time
```

The ESTIMATE computes the regression over the first five usable time periods (allowing 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.
linreg(define=eweq) w
# constant y z
*

system(model=vecm)
variables y z w
lags 1 2
det constant
ect eweq
end(system)
*

estimate

This estimates a model with cointegration (VECM), where the error correction equation is generated from an "Engle-Granger" regression.

Output for a VECM

This is the output for the first equation in the VECM model demonstrated above. Several things to note:

1. The 2 lags in a VAR setup is the number of lags in the expanded VAR, which reduces to 1 lag as the model is actually estimated.
2. The lagged differenced dependent variables are shown with D_ prefixes on the name of the original variable.
3. The lagged error correction term(s) are shown as ECn, where n is the number. If there is just one cointegrating vector, that will be EC1; if there is more than one, they will be EC2, EC3, etc.

<table>
<thead>
<tr>
<th>Dependent Variable: Y</th>
<th>Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean of Dependent Variable</td>
<td>0.0103956122</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std Error of Dependent Variable</td>
<td>0.3270248297</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard Error of Estimate</td>
<td>0.3151203397</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum of Squared Residuals</td>
<td>9.2349770506</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Durbin-Watson Statistic</td>
<td>1.9036</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Variable** | **Coeff** | **Std Error** | **T-Stat** | **Signif** |
--- | --- | --- | --- | --- |
1. D_Y{1} | 0.178276649 | 0.164688865 | 1.08251 | 0.28182662 |
2. D_Z{1} | 0.312995255 | 0.161102252 | 1.94284 | 0.05506060 |
3. D_W{1} | -0.367720385 | 0.161967563 | -2.27033 | 0.02549618 |
4. Constant | 0.006065021 | 0.032078520 | 0.18907 | 0.85045154 |
5. EC1{1} | -0.417973964 | 0.150046937 | -2.78562 | 0.00647464 |

See Also . . .

**UG, Chapter 7** | 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. It sets element \((I, J)\) or element \((I)\) of an array, for all \(I\) and \(J\), according to the specified formula. Use COMPUTE for more standard matrix operations (such as inversion and multiplication).

```
ewise array(I, J) = function of (I, J) \quad \text{for two-dimensional arrays}
ewise array(I) = function of (I) \quad \text{for one-dimensional arrays}
```

**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. 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.

You can apply EWISE to arrays of any data type (including arrays of arrays).

You can use any INTEGER variable in place of \(I\) and \(J\). However, \(I\) and \(J\) are already defined as INTEGER’s (primarily for use in EWISE); any other variable would need to be DECLARE’d.

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.

```
dec vect[symm] vs(10) vsinv(10)

...Instructions setting VS(1),...,VS(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

```
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

```
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 a `PARMSET` 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 ALL option)
```

**Wizard**

In the *Statistics—Regression Tests* wizard, select the *Exclusion Restrictions* radio button.

**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]/noprprint`
  
  `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 was called `WHOLE` in version 6 or earlier.)

- `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 3.3 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 (*Additional Topics*, Section 1.4).
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
These are done as Wald tests, with formulas described in Section 3.3 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 DDV or LDV 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

<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>%NDFTEST</td>
<td>(numerator) degrees of freedom for the test (INTEGER)</td>
</tr>
</tbody>
</table>

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 1.4.5 of the Introduction and Section 15.2 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 procedure called “PROCNAME”, RATS will search for a file called name **PROCNAME.SRC**. It will check (in order) the "parent" directory if one procedure calls another, your “Procedure Directories” (defined in 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 %size(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 15 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.
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.

```
proc quickie matpar
  type symmetric *matpar
  if .not.%defined(matpar) {
    disp "Syntax: @quickie matrix"
    return
  }
  compute matpar=inv(matpar)
end
```

Example

```
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 $
  2019:1 2019:12 flong
```

LONGRATE is the SERIES parameter, 2019:1 and 2019:12 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 15 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, SQRT, MOVE — Convenience Transformations
These are convenience instructions as you can do (effectively) the same thing using
SET. The LOG instruction is similar, but is used often enough that we listed it separate-
lly. EXP takes the inverse natural log a series (that is, returns $e^x$), SQRT the square
root, and MOVE simply copies information around. Of the three, MOVE is the one most
likely to be helpful if you need to shift information to a different set of entries, as the
SET instruction for doing that can be a bit hard to read if it isn’t just a lag shift.

<table>
<thead>
<tr>
<th>exp</th>
<th>series</th>
<th>start</th>
<th>end</th>
<th>newseries</th>
<th>newstart</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt</td>
<td>series</td>
<td>start</td>
<td>end</td>
<td>newseries</td>
<td>newstart</td>
</tr>
<tr>
<td>move</td>
<td>series</td>
<td>start</td>
<td>end</td>
<td>newseries</td>
<td>newstart</td>
</tr>
</tbody>
</table>

Parameters

**series**
Series to transform.

**start**  **end**
Range to transform. By default, 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

```plaintext
exp x / ex
set ex = exp(x)
```
are equivalent instructions for making EX equal to the exponential of X.

```plaintext
move beer 1991:1 1991:12 b1991 1
move beer 1993:1 1993:12 b1993 1
move beer 1994:1 1994:12 b1994 1
graph(footer="Figure 2-2 Seasonal Plot of Beer Production",$
   xlabels=||"Jan","Feb","Mar","Apr","May","Jun","$
        "Jul","Aug","Sep","Oct","Nov","Dec"||) 4
```

# b1991
# b1992
# b1993
# b1994

This creates a graph of data from four years but with the four years stacked on top of
each other across the months (so the time axis is really just the twelve months of the
year). To do that, it’s necessary to rearrange the data so that the 12 entries for each
year go into entries 1 to 12. The SET instruction equivalent to the second MOVE would
be something like

```plaintext
set b1992 1 12 = beer(1992:1-1+t)
```
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.

```
extremum( options ) series start end
```

Wizard

From the Statistics—Univariate Statistics wizard, choose "Extreme Values".

Parameters

series Series you are analyzing.
start end Range to use. If you have not set a SMPL, this defaults to the defined range of series.

Options

[print]/noprint
title="title for output" ["Extreme Values for Series xxx"]
You can use NOPRINT to suppress the output.

smpl=SMPL series or formula ("SMPL Option" on page RM–546)
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.

shuffle=SERIES[INTEGER] for remapping ("SHUFFLE Option" on page RM–548)

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

```
ext gdpgrowth
```

Extremum 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>fft</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td>ift</td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
</tbody>
</table>

**Parameters**

- **cseries**: The complex series to transform.
- **start, 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

$$\tilde{X}(2\pi j/T) = \sum_{t=1}^{T} X(t) \exp\left(-2\pi ij(t-1)/T\right); j = 0, 1, \ldots, T-1$$

The frequencies computed run from 0 to $2\pi(T-1)/T$ by increments of $2\pi/T$. Frequency 0 goes into entry newstart.

**Usage**

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 %FREQSIZE recommendation is doubled.

```
compute nords = %freqsize(2017:12)*2
freq 1 nords
rtoc
# employ
# 1
fft 1
cmult(scale=1.0/2017:12) 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.

DIFFERENCE does a specific set of differencing and fractional differencing filters. In addition the techniques provided by FILTER and DIFFERENCE, there are many rats procedures available that implement other filtering techniques.

```
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

You can use the Data/Graphics—Filter/Smooth wizard to do various filtering operations. Just which technique to apply using the "Filter Type" drop down.

Parameters

- **series**: Series to transform
- **start** and **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.
Options

- **type**=[general]/flat/henderson/spencer/hp/hp-detrend/lagging/centered
- **width**=base window size (for TYPE=FLAT,HENDERSON,SPENCER,LAGGING)
- **span**=same as WIDTH (used in older versions of RATS)
- **by**=repetitions of simpler filter, used with TYPE=FLAT
- **tuning**=tuning value for TYPE=HP or TYPE=HPDETREND [depends on CALENDAR]

**TYPE** indicates the type of filter to be used. See the "Technical Information" for details on each of these.

**TYPE=GENERAL** (the default) allows any pattern—if used, it requires use of either the supplementary cards or the EQUATION option to indicate the form of the filter. With **TYPE=GENERAL**, the zero lag is assumed to have a coefficient of 1.0 unless included explicitly.

**TYPE=FLAT** is a centered flat moving average, with width given by **WIDTH**. It can also be used for an \( N \times M \) convolution of flat filters by combining the **WIDTH** and **BY** options.

**TYPE=HENDERSON** gives a centered Henderson filter with width given by the **WIDTH** 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=HPDETREND** produces the residuals from the Hodrick-Prescott filter (generally interpreted as the "cycle"). The **TUNING** option applies as it does for **TYPE=HP**.

**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 "Using the EQUATION Option". Omit the supplementary cards when you use it.
**Filter**

**remove=mean/trend/seasonal/both**
Removes from series by linear regression the mean, mean and trend, seasonal
(using dummies) and mean, or trend and seasonsals, 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 FILTER to normalize the weights to sum to 1—this allows you to just supply a shape using WEIGHTS and let FILTER figure out the normalization. For example: TYPE=CENTERED,

**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, et al 1998).

**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.
**Filter**

**Technical Information**
For \texttt{TYPE=FLAT}, if the window size is odd (\texttt{WIDTH}=2N+1), the filtered series is
\[
\hat{x}_t = \frac{1}{(2N+1)}(x_{t-N} + x_{t-N+1} + \ldots + x_{t+N-1} + x_{t+N})
\]
If it’s even (\texttt{WIDTH}=2N), the output is
\[
\hat{x}_t = \frac{1}{2N}(0.5x_{t-N} + x_{t-N+1} + \ldots + x_{t+N-1} + 0.5x_{t+N})
\]
This is generally used with seasonal data where the \texttt{WIDTH} is the length of the seasonal, and is also known as a $2 \times N$ filter.

If \texttt{TYPE=FLAT} and you use both the \texttt{WIDTH} and \texttt{BY} options, the result is the same effect as a flat filter of width given by the \texttt{WIDTH} applied to the output of a flat filter of width given by the \texttt{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 and Prescott, 1997) 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 \texttt{TUNING} option. This is solved internally by Kalman smoothing. See the \texttt{HPFILTER.RPF} example for additional technical details.

**Missing Values**
If any of the values required to compute an entry are missing, \texttt{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 \texttt{TRUNCATE} or \texttt{EXTEND} options.

**Examples**
\begin{verbatim}
data 1955:1 2010:6 ip
filter(type=flat,width=12) ip 1955:7 2009:12 trdcycle
\end{verbatim}
This sets \texttt{TRDCYCLE} \(t = (0.5 \times IP_{t-6} + IP_{t-5} + \ldots IP_{t+5} + 0.5 \times IP_{t+6})/12\). Note that the range is set to run from (1955:1)+6 to (2010:6)–6. This use of an even window size means that each calendar month will get the same weight in the average, and thus a seasonal will tend to be flattened.
filter(remove=hptrend,tuning=1600) y / detrend

makes DETREND equal to the difference between Y and its Hodrick-Prescott trend estimate.

filter(type=centered,width=5,extend=repeat) strikes / sstrikes
graph(footer="Simple 5-term Moving Average of Strike Data",$
   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

does a 23-term Henderson with minimum MSE 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 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 fractional difference filter (Section 14.10 of the User’s Guide. Note that 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 / fracdiffs
# lags
# coeffs
Filter

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
```
**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**).

```plaintext
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** (Section 4.6, User’s Guide).

- **method=[simplex]/genetic/bfgs/annealing/ga/grid**
  These control the optimization method. Of these, only BFGS makes assumptions about differentiability (it requires the function be twice continuously differentiable) and is the only one capable of producing standard errors for the estimates. **SIMPLEX** requires continuity, while the others have no special requirements. **GRID** is specific to **FIND** and does a grid search over a range of values provided by the **GRID** option.

- **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. The "iteration" counts for simplex, genetic, annealing and GA are designed to give a similar amount of calculation to what an iteration in BFGS requires. 
pmethod=simplex/genetic/bfgs/grid
piters=number of PMETHOD iterations to perform [none]

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.

grid=VECTOR[VECTOR] with values for grid search (METHOD=GRID only)

With METHOD=GRID or PMETHOD=GRID, use this option to provide the values you want FIND to search over. The VECT[VECT] should contain one vector for each parameter in the parameter set being estimated.

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 4.2 in the User’s Guide.

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 as the other methods don’t assume enough about the function to allow standard errors calculations. 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 4.5 of the User’s Guide) are unlikely to be interpretable as standard errors.

[print]/noprint
vcv/[novcv]

title=”description of optimization being done”

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.

Statement Block

FIND is actually very similar to the instruction LOOP (User’s Guide, Section 15.1). A function evaluation will execute all instructions between the FIND and the END FIND. A BREAK instruction within the statement block will cause 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. 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 code below does a preliminary grid search for $C$ (estimating the other parameters) then estimates a final model starting at the best value found by the grid search.

```
stats(fractiles) y
compute grid=%seqa(%fract01,(%fract99-%fract01)/99.0,100)
nonlin(parmset=conly) c
nonlin(parmset=gammaonly) gamma
find(parmset=conly,method=grid,grid=||grid||) min %rss
  nlls(frml=star,parmset=regparms+gammaonly,noprint) y
end find
nonlin(parmset=startparms) gamma c
nlls(frml=star,parmset=regparms+startparms,print) y
```

The next example is more complex. 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.

```plaintext
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
```

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 4.5 in the User's Guide).

![Output Table]

Variables Defined

- **%FUNCVAL**: Final value of expression (REAL)
- **%BETA**: The estimated parameter values (VECTOR)
- **%NFREE**: The number of free parameters (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)
- **%LAGRANGE**: VECTOR of Lagrange multipliers if estimating with constraints and METHOD=BFGS.
FIXED — Setting Values of Arrays in PROCES 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.

```plaintext
fixed datatype array(dimensions)
  values for the array (separated by spaces)
```

**Parameters**
- `datatype`: can indicate an array or array of arrays of integer, real, complex, label or string values.
- `array(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.

  These must be literal values; expressions are not allowed.

  If you use an array of arrays, such as:

  ```plaintext
  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** or **COMPUTE**.
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.

**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() cseries start end newcseries newstart
```

**Parameters**
- **cseries**: Source 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 \( N/\text{reduction factor} \) 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=768
freq 5 nords
rtoc 1966:1 2017:11
#  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 768/3=256 entries, which are entries 1+257+513, 2+258+514,... 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 5 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 (if no MODEL)
# first period shocks (only with INPUT option)
# list of path series (only with PATHS option)
```

Wizards

The Time Series—VAR (Forecast/Analyze) wizard can 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 Time Series—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 versions before 7 have been replaced by the STEPS, FROM, and TO options described below. RATS 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
newstart
```

The equation name or number.

(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.

(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*

Of the two ways to input the form of the model to be solved (the other is with supplementary cards), this is the more convenient 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 *i*th 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**.
**Forecast**

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 SKIPSAVE Option".

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 8.3)

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. See "Options for Adding Shocks".

**Variables Defined**

%FSTART Starting entry of forecasts (INTEGER)

%FEND Ending entry of forecasts (INTEGER)

**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 (unless you use the STATIC option).

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 8.3 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 2019:1, but one of your series has no data for 2018:4 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 7.7.2 in the *User’s Guide* for information on methods of incorporating all known data into the forecasts.

**Examples**

system(model=canmodel)
variables canrgdps canmls 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 2018:7. The GRAPH graphs both the forecasts and the last year of actual data.

forecast(print,from=2018:7,to=2020:6) 1
# unempeq foreun
graph 2
# unemp %fstart %fend
# foreun
Next, equation \texttt{M1EQ} has the log of \texttt{M1} (\texttt{LOGM1}) as its dependent variable, \texttt{RATEEQ} is an equation for an interest rate, and \texttt{PRICEQ} is for \texttt{LOGPRICE}. This computes forecasts over 2018:2 to 2022:4 (fifteen quarters) and appends them to the original data series. The \texttt{SET} instructions take the anti-logs of \texttt{LOGM1} and \texttt{LOGPRICE}, and the \texttt{PRINT} instruction prints the forecasts along with the last five quarters of data. This uses the variables \%FSTART and \%FEND defined by \texttt{FORECAST} to simplify the steps after doing the forecasts.

\begin{verbatim}
forecast(from=2018:2,steps=15) 3
# mleq logm1
# rateeq rate
# priceq logprice
set m1 %fstart %fend = exp(logm1)
set price %fstart %fend = exp(logprice)
print %fstart-5 %fend m1 rate price
\end{verbatim}

You can do the same operation using a \texttt{GROUPed} system by incorporating definitional identities for the log relationships:

\begin{verbatim}
frml(identity) m1iden m1 = exp(logm1)
frml(identity) priceiden price = exp(logprice)
group smallmod mleq rateeq>>rate priceeq $
   m1iden>>m1 priceiden>>price$
forecast(model=smallmod,from=2018:2,to=2022:4,print)
\end{verbatim}

\section*{Options for Adding Shocks}

\begin{itemize}
  \item \texttt{matrix=RECTANGULAR array of shock paths}
  \item \texttt{paths/[nopaths]}
    \begin{itemize}
      \item You can use \texttt{MATRIX} or \texttt{PATHS} to input paths of shocks over more than one period. With \texttt{MATRIX}, the columns of the \texttt{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 \texttt{steps}. Shocks will be zero for steps beyond those supplied by the array.
    \end{itemize}
  \item \texttt{input/[noinput]}
  \item \texttt{shocks=VECTOR for first period shocks}
    \begin{itemize}
      \item You can use either of these options to input general first period shocks. With \texttt{INPUT}, a supplementary card provides the values; with \texttt{SHOCKS}, the indicated \texttt{VECTOR} provides the shocks.
    \end{itemize}
\end{itemize}
Examples with Added Shocks

```
group macro  conseq>>f_cons investeq>>f_invest  $
   rateeq>>f_rate  gnpid>>f_gnp
set m = m{1}*1.015
forecast(input,model=macro,from=2018:3,to=2021:2)
#  0.0  0.0  0.5
```

forecasts a small macro model, assuming 6% (compound) growth in the (exogenous) money stock and “add factoring” the interest rate equation by .5.

```
boot select / 1960:1 2018:4
set path1 = resids1(select)
set path2 = resids2(select)
forecast(paths,from=1960:1,to=2018:4) 2
# x1eq x1simul
# x2eq x2simul
# path1  path2
```

generates \textsc{X1SIMUL} and \textsc{X2SIMUL} as bootstrapped realizations of a model. \texttt{BOOT} combined with the two \texttt{SET} instructions creates a path of residuals selected randomly from the estimation period.

Using the \texttt{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. Suppose you are doing a rolling regression with quarterly data and are only interested in the forecasts for one year past the end of the estimation sample. You can’t do:

```
do end=2009:1,2018:1
   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 \texttt{SKIPSAVE} option:

```
do end=2009:1,2018:1
   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 \texttt{YFORE} series. When the loop is complete, \texttt{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.

```
frequency  cseries  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 lengths with small prime factors (2, 3 and 5) than for lengths that 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).

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.
Examples

frequency  8  512

Creates 8 complex series with 512 entries.

compute nords = 2*%freqsize(2017:12)
frequency  6  nords

This creates 6 complex series with twice the recommended length. There are some applications where using at least double the number of frequencies compared to the data length is required.

Variables Defined by FREQUENCY

%Z is a RECTANGULAR[C,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.

\[
\text{frml (options) formulaname} = \text{function}(T)
\]

\[
\text{frml (options) formulaname depvar} = \text{function}(T)
\]

\[
\text{frml (regressors, other options) formulaname depvar}
\]

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.

Wizard

You can use the Statistics—Equation/FRML wizard to define formulas.

Parameters

- formulaname: The symbolic name you are assigning to this formula.
- depvar: (Optional) Dependent (or left-side) variable for full equations.

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
adddpars/[noadddpars]
  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
  VECTOR. 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
  coefficients 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 pa-
    rameter set.

  You must also use VECTOR or NAMES to specify how the parameters are created.

coeffs=VECTOR of (fixed) coefficients
  If you use EQUATION, LASTREG, or REGRESSORS and want the FRML coefficients
  to be treated as fixed at a particular set of coefficients (other than the ones cur-
  rently in the input EQUATION or the last regression if you're using LASTREG), use
  the COEFFS option to input those. Note that the FRML coefficients are fixed at the
  values at the time that the FRML instruction is executed—if you want to be able
  to change them, later, use the VECTOR option instead (even if you don't plan on
  using the FRML for estimation).

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.
Description

**FRML** defines a function of the form

\[ f(X_{1t}, X_{2t}, \ldots, X_{kt}, \beta_1, \beta_2, \ldots, \beta_p) \]

or

\[ y_t = f(X_{1t}, X_{2t}, \ldots, X_{kt}, \beta_1, \beta_2, \ldots, \beta_p) \]

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

`formulaname(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

```plaintext
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.

```plaintext
frml(identity) gdpid gdp = consmptn+invest+govtexp+netexp
frml(identity) logmlid logml = log(ml)
```

```plaintext
sur(inst) 3
# conseq
# inveq
# wageeq
```

```plaintext
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 $X_1$ and $X_2$ with first-order autocorrelation correction using common factor restrictions. (Note: these should give almost identical results to AR1).

```plaintext
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

or:

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
```

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 $B_1$, $B_2$ and $B_3$ for the three regression coefficients.
- Initializes $B_1$, $B_2$ and $B_3$ to the values estimated by LINREG.
- Adds $B_1$, $B_2$ and $B_3$ 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

\[
\text{frml truncate} = \%\text{logdensity}(\text{sigmasq}, y-\text{rhsfrml}) - \log(\%\text{cdf}((\text{rhsfrml}(t)-\text{tr})/\sqrt{\text{sigmasq}}))
\]

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:

\[
\text{frml truncate} = (z=\text{rhsfrml}(t)) , \%\text{logdensity}(\text{sigmasq}, y-z) - \log(\%\text{cdf}((z-\text{tr})/\sqrt{\text{sigmasq}}))
\]

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.

ARCH and GARCH models are the most common types of such recursively defined functions. For instance, in an ARCH-M model, the variance depends upon the lagged residual and the residual depends upon the variance. We can’t write down two simple FRML’s for this without each (at least indirectly) referencing itself. While this can be done with GARCH, we’ll show how to define FRML’s to do it. The first of these uses the \(UU\) series to save the squared residuals for use in the variance formula.

\[
\text{set u} = 0.0 \\
\text{set uu} = \%\text{seesq} \\
\text{frml archvar } = a0 + a1 * uu{1} \\
\text{frml regresid } = y - b1 - b2 * x1 - b3 * \sqrt{\text{archvar}(t)} \\
\text{frml archlogl } = u(t)=\text{regresid}(t), uu(t)=u(t)^2, \%\text{logdensity(archvar(t),u)}
\]

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,

def 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.

def vector b(n)
def 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 used for many of the components of a state-space model by DLM.

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 statement 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 15.2 of the User’s Guide). However, procedures are designed to mimic RATS instructions, and thus cannot be used from within an expression. Functions, on the other hand, are designed to be used in expressions, and thus make it possible to embed complicated 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 save 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 (set on the File—Preferences operation).

Indirect References

You can set up a PROCEDURE or FUNCTION to accept a FUNCTION as an option or parameter, and then pass through to it when you execute a specific function. The type specification for such as indirect function reference is

```
FUNCTION[returntype](argtype1,argtype2,...,argtypeN)
```

if it has N arguments. For instance,

```
procedure MCVARDoDraws
*
  option model      model
  option integer    steps   48
  option integer    draws   1000
  option vect[int]  accum
  option function[rect](symm,model) ffunction

This adds an option called FFUNCTION (for Factor FUNCTION) to the procedure. This takes a SYMMETRIC and MODEL as its parameters. Within the procedure, this is used as:

```
if %defined(ffunction)
  compute factor=ffunction(%outerxx(fsigmad),model)
else
  compute factor=fsigmad
```

This checks to see if FFUNCTION has been defined. If FFUNCTION isn’t used in executing @MCVARDO draws, it won’t be. If it is defined, it gets used to compute FACTOR; if not FSIGMAD (which in this case is the Cholesky factor) gets used instead.

```
function bqfactor sigma model
  type rect bqfactor
  type symm sigma
  type model model
  *
  compute bqfactor=%bqfactor(sigma,%modellagsums(model))
end
```
Examples

This defines a VECTOR of probabilities in a regime-switching model, return a 2-VECTOR with the density functions in the two regimes, where the regimes differ based upon the values PHI.

```ratml
function RegimeF time
type vector  RegimeF
type integer  time
local integer  i
*
dim RegimeF(2)
ewise RegimeF(i)=exp(%logdensity(sigsq,$
    %eqnrvalue(xeq,time,phi(i))))
end
```

This computes the tri-gamma function (second derivative of log gamma) of its argument. (Note: the function %TRIGAMMA is now built-in, but this offers a good example of writing a general purpose function). 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 the function name (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.

```ratml
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
```
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 four times faster on average. See Chapter 9 in the User’s Guide 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

There are separate Time Series—ARCH/GARCH(Univariate) and Time Series—ARCH/GARCH(Multivariate) wizards. However, the GARCH instruction has several options not included in the Wizards, so be sure to refer to the information below if you need capabilities beyond those available using the wizards.

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.

- mean/nomean  [default is MEAN unless you use REGRESSORS]
  
  MEAN includes a constant term in the mean equation(s). Use NOMEAN if you want
  n 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.
distrib=[normal]/t/ged
shapeparm=\textit{input value for shape parameter for t or GED [estimated]}
The assumed distribution of the error process, Normal, t (Student) or Generalized Error Distribution. If using T or GED, you can use SHAPEPARAM to provide your own value for the shape parameter. If you don’t, it will be estimated.

asymmetric/[noasymmetric]
ASYMMETRIC 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.

hadjust=\textit{FRML to be evaluated after variance calculation}
uaadjust=\textit{FRML to be evaluated after residual calculation}
These allow “on-the-fly” calculations using current variances or residuals, respectively, simplifying the coding for many types of GARCH-M and related models. If you use HADJUST, the variances at entry T are computed first, then the HADJUST expression is computed based on those updated variances. RATS then computes the time T residuals and, if there is a UADJUST option, that expression is evaluated using the updated variances and residuals. These work with both univariate and multivariate models, and you can include the variances and residuals in the expression by referencing the variable names supplied on the HSERIES or HMATRICES (variances) and RESIDS or RVECTORS (residuals) options.

condition/[nocondition]
If CONDITION, GARCH conditions on the required lagged residuals rather than assigning them the presample values.

i=nodrift/drift
This constrains the GARCH coefficients (all a’s and b’s) to sum to one. (In an egarch model only the b’s are included). For multivariate models, each component is constrained separately. With I=NODRIFT, constant terms in the variance equations are constrained to zero. With I=DRIFT, constant terms are estimated. You cannot use I with MV=BEKK or MV=VECH.

presample=\textit{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.

likelihood=(output) \textit{SERIES of log likelihood elements}
Use this to save the entry by entry log likelihood values.
ARCH/GARCH Options—Univariate Models Only

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. For any other mean equation, use the REGRESSORS option and include all 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]

EXPONENTIAL does 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)

These save the estimated variances and residuals, respectively, into SERIES.

ARCH/GARCH Options—Multivariate Models Only

mv=[standard]/bekk/diag/cc/dcc/vech/adcc/ewma/tbekk/dbekk/cholesky

Chooses the form for a multivariate model. See Section 9.4 in the User’s Guide for details. The default is the standard multivariate GARCH model.

BEKK gives the “BEKK” formulation (also known as BEK or EK), which imposes positive-definiteness on the covariance matrix. DBEKK (diagonal BEKK) restricts the lagged variance and residual terms to have diagonal multiplying matrices. TBEKK (triangular BEKK) restricts the lagged variances and residuals so the front multiplier matrix is lower triangular—the results will depend on variable order.

VECH estimates the “vech” model. EWMA implements an exponentially weighted moving average model—an IGARCH model with a single parameter applying to all components. The other choices are restricted correlation models. DIAGONAL estimates separate univariate GARCH models for each dependent variable, so the covariances are restricted to zero. CC gives the Constant Correlation model, DCC implements the Dynamic Conditional Correlations model, while ADCC does an asymmetric Dynamic Conditional Correlations model. CHOLESKY is an alternative to CC/DCC that does univariate GARCH models and then combines them into full covariance matrix by $H=LDL'$ where $L$ is lower triangular with 1’s on the diagonal and $D$ is the diagonal matrix of the variances of the univariate models. Results depend upon ordering.

variances=[simple]/varma/exponential/spillover/koutmos

Use the VARIANCES option to select the form of the variance terms.

model=MODEL (of linear equations) for the mean equations

Use MODEL to supply your own mean model. You can create the model using SYSTEM, or use LINREG or EQUATION commands to define the mean equations, then use GROUP to group them into a model. Otherwise, the MEAN/NOMEAN option determines the form of the mean equation.
hmatrices=SERIES[SYMM] estimated covariance matrices
mvhseries=SYMM[SERIES] estimated variance/covariance series
rvectors=SERIES[VECT] of non-standardized residuals (multivariate)
HMATRICES saves the estimated covariance arrays as a SERIES of SYMMETRIC arrays while MVHSERIES saves the variances and covariances as a SYMM[SERIES]. RVECTORS saves the (non-standardized) residuals to a SERIES of VECTORS. Note that HMATRICES and MVHSERIES save the same information, just in different variable types.

stdresid=(output) VECT[SERIES] of (jointly) standardized residuals
factorby=[cholesky]/eigendecomp
If the model is correct, then the jointly standardized residuals should be serially uncorrelated with mean zero and covariance matrix equal to the identity matrix with no residual GARCH effects. Given the model residuals and the series of estimated covariances, there are many (that is, an infinite number) of ways to transform them into jointly standardized residuals. GARCH gives you the option of a Cholesky factorization or an eigen-based one. Note that different ways of doing the standardization will result in different standardized residuals and thus different diagnostics.

vechmat=(output) HASH[RECT] of matrices for a VECH representation
STANDARD, BEKK, VECH, TBEKK and DBEKK models are all restricted forms of a full VECH representation. For those models, you can use the VECHMAT option to create a HASH[RECT] with the matrices for a VECH representation of the model. That can be used for forecasting and variance impulse responses. Note that only these restricted VECH models allow a relatively simple forecasting formula. The elements of the HASH are "C" for the variance constant, "A" for the coefficients on the (vech'ed) outer product of the lagged residuals, "B" for the coefficients on the (vech'ed) lagged covariance matrix and "D" (if needed) for the coefficients on the (vech'ed) outer product of the sign-restricted residuals if you used asymmetry.

signs=VECTOR of triggers for asymmetric behavior [-1 for all]
Use the SIGNS option to provide +1 or -1 values for the signs that trigger asymmetric behavior in each component. See page 300 in the User’s Guide for details.

xbekk=[combined]/separate
Determines how "X" regressors for a BEKK model are handled. Both of these add a lower triangular matrix to the parameter set for each X regressor. The default (COMBINED) does a linear combination of the constant variance matrix plus the X effects and squares the combined matrix:

\[(C + \sum E_i x_{it} + \ldots + E_k x_{kt})(C + \sum E_i x_{it} + \ldots + E_k x_{kt})^\prime\]

The SEPARATE choice takes the outer product of each term separately:

\[CC^\prime + x_{it}^2 E_i E_i^\prime + \ldots + x_{kt}^2 E_k E_k^\prime\]
Both preserve the positive-definiteness of any covariance matrix generated. The latter, however, means that each of the X terms must increase the variance, which may be too strong an assumption in many cases. (This option was added with version 9.2).

DCC=[COVARIANCE]/CORRELATION/CDCC

QBAR=initial $Q$ matrix for DCC [empirical]

DCC chooses the form of the recursion for the model for the "GARCH" submodel that determines the dynamic correlations. The default is DCC=COVARIANCE, which means

$$Q_t = (1-a-b)\bar{Q} + au_{t-1}u'_{t-1} + bQ_{t-1}$$

where $u_t$ is the mean model residual and $\bar{Q}$ is the unconditional covariance matrix. DCC=CORRELATION is Engle’s original idea of

$$Q_t = (1-a-b)\bar{Q} + a\varepsilon_{t-1}\varepsilon'_{t-1} + bQ_{t-1}$$

where $\varepsilon_t$ is the vector of standardized residuals ($u_t$ divided by the model estimate for its standard deviation) and $\bar{Q}$ is the sample covariance matrix of the $\varepsilon_t$. Engle’s recursion is really designed for “two-step” estimation procedures which estimate univariate models first and take the standardized residuals as given in a second step of estimating the joint covariance matrix. However, it can't be applied easily with any of the VARIANCE models which have interactions among the variances from different equations. It also, in practice, rarely fits better than DCC=COVARIANCE, and often fits quite a bit worse. (It tends to be more easily dominated by outliers.) DCC=COVARIANCE is the calculation that GARCH has been doing since DCC was added and remains the default.

DCC=CDCC (corrected DCC) is based upon the suggestion in Aielli(2013). Aielli shows the Engle’s dcc recursion isn’t internally consistent, in the sense that $E\varepsilon_t\varepsilon'_t \neq Q_t$ as it would be if the secondary model were actually "GARCH" and proposes an alternative which fixes that. Aielli’s model has the same problem as Engle’s that it doesn’t really apply except to models which can be done in two steps (mainly because the calculation of the implied $Q$ requires that the individual variances be known), so DCC=CDCC uses a “feasible” estimator of the sample correlation matrix.

For whichever model is chosen for DCC, the QBAR option can be used to override the sample calculation of the $Q$ in case you want to experiment.
General Estimation Options

The following are common to most of the non-linear estimation commands in RATS.

\textbf{method=bhhh/[bfgs]/simplex/genetic/annealing/ga/evaluate}

This chooses the method used for the final estimation. The two most common choices for this are \texttt{BFGS} (the default) and \texttt{BHHH}. These are the only two which can produce standard errors—the others don't assume enough to permit that. Instead, those are derivative-free methods which can compute point estimates of the coefficients. They can be helpful in refining initial guess values and are usually used as a \texttt{PMETHOD} for preliminary estimation. See Chapter 4 in the \textit{User's Guide} for a technical description of these.

With \texttt{METHOD=EVALUATE}, \texttt{GARCH} simply evaluates the model given the initial parameter values (input using the \texttt{INITIAL} option), without trying to estimate new coefficient values. You can use options like \texttt{HSERIES} and \texttt{RESIDS} to save the resulting variance and residual series for evaluation.

\begin{verbatim}
iterations=iteration limit [200]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]
\end{verbatim}

\texttt{ITERATIONS} sets the maximum number of iterations, \texttt{SUBITERS} sets the maximum number of subiterations, \texttt{CVCRIT} the convergence criterion. \texttt{TRACE} prints the intermediate results. See Chapter 4 in the \textit{User's Guide} for more details.

\begin{verbatim}
pmethod=bhhh/bfgs/[simplex]/genetic/annealing/ga
piter=number of PMETHOD iterations to perform [none]
\end{verbatim}

Use \texttt{PMETHOD} and \texttt{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 \texttt{BFGS}, use \texttt{PMETHOD=SIMPLEX}, \texttt{PITERS=10}, and \texttt{METHOD=BFGS}.

\begin{verbatim}
[print]/noprint
vcv/[novcv]
title="description of estimation" [depends upon options]
\end{verbatim}

These control the printing of regression output, the printing of the estimated covariance/correlation matrix of the coefficients (page Int–77 of the \textit{Introduction}) and the title of the output. By default, the title will be the type of model (“GARCH Model”, “E-GARCH Model”, etc.).

\begin{verbatim}
robusterrors/[norobusterrors]
\end{verbatim}

\begin{verbatim}
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]
\end{verbatim}
These permit calculation of a consistent covariance matrix allowing for heteroscedasticity or (with LAGS) serial correlation (you need the ROBUSTERRORS option in either case). For more information, see Sections 2.2, 2.3 and 2.4 of the User's Guide and "Long-Run Variance/Robust Covariance Calculations" on page RM–540.

**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" 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 4.2 in the User's Guide.

**derives**=(output) VECTOR[SERIES] for partial derivatives [unused]

This saves the series of partial derivatives of the log likelihood. The first series in the VECTOR will be the partials with respect to the first parameter displayed in the GARCH output, the second series will be the partials with respect to the second parameter, and so on.

**parmset**=PARMSET to use for additional parameters [not used]

**density**=FUNCTION to compute the log density [not used]

You can use these to extend the GARCH instruction to handle certain types of additional models. The DENSITY option FUNCTION must take two arguments: the first is the REAL variance (univariate) or SYMMETRIC covariance matrix (multivariate) produced by the GARCH recursion, and the second is the REAL residual (univariate) or VECTOR of residuals (multivariate). Using those, it should return the log density to be used for the likelihood element at the current entry. You can use in the FUNCTION any of the parameters in the PARMSET that you provide, and you can also use the current entry number T if something is time-varying. For instance,

```plaintext
nonlin(parmset=switchparms) nu1 nu2
function logswitcht h u
  type real h u
  compute logswitcht=%logtdensity(h,u,%if(t<=1000,nu1,nu2))
  end
compute nu1=nu2=10.0
garch(parmset=switchparms,density=logswitcht) / dlogdm
```

will estimate a standard GARCH model but with t distributed errors with different degrees of freedom for the first 1000 observations and the remainder of the sample. The added parameters will come at the end of the output. Note that you do not use DISTRIB=T with this (that would add a SHAPE parameter that has no effect, since you're overridden the distribution calculation).
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)
7. Shape parameter (degrees of freedom for t) for non-normal errors (if any)

For multivariate models, many of these are matrices. Within each category, elements are ordered by row. For symmetrical or lower triangular matrices, 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. With more than one lag, each lag is in a separate block.

Variables Defined by GARCH

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BETA</td>
<td>Coefficient VECTOR</td>
</tr>
<tr>
<td>%XX</td>
<td>Covariance matrix of coefficients (SYMMETRIC)</td>
</tr>
<tr>
<td>%TSTATS</td>
<td>VECTOR containing the t-stats for the coefficients</td>
</tr>
<tr>
<td>%STDERRS</td>
<td>VECTOR of coefficient standard errors</td>
</tr>
<tr>
<td>%NOBS</td>
<td>Number of observations (INTEGER)</td>
</tr>
<tr>
<td>%NREG</td>
<td>Number of regressors (INTEGER)</td>
</tr>
<tr>
<td>%NFREE</td>
<td>Number of free parameters (INTEGER)</td>
</tr>
<tr>
<td>%NVAR</td>
<td>number of variables (INTEGER)</td>
</tr>
<tr>
<td>%RESIDS</td>
<td>Series containing the residuals (SERIES)</td>
</tr>
<tr>
<td>%FUNCVAL</td>
<td>Log likelihood (REAL)</td>
</tr>
<tr>
<td>%ITERS</td>
<td>iterations completed (INTEGER)</td>
</tr>
<tr>
<td>%CONVERGED</td>
<td>=1 or 0. 1 indicates that the process converged (INTEGER)</td>
</tr>
<tr>
<td>%CVCRT</td>
<td>final convergence criterion (REAL). This will be equal to zero if the sub-iteration limit was reached on the last iteration</td>
</tr>
<tr>
<td>%LOGL</td>
<td>Log likelihood (%FUNCVAL and %LOGL are identical) (REAL)</td>
</tr>
</tbody>
</table>
Examples

ARCH(2) on an AR(1) regression model, with the presample variance fixed at the value of %SEESQ from a previous regression.

```
garch(p=0,q=2,regressors,presample=||%seesq||) / 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,dist=t,shape=5,resids=at,hseries=fvar) / rt
```

EGARCH(1,1) model with asymmetry on an MA(1) mean model.

```
garch(p=1,q=1,exp,asymmetric,regressors) / ibmln
  # constant %mvgavge{1}
```

GARCH-X(1,1), adding the variable $U$ to the variance equation.

```
garch(xreg,p=1,q=1,resids=at,hseries=fv) / rt
  # U
```

Bivariate GARCH(1,1) model

```
garch(p=1,q=1,method=bfgs) / futures spot
```

Trivariate DCC GARCH(1,1) model, with separate mean models for each variable.

```
group meanm rspeq rnkeq rhseq
  garch(p=1,q=1,mv=dcc,model=meanm,method=bhhh)
```

This uses HADJUST to include the square root of the variance of the third equation (the OILGROW equation) as a regressor in the mean equation of a var garch model. This updates the variable HOIL as a function of the variances, which are being saved in the (SERIES of SYMMETRIC arrays) HH:

```
set hoil = 0.0
system(model=basevar)
  variables pgrow ipgrow oilgrow rate
  lags 1 to nlags
  det constant hoil
end(system)
garch(model=basevar,mv=diag,hmatrices=hh, $
  hadjust=%(hoil=sqrt(hh(t)(3,3))))
```
GBOX — Graphing Box Plots

**GBOX** produces box plots (also known as “box-and-whisker” plots) for one or more series. Box plots provide a simple graphical representation of some of the basic statistical properties of one or more series, including the median, the interquartile range, the maximum and minimum values, and significant outliers.

```
gbox( options ) number hfield vfield
# series start end  (one card for each series)
```

**Wizard**

If you generate a series list, you can use the toolbar to generate a quick box plot from selected series.

**Positioning**

When using **SPGRAPH** to put multiple graphs on a page, RATS normally fills the page by column starting at the top left (field 1,1). To place a graph in specific field, use either the **ROW** and **COL** options or the **hfield** (column) and **vfield** (row) parameters.

**Parameters**

- **number**       Number of series to graph. The maximum permitted is twenty.
- **hfield**       See “Positioning” above.
- **vfield**       See “Positioning” above.

**Supplementary Cards**

Use one supplementary card for each series you want to plot.

- **series**       the series to be plot.
- **start**        (Optional) the range to use in generating the box plot. 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.

**Options**

Most of the **GBOX** options are the same as those available on the **GRAPH** instruction. These common options are listed briefly below—see **GRAPH** for details on these. The options that are unique to **GBOX** are described in more detail below.

- **[axis]/noaxis** Draw horizontal axis if Y=0 is within bounds
- **col=column number** Column for this graph in the **SPGRAPH** matrix.
- **extend/[noextend]** Extend horizontal grid lines across graph
- **footer=footer label** Adds a footer label below graph
frame=[full]/half/ Controls frame around the graph
   none/bottom
header=string Adds a header to the top of the graph
height=value Sets height of graph in inches
hlabel=label Adds a label to the horizontal axis
log=value Base for log scale graphs
max=value Value for upper boundary of graph
min=value Value for lower boundary of graph
picture=pict.code Picture code for axis label numbers
row=row number Row for this graph in the SPGRAPH matrix.
scale=[left]/right/both/none Placement of vertical scale
   smpl=series or frml Series/formula indicating entries to be graphed
   subhead=string Subheader string for graph
   vgrid=vector Values for grid lines across from vertical axis.
   vlabel=label Label for the vertical axis
   vticks=number or VECTOR Maximum number or specific list of vertical ticks
   width=value Sets width of graph in inches
   window=string Title for graph window

Options Specific to GBOX

labels=VECTOR[string] of labels for the plots [series names] By default, GBOX labels the x-axis with the names of the series being plotted. If you prefer, you can provide your own labels using the LABELS option. The first box plot will be labeled with the first string in the vector, the second plot with the second string, and so on.

group=SERIES or FRML with distinct values for each group Use this option to have GBOX divide the series being graphed into groups, or sub-samples, based on the values of this series or formula. For example, if you supply a 0/1 dummy variable, GBOX will do two plots for each series—one plot using observations where the GROUP series contains zeros, and another for observations where the GROUP series contains ones.

Examples
This is generated using data taken from Johnston and DiNardo (1997). The resulting box plot demonstrating the elements is shown below.

open data cps88.asc
data(format=prn,org=columns) 1 1000 age exp2 grade ind1 $
   married lnwage occl partt potexp union weight high
Gbox
# lnwage
Elements of a Box Plot

Box plots include the following elements:

- A line representing the median value
- A box representing the interquartile range (IQR). The top and bottom lines of the box correspond to the 75th and 25th percentiles, respectively.
- Vertical lines, or “whiskers” indicating 1.5 times the IQR in either direction from the 75th and 25th percentiles. This is about 2.7 standard deviations on either side of the median for a Normal series.
- Short horizontal lines representing the maximum and minimum values.
- Dots representing outliers (if any). Outliers are defined as values that fall outside 1.5 times the IQR in either direction.
This draws a separate box plot for each year’s worth of data in the quarterly series WASH.

```
open data washpower.dat
calendar(q) 1980
data(format=free,org=columns) 1980:1 1986:4 wash
gbox(group=%year(t),$
# wash
```
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.

\[
g\text{contour } (\text{options } ) \quad h\text{field} \quad v\text{field}
\]

Positioning

When using SPGRAPH to put multiple graphs on a page, RATS normally fills the page by column starting at the top left (field 1,1). To place a graph in specific field, use either the \texttt{ROW} and \texttt{COL} options or the \texttt{hfield} (column) and \texttt{vfield} (row) parameters.

Parameters

\[
h\text{field} \quad v\text{field} \quad \text{See “Positioning” above.}
\]

Options—Specific to GCONTOUR

The following is a list 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.

\[
x=\text{VECTOR of } X \text{ grid values } [\text{required}]
\]

\[
y=\text{VECTOR of } Y \text{ grid values } [\text{required}]
\]

\[
f=\text{RECTANGULAR array supplying function values } [\text{required}]
\]

Use \texttt{X} to supply a vector of grid values for the \texttt{x}-axis, and \texttt{Y} to supply a vector of grid values for the \texttt{y}-axis. Use \texttt{F} to supply a \texttt{RECTANGULAR} array with dimensions \(\dim(x) \times \dim(y)\) containing the function values for each \(x,y\) grid point. For example, \texttt{F(1,2)} should contain the function value associated with the grid point given by \(X(1),Y(2)\). \texttt{X} and \texttt{Y} are usually created using the \%SEQA function or an \texttt{EWISE} instruction. The \texttt{F} array is usually set using \texttt{EWISE}.

\[
\text{number}=\text{number of contour lines } [30]
\]

\[
\text{contours}=\text{VECTOR of specific } \texttt{f} \text{ values}
\]

Use \texttt{CONTOURS} to supply a \texttt{VECTOR} containing the specific function values at which you want contour lines to be plotted. Otherwise, RATS will draw the number of lines specified by the \texttt{NUMBER} option, with the contour values chosen to provide a roughly equal distance between each value.

Options—Common to SCATTER and GCONTOUR

\[
\text{axis}=\text{none/vertical/horizontal/}[\text{both}]
\]

\[
\text{col}=\text{column number}
\]

\[
\text{extend}=\text{[none]/vertical/horizontal/both}
\]

\[
\text{footer}=\text{footer label}
\]

\[
\text{frame}=\text{[full]/half/none/bottom}
\]

\[
\text{Draw } x=0 \text{ and/or } y=0 \text{ axes}
\]

\[
\text{Column in SPGRAPH matrix.}
\]

\[
\text{Extend grid lines across graph}
\]

\[
\text{Adds a footer label below graph}
\]

\[
\text{Controls frame around graph}
\]
GContour

header=header string for graph
height=height in inches
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 code for x-axis
hscale=[lower]/upper/both/none
hshade=RECTANGULAR with shading zones
hticks=number or VECTOR
lines=RECTANGULAR with slope/intercept
row=row number
subhead=subheader string
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 code for y-axis
vscale=[left]/right/both/none
vshade=RECTANGULAR with shading zones
vticks=number or VECTOR
width=width in inches
window=string for window title
xlabels=VECT[STRING] for x-axis labels

Example

This does contours of the likelihood surface for a GARCH(1,1) model. To avoid an ex-cessive number of contours, the function values are set to NA if too small.

```r
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)
```
**GOTO (or BRANCH)— Go 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 **GOTO**. You can also use **BRANCH** as a synonym for **GOTO**.

```
goto gotolabel
```

**Parameters**

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

**Placing Branch Labels in a Program**

The **GOTO** instruction and the **gotolabel** must be part of the same compiled section, although the **GOTO** instruction can be “nested” deeper than the **gotolabel**, 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 **gotolabel** with a colon (:). Put at least one blank between the end of the label and the next instruction.

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

```
GOTO 100
   dofor xmat = amat bmat cmat
      :100 matrix xmat=inv(xmat)
   end dofor
```

**Example**

In the example below, there is no good alternative to **GOTO**. 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
           goto done
      end do j 
   end do i
   :done
   display "Result Achieved At Variable" i "Lag" j
}
```
GRAP—High-Resolution Time Series Graphs

*GRAPH* produces high-resolution time series graphs. It has a large number of options, which permit you to control most aspects of the presentation.

```
graph(options) number hfield vfield
# series start end stylenum (one card per series)
```

**Wizard**

*Data/Graphics—Graph* is the wizard for the *GRAPH* instruction. Note that in order to keep the Wizard from getting too complicated, some of the *GRAPH* options have been omitted in the wizard. 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 Window*. First, select View—*Series Window* 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. Those are for quick looks—they don’t generate instructions.

**Positioning**

If you’re using *SPGRAPH* to put multiple graphs on a single page, by default, the fields are filled by column, starting at the top left (field 1,1). If you want to fill a particular field instead, use either the combination of *ROW* and *COL* options or *hfield* (for the column) and *vfield* (for the row) parameters.

**Parameters**

- **number**: Number of series to graph.
- **hfield**  **vfield**: See “Positioning” above.

**Supplementary Cards (one per series, omit if using SERIES option)**

- **series**: the series to be graphed.
- **start**  **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.
- **stylenum**: (Optional) An integer value (1 through 30) specifying the line, fill, or symbol style 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 3.4 and Section 3.16 of the *Introduction* for details. A style number over 30 is “wrapped” to a number in range (41 is the same as 11).
Options

This is a brief alphabetical listing. Detailed descriptions organized by function follow.

[axis]/noaxis
[box]/nobox
col=column number
[dates]/nodates
extend/[noextend]
footer=footer label
frame=[full]/half/
none/bottom
grid=gridseries
header=string
height=value
hlabel=label
[kbox]/nokbox
key=[none]/upleft/upright/
loleft/loright/above/
below/left/right/attach
kheight=value
klabel=vect[strings]
kwidth=value
[ksample]/noksampling
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.code
row=row number
scale=[left]/right/
both/none
series=vector[series]
shading=series
smpl=series or frml

Draw horizontal axis if Y=0 is within bounds
Replaced by the FRAME option
Column for this graph in the SPGRAPH matrix.
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
Sets height of graph in inches
Adds 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 code for axis label numbers
Row for this graph in the SPGRAPH matrix.
Placement of vertical scale
VECTOR of SERIES to graph
Series with non-zeros at entries to be shaded
Series/formula indicating entries to be graphed
Graph

Style of the graph

style=[line]/polygon/bar/
stackbar/overlap/
vertical/step/symbol/
midpolygon/fan/dots/spike

Subheader string for graph

subhead=string

Style numbers to use (with SERIES option)

symbols=VECT[INTGER]

Put tick marks, dates/entries on horizontal axis

ticks/noticks

Values for grid lines across from vertical axis.

vgrid=vector

Label for the vertical axis

vlabel=label

Maximum number of vertical ticks

vticks=number

Sets width of graph in inches

width=value

Title for graph window

window=string

For labeling points on the horizontal axis

xlabel=VECT[STRINGS]

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 win-
dows are titled using either the HEADER or FOOTER strings; if you have neither of
those, nor WINDOW, they are “Graph.01,” “Graph.02,” etc.

footer=STRING for graph footer

Adds a left-justified label in the lower left corner of the graph. To prevent a footer
from getting too wide, you can use the characters \ to insert a line break.

frame=[full]/half/none/bottom

[frame]/nobox

FRAME controls the box displayed around the outside of the graph. HALF displays
the frame only to the left and below the graph, omitting the top and right sides.
BOX/NOBOX is an older option for controlling the box (frame) which is superseded
by the more flexible FRAME. NOBOX is equivalent to FRAME=None.

smpl=SMPL series or formula (Chapter 25 of the User's Guide)

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.

row=row number

col=column number

When using SPGRAPH, you can use ROW and COLUMN (or the hfield and vfield
parameters) to manually specify the position of the graph in the SPGRAPH grid.
**Graph**

\[ \text{style}=[\text{line}]/\text{polygonal}/\text{bar}/\text{stacked}/\text{overlapping}/\text{vertical}/\text{step}/\text{symbol}/\text{midpolygon}/\text{fan}/\text{dots}/\text{spike} \]

**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—use **SPIKE** instead.

**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.

**series=VECTOR[SERIES] of series to be graphed** [unused]

**symbols=VECTOR[integer] of style numbers** [unused]

The **SERIES** option provides a convenient way to graph all of the series stored in **VECTOR of SERIES**. Omit the supplementary cards when using the **SERIES** option. When using **SERIES**, you can also use **SYMBOLS** to supply a **VECTOR of INTEGERS** with the style numbers you want to use for the corresponding series.
Graph

height=height of graph in inches
width=width of graph in inches

These options specify a fixed size for the graph. You must use both options together to have RATS display the graph at the specified size. The “Fix” toolbar button will be “on”, freezing the proportions of the graph. RATS maintains the size if you print or save the graph. Resizing the window will resize the graph proportionally. Use the “Unfix” button if you want to change the proportions. To specify sizes in centimeters, use the %CM() function to convert centimeters to inches.

Graph Header Options

header= "Header string"  (in quotes: "...")  [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. You can also insert line breaks in the string, by including the characters "\" at the point where you want the line break.

subheader= "Subheader string"  (in quotes: "...")  [none]
The subheader is centered above the graph and below the header. By default, it is presented in 18 point italics. As with the HEADER option, you can use the characters "\" to insert a line break.

Time Axis Options

hlabel= "Horizontal scale label"  (in quotes: "...")  [none]
HLABEL is centered below the graph and below the entry labels (if any). As with the HEADER option, you can use the characters "\" to insert a line break.

[ticks]/noticks
NOTICKS suppresses the tick marks and entry labeling on the time axis.

[dates]/nodates
number=labeling number for first entry
By default, GRAPH labels entries with dates (if possible) on the horizontal axis. How RATS represents dates depends on the font sizes (controlled with GRPARM), the number of observations, and the size and shape of the graph. With a relatively short series of daily data, GRAPH will probably use full month names with dates at each entry. With long annual series, it may only label one out of every five years. Year or month labels are centered under entries covered by that date.

With NODATES, GRAPH labels graphs with entry numbers. You can use NUMBER (with NODATES) to use a number other than the entry number for the first observation. For instance, RATS stores autocorrelations with the 0 lag in entry 1. To label them correctly, use the options NODATES and NUMBER=0.
**Graph**

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 overrides the standard date or entry labels with the supplied labels. These will be spaced equally across the x-axis. If you don’t have one label per data point, be very careful to ensure 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 well.

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. 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 code 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. For more information, see Chapter 25 of the User's Guide.

[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.
Graph

\texttt{vticks=} maximum vertical ticks or VECTOR with specific values \[9\]
\texttt{extend/}[noextend]
\texttt{vgrid=} VECTOR of grid line values for the vertical axis \[unused\]

RATS normally marks the vertical axis with small tick marks. \texttt{VTICKS} sets the maximum number of labeled tick marks on the vertical scale, or you can provide a VECTOR with specific locations. The \texttt{EXTEND} option draws dotted lines across the full width of the graph (RATS ignores \texttt{EXTEND} for overlay/two-scale graphs). \texttt{VGRID} accepts a VECTOR of vertical-axis values at which horizontal lines will be drawn going across the graph.

Two-Scale (Overlay) Graph Options

Two-scale graphs are created with a single \texttt{GRAPH} instruction, using \texttt{OVERLAY} and the related options. See page Int–134 in the \textit{Introduction} for more information.

\texttt{overlay=} line/polygonal/bar/stacked/overlapping/
vertical/step/symbol/midpolygon/fan/dots/spike

\texttt{OVERLAY} makes the graph a two-scale graph, and specifies the style to use for the overlaying series (right-side scale series). See \texttt{STYLE} for details on the styles.

\texttt{ovsamescale/}[noovsamescale]

You can use \texttt{OVSAMESCALE} to force both the regular and the overlay series to share a common scale. They will just be shown in different styles.

\texttt{ovcount=} Number of series for right-side scale \[1\]

The last \texttt{Number} series listed on the supplementary cards are graphed using the right-side scale. The other series are graphed using the left-side scale.

\texttt{omax=} Maximum value for right-side scale \[largest\ value\]
\texttt{omin=} Minimum value for right-side scale \[smallest\ value\]
\texttt{ovlabel=} Vertical scale label (in quotes: \texttt{"..."} or \texttt{STRING}) \[none\]

\texttt{OMAX} and \texttt{OMIN} allow you to set the maximum and minimum values, respectively, for the right-side scale. These function like the \texttt{MAX} and \texttt{MIN} options (which control the left-side scale when doing a two-scale graph). \texttt{OVLABEL} allows you to supply a label for the right-side scale.

\[ovkey]/noovkey

You can use \texttt{NOOVKEY} to eliminate the key for the overlay series, if the meaning is either obvious, or provided using the labels.

\texttt{ovrange=} fraction of scale overlap \[1.0\]

\texttt{OVRANGE} offsets the left and right scales vertically, so that the \texttt{STYLE} side and the \texttt{OVERLAY} side are each graphed using only a portion of the available vertical space. With the default value of 1.0, both scales overlap fully (share the same vertical space). With \texttt{OVRANGE}=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 graph and Y-axis labeling
- RIGHT: right side, centered vertically, outside graph and Y-axis labeling
- ATTACHED: used with LINES and SYMBOLS styles, this puts the labels inside the graph near the lines or symbols, 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 series. Use KLABEL to supply your own labels. You can create the VECTOR[STRINGS] ahead of time, or enter it using the ||...|| matrix notation (see page UG–26 in the User’s Guide). The order of labels in the VECTOR should match the order of the supplementary cards. You can use \ in a string to put a line break in the string.

[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]

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. KHEIGHT and KWIDTH allow you to control the size and proportion of the box by specifying the height and width of the box as a fraction of the graph’s overall height and width. You must specify both options.

[ksample]/noksample

NOKSAMPLE eliminates the sample line styles, colors, or fill patterns 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.

For graphing a large number of series on a single graph, you may want to collect the series into a VECTOR of SERIES and supply that variable using the SERIES option rather than using supplementary cards. You can also use LIST and CARDS to automate the supplementary card list.
Graph

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 graph]

Color versus Black and White

Graphs are normally displayed in color, and output in color if you save the graph to a file or print it on a color printer. RATS will automatically convert the graph to a black and white style if you print it on a black and white printer.

You can also use the toolbar icon to preview the black and white version (click the button again to switch back to color). The PATTERNS option forces RATS to generate the graph in black and white mode only, which can be useful if you are only concerned with the black and white appearance.
Lines, Fill Patterns, and Symbols

When displaying graphs in color, RATS normally uses solid lines and fill patterns, and the first symbol in the table below for all series, using different colors to distinguish series.

For black and white display or output, RATS will switch to using the different line patterns, fill patterns, and symbols shown below.

The numbers in the left-hand column are the values for the \textit{stylenum} parameter that select the associated pattern or symbol when using the default graph style sheet. By default, style 1 is used for the first series being graphed, style 2 for the second graph, and so on.

<table>
<thead>
<tr>
<th>Number</th>
<th>Line Pattern</th>
<th>Fill Pattern</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Custom Styles

You can use the Graph Style Sheet feature to define and use your own custom styles. You can define your own colors, mix pattern and color attributes, adjust line thicknesses, and more. See Section 3.16 in the \textit{Introduction} and \texttt{GRPARM} for more information.
GROUP — Building General Simultaneous Models

Groups formulas and/or equations to create a MODEL for systems estimation, simulation, forecasting and DSGE model solution. There is no limit on the number of equations and formulas you can include in a model.

\[ \text{group( options ) model list of formula} \gg \text{result fields} \]

Parameters

- **model**: The MODEL being created or modified.
- **formula} \gg \text{result}**: Each field gives the name of an EQUATION or FRML. 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. For example, you might want to save the simulations for some formulas in the model, but not for others. (In most cases, the instruction(s) in which you use the MODEL will have some way to capture the output as well).

You can use a VECTOR[FRML] or VECTOR[EQUATION] to make up all or part of the list. If you do that, you can’t use the >> on those.

Options

- **cv=SYMMETRIC** covariance matrix of residuals

This option provides the (default) covariance matrix for the residuals if you do SIMULATE, ERRORS or IMPULSE; you can also input this matrix directly into those instructions using their own CV options. 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 A Simultaneous Equations Model

Your model will consist of estimated structural equations and identities. You should list the identities last. No variable should be the dependent variable in more than one formula—these restrictions are due to the Gauss-Seidel algorithm used to solve models. 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. The following, for instance, has several (linear) transformations that are used in estimating equations, so to close the model, it’s necessary to add identities (YDIFFID for YDIFF and RSUMID for RSUM) that describe those transformations.
frml(identity) gnpid gnp = cons+invest+govt
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

Examples

This groups three univariate autoregressions to form a model named AR1, and uses
that for the mean model for the GARCH.

equation(constant) jpnej xjpn 1
equation(constant) fraeq xfra 1
equation(constant) suieq xsui 1
group ar1 jpnej fraeq suieq
garch(p=1,q=1,model=ar1,mv=dcc,pmethod=simplex,piter=10)

This defines six equations (four identities and two defining shocks) and groups them
into a model for solution by DSGE.

declare series a1 a2 eps eta
declare real alpha lambda sig_eta sig_eps
  *
frml(identity) f1 = x-(x{1}+a1-lambda*a1{1})
frml(identity) f2 = mu-$
  (((1-lambda)*x{1}+lambda*mu{1}+a2-lambda*a2{1})
frml(identity) f3 = a1-1.0/(lambda+(1-lambda)*alpha)*(eps-eta)
frml(identity) f4 = a2-(1.0/(lambda+(1-lambda)*alpha)*$
  ((1+alpha*(1-lambda))*(eps-(1-lambda)*eta))
frml d1 = eps
frml d2 = eta
  *
group cagan f1 f2 f3 f4 d1 d2

This creates three equations to describe the exogenous variables in a simultaneous
system, groups them into a model called EXOGS, then “adds” it to the existing structural
model for forecasting.

linreg(define=moneyeq) m
  # constant m{1 2}
linreg(define=govteq) govt
  # constant govt{1 2}
frml(identity) mdiffid mdiff = m-m{1}
group exogs moneyeq govteq mdiffid
forecast(model=exogs+prsmall,print,from=1986:1,to=1990:4)
GRParm

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 for KEYLABELING.

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 newsize parameter. For example:

```
grparm(nobold) vlabel *
```

The choices for labeltype 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 footer. Default = 18 points.
- **hlabel**: Font size for the HLABEL. Default = 20 points, bold. Applies to HLABEL option on GBOX, GCONTOUR, GRAPH, SCATTER, and SPGRAPH.
- **vlabel**: Font size for the VLABEL. Default = 20 points, bold. Applies to VLABEL option on GBOX, GCONTOUR, 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.
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 "Choosing Fonts" (page Int–155) of the *Introduction*.

**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 page Int–129 in the *Introduction*.

**height=height of graph in inches [full document window]**

**width=width of graph in inches [full document window]**

These options specify a fixed default size for subsequent graphs. You must use both options together. RATS maintains the size if you print or save the graph. To specify sizes in centimeters, use the `%CM()` function to convert to inches.

**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.

**define="style definition"**

_style definition_ is a string which re-defines one of the line, fill or symbol styles, for instance, **DEFINE="LINE_BW_2=0,.75,3.0"** defines black and white line style 2 as a medium gray solid line with 3 times the standard width.

**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 Style Sheets" on the next page for details.

**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 do:

```
compute fontstyles = %grparm()
```

You can revert back to those settings later by doing:

```
grparm(recall=fontstyles)
```
background=stylenum
This allows you to choose a background for the main box of the graph other than the default choice, which is a solid white background. The stylenum is one of the color fills—you can select from one of the default style choices, or a custom style defined as part of a style sheet (see next page).

shading=stylenum
This allows you to choose a different style for the SHADING (on GRAPH) or VSHADE or HSHADE (on SCATTER and GCONTOUR). The default is a very light solid gray. The stylenum is one of the color fills—you can select from one of the default style choices, or a custom style.

grid=stylenum
This allows you to choose a different style for the GRID or VGRID options (on GRAPH), VGRID or HGRID (on SCATTER and GCONTOUR) or VGRID (on GBOX). The default is a “hairline” (very thin) solid black. The stylenum is one of the color fills—you can select from one of the default style choices, or a custom style.

cmlabels=VECTOR[STRING] with short (1 or 2 character) month labels
smlabels=VECTOR[STRING] with short (about 3) month labels
lmlabels=VECTOR[STRING] with full month names
They allow you to localize the labeling of months in graphs. The default is the standard English names and abbreviations. For instance,

```
grparm(cmlabels=||"E","F","M","A","M","J","J","A","S","O","N","D"||)
grparm(smlabels=||"enero","feb","marzo","abr","mayo","jun","jul","agosto","set","oct","nov","dec"||)
grparm(lmlabels=||"enero","febrero","marzo","abril","mayo","junio","julio","agosto","septiembre","octubre","noviembre","diciembre"||)
```

would use Spanish month names and abbreviations instead. Note that the graphs themselves don’t have the labels built in—instead, they save the raw date information. The substitution is done whenever any graph is displayed, printed or exported after the GRPARM instruction.

Graph Style Sheets
As discussed in Section 3.16 in the Introduction, 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 GRPARM instruction is used to load the definitions saved in a style sheet into memory.
You first use an **OPEN** instruction to open the style sheet, and then use **GRPARM** with the **IMPORT** option to load the styles. For example, if you have a style sheet called **GRAPHSTYLES.TXT**, you might do:

```plaintext
open style graphstyles.txt
grp parm(import=style)
```

See page Int–151 of the *Introduction* 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 **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 **GRPARM HEADER 20** (larger relative to the 18 point default), not **GRPARM HEADER 12**, which would produce smaller headers.

**Examples**

```plaintext
grparm(bold) keylabeling 18
grparm(monthlabels 12
```

The first sets the key labels to 18 point bold; the second increases the month and day labels to 12 point.

```plaintext
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
```

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.

```
grtext(options) "string"
```

Usage

The basic procedure is:

1. Use SPGRAPH to initiate the special graph.
2. Use GRAPH, SCATTER or GCONTOUR 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.

It’s usually a good idea to do the graph first to decide where you want the added text. (GRTEXT should usually be the final step when preparing a graph for publication). The best choice for the ALIGNMENT and DIRECTION options may not be apparent until you see how the graph lays out.

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. You can include line breaks by inserting the characters \ in the string at the points where you want line breaks.

Options

position=upleft/upright/loleft/loright/rightmargin/bottommargin/ leftright/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)

By default, the text will be centered in the graph. You can use these options to control the positioning 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. 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
direction=compass heading in degrees (integer from 0 to 360)

**ALIGNMENT** determines whether the text should be centered or right- or left-justified at the specified position. **VALIGNMENT** determines whether that position is the vertical center, the top, or bottom.

**DIRECTION** allows you to position the text by specifying a direction from the (x,y) point as a compass heading in degrees. For example, DIRECTION=0 (or 360) will center the text at a point just above the (x,y) location; DIRECTION=45 will display left-justified text, starting just above and to the right of the (x,y) location; DIRECTION=270 will display right-justified text directly to the left of the (x,y) point.

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 page Int–155 of the **Introduction** for details.

**bold/[nobold]**
**italics/[noitalics]**

These display the string in bold and/or italic type.

**box/[nobox]**

Puts a box around the text.

**transparent/[notransparent]**

**GRTEXT** strings are normally displayed with an opaque white background, so any lines, patterns or symbols lying “under” the string will be obscured from view. With **TRANSPARENT**, only the text itself will be opaque—all the white space within and between letters will be transparent, allowing any underlying graph elements to show through.
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
grtex(position=upleft) s
spgraph(done)

In the example below, 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)
extremum(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:
grtex(align=left,valign=bottom,entry=maxentry,y=maxval) $
maxstring
grtex(align=left,valign=top,entry=minentry,y=minval) $
minstring
spgraph(done)
Maximum = 10.75
Minimum = -3.69
GSave

GSAVE — Saving Graphs

GSAVE initiates (or turns off) the automatic saving of graphs to disk. It replaces the older ENVIRONMENT GSAVE=... GFORMAT=... operation.

gsave(options)  template

Parameter

template  A template (or a specific filename) to be used for saving graphs. See “Filename Templates” below for details. Execute GSAVE without a parameter to turn off the automatic saving of graphs.

Options

format=[rgf]/portrait/landscape/pdf/lpdf/wmf/png/pict

By default, graphs are saved in RGF (“RATS Graph”) format. Use FORMAT to select a different format. PORTRAIT and LANDSCAPE are PostScript format in portrait and landscape orientations, respectively. PDF and LPDF are Adobe PDF in portrait and landscape modes respectively. Portrait means standard orientation so you don't have to rotate the page. To keep the proportions reasonable it will, by default, use only about the half of a page. Landscape rotates clockwise and, since the height of the page is then the width, uses the full page. PNG is Portable Network Graphics (bitmap).

WMF is Windows Metafile (or "Picture"), while PICT is an older Macintosh format. The availability of some formats is platform-dependent.

[header]/noheader  [footer]/nofooter

Use NOHEADER and/or NOFOOTER to strip header or footer labels from the graph. This is useful if you want to use headers or footers to identify the graphs as displayed on screen in RATS, but don't want those labels included in the version saved to disk, such as when generating graphs for publication where any captions will be added in the word-processing software. If applied to an SPGRAPH or nested SPGRAPHS, only the headers/footers on the outer SPGRAPH would be stripped—any such labels on the individual graphs (or nested SPGRAPHS) are retained.

patterns/[nopatterns]

Use PATTERNS to switch to black and white only when graphs are saved or exported—they will still show as color on the screen. This applies to all graphs saved or otherwise exported, whether with GSAVE, or menu operations.
Filename Templates

The template will usually be a filename string that includes an asterisk (*) symbol. For each graph you generate, RATS will save the graph using a filename constructed by replacing the asterisk with a sequence number. Omit the asterisk to save a single graph using a specific name.

Example

gsave(format=portrait) "ImpulseResponse*.eps"
@varirf(model=canmodel,steps=12,page=byshocks)

This uses the \texttt{@VARIRF} procedure to generate a set of impulse response graphs. The graphs will be saved as PostScript files called “ImpulseResponse1.eps”, “ImpulseResponse2.eps”, and so on.
GSet

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

- **series**: The series of arrays to create or set with this instruction.
- **start end**: The range of entries to set. If you have not set a SMPL, this defaults to the standard workspace.
- **function(T)**: function of the entry number T giving the value for entry T of series. This should evaluate to the type of the elements of the series. There should be at least one blank on either side of the =. 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.

Options

- **startup=expression evaluated at period “start”**
  Use the START option to provide an expression which is computed just once, before the regular formula is computed. This allows you to do any time-consuming calculations that don’t depend upon time. It can be an expression of any type.

- **smpl=SMPL series or formula** (Chapter 25 of the User’s Guide)
  Only entries for which the SMPL series or formula are non-zero will be set.

- **[panel]/nopanel**
  When working with panel data, NOPANEL disables the special panel data treatment of expressions which cross individual boundaries.

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)\ast 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, set equal to the \(X_u\) from the last regression (%eqnxvector(0,t) is \(X_t\)). SSQTEST is the series of values for \((X_u\)\) \(XXS\), \((X_u\)).
HALT — Stopping Execution From Within Compiled Sections

HALT stops execution of the program. You must use HALT 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
```
loops until the user chooses “Quit” from the menu.

See Also . . .

```
END
BREAK
RETURN
```

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" for details.

HISTORY can only work for linear models as it relies upon linearity properties of moving average representations.

The syntax for HISTORY has changed quite a bit over the years. A pre-version 7 HISTORY instruction will look quite different from the preferred setup now, as it would use parameters and supplementary cards rather than options and MODELS. A description of the older syntax is provided below.

```
history(options)
```

Wizard

In Time Series—VAR (Forecast/Analyze) wizard choose "Historical Decomposition" in the Action drop down.

Options

```
model=model name

Of the two ways to input the form of the model to be solved (the other is with supplementary cards), this is the more convenient. MODELS are usually created by GROUP or SYSTEM. It can include FRML's (formulas) only if they represent simple linear equations, as HISTORY requires that the model be fully linear. If the model includes any identities, those should be last in the model. If you use MODEL, omit the “equation” supplementary cards.
```

```
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.
```
base = VECT[SERIES] for base effects [unused]
effects = RECT[SERIES] for cumulated effects of residuals [unused]

If there are $M$ equations and $N$ shocks, BASE creates a VECT[SERIES] with dimension $M$ with the base forecasts of the dependent variables. EFFECTS creates a RECT[SERIES] with dimensions $M \times N$ where column $j$ gives the cumulated effects to the dependent variables of shock $J$. These were added with 10.0 as a simpler alternative to the older RESULTS option.

results = RECTANGULAR[SERIES] for result series

This provides a RECTANGULAR array of SERIES which will be filled with the results—we would recommend using the simpler BASE and EFFECTS options instead. If there are $M$ equations and $N$ shocks, its dimensions will be $(N+1) \times M$. 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.

cv = SYMMETRIC covariance matrix of residuals [from MODEL]
factor = RECTANGULAR decomposition matrix

Use CV when you want orthogonalized innovations using the Cholesky factorization (User’s Guide, Section 7.5). If you are using MODEL and omit this option, HISTORY 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 7.5.2). This option was called DECOMP before version 7. DECOMP is still recognized as a synonym for FACTOR.

print/[noprint]
window = "Title of window"

Use PRINT to display the responses to the output window or file. 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 FACTOR option.

Variables Defined

%FSTART Starting entry of forecasts (INTEGER)
%FEND Ending entry of forecasts (INTEGER)
Technical Description

The historical decomposition is based upon the following partition of the moving average representation

\[ y_{T+j} = \sum_{s=0}^{j-1} \Psi_s u_{T+j-s} + \sum_{s=j}^{\infty} \Psi_s u_{T+s-j} \]

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

```r
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 previous 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.”

**Older Syntax**

```
history(options) equations periods start VCVmatrix
# equation series newstart column (one for each equation if no MODEL)
```

**Parameters**

- **equations**: Number of equations in the system.
- **periods**: number of periods to decompose. Prefer the newer STEPS or FROM and TO options.
- **start**: starting period. Prefer the newer FROM option.
- **VCVmatrix**: covariance matrix for Cholesky factor. Prefer the newer CV option.

**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, if it isn’t an identity, and you didn’t estimate it with a RATS instruction like LINREG, you must associate a residuals series with it using ASSOCIATE.
- **series**: first series in block of equations+1 to hold the decomposition for the dependent variable of this equation. Use numbered series
or a \texttt{VECTOR} of \texttt{SERIES}. The base forecast is put into the first of these, and the effects into the next equations series.

Prefer the newer \texttt{RESULTS} option.

\begin{description}
\item[\texttt{newstart}] (Optional) The first entry for the created series. By default, it is the same as the \texttt{FROM} option value.
\item[\texttt{column}] (Optional) The column (row) of the \texttt{CV} matrix which corresponds to \textit{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). 
\end{description}
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 software packages, 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)
```

**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 should surround it with a { before the IF and a } after the end of the logical 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 }).

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 (and we strongly recommend that you get in the habit of using indenting), 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 } 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
```

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: \[
\text{set patch} = \%\text{if}(\%\text{valid}(xs),xs,xv)
\]

The \textit{wrong} way: \[
\begin{align*}
\text{if } \%\text{valid}(xs) \\
\text{set patch} &= xs \\
\text{else} \\
\text{set patch} &= xv
\end{align*}
\]

This handles a four branch decision. If $\text{LAGS}>1$ and $\text{TREND}$ is non-zero, the first \texttt{LINREG} is executed. If $\text{LAGS}>1$ and $\text{TREND}$ is zero, it's the second. Because the \texttt{\{\}'}s wrap around those two, the second \texttt{ELSE} applies when $\text{LAGS}$ is not bigger than 1. So if $\text{LAGS}<=0$, and $\text{TREND}$ is non-zero, the third \texttt{LINREG} is executed and finally if $\text{LAGS}<=0$ and $\text{TREND}$ is zero, the fourth \texttt{LINREG} is executed. (Note that the check for $\text{LAGS}>1$ is no longer necessary: \texttt{SDIFF\{1 to 0\}} is now interpreted in the natural way of meaning no lags of \texttt{SDIFF}).

\[
\begin{align*}
\text{if } \text{lags}>1 \{ \\
\text{if } \text{trend} \\
\text{linreg(noprint) series startl+lags endl} \\
\hspace{1cm} \# \text{ series\{1\} constant strend sdiff\{1 to lags-1\}} \\
\text{else} \\
\text{linreg(noprint) series startl+lags endl} \\
\hspace{1cm} \# \text{ series\{1\} constant sdiff\{1 to lags-1\}} \\
\} \\
\text{else } \{ \\
\text{if } \text{trend} \\
\text{linreg(noprint) series startl+1 endl} \\
\hspace{1cm} \# \text{ series\{1\} constant strend} \\
\text{else} \\
\text{linreg(noprint) series startl+1 endl} \\
\hspace{1cm} \# \text{ series\{1\} constant}
\}\end{align*}
\]

This is a three branch choice all controlled by the value of \texttt{TRANS}.

\[
\begin{align*}
\text{if } \text{trans} == 1 \\
\text{set transfrm nbeg nend} &= \text{series} \\
\text{else if } \text{trans} == 2 \\
\text{set transfrm nbeg nend} &= \log(\text{series}) \\
\text{else} \\
\text{set transfrm nbeg nend} &= \sqrt{(\text{series})}
\end{align*}
\]

\textbf{See Also . . .}

\emph{UG}, Section 15.1 \hspace{1cm} The RATS Compiler.
Impulse

**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. In a non-linear system, the responses to shocks depend upon the initial point around which you are expanding.

The syntax for **IMPULSE** has changed quite a bit over the years. A pre-version 7 **IMPULSE** instruction will look quite different from the preferred setup now, as it would use parameters and supplementary cards rather than options and **MODELS**. A description of the older syntax is provided at the end.

```
impulse( options )
  # first period shocks (only with INPUT option)
  # list of path series (only with PATHS option)
```

**Wizard**

In the *Time Series—VAR (Forecast/Analyze)* wizard choose "Impulse Responses" in the Action drop down.

**Options**

```
model=model name
  Of the two ways to input the form of the model to be solved (the other is with supplementary cards), this is the more convenient. **MODELS** are usually created by **GROUP** or **SYSTEM**. It can include **FRML**'s (formulas) only if they represent simple linear equations, as **IMPULSE** requires that the model be fully linear. If the model includes any identities, those should be last in the model.

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.

column=component to shock [shock all components]
  By default, **IMPULSE** 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 Cholesky factorization (User’s Guide, Section 7.5). 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 7.5.4). This option was called DECOMP in earlier versions. DECOMP is still recognized as a synonym for FACTOR. FACTOR can have a reduced number of columns (shocks), but must have rows equal to the number of (structural) equations.

results=RECTANGULAR[SERIES] for result series
flatten=(output) RECT for use with %RESPONSES

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 $M \times M$. 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 $M \times 1$. Each series created will be filled from entries 1 to $steps$.

FLATTEN packs the results into a $NVAR \times NSHOCKS \times steps$ matrix in the form needed for an element of the %RESPONSES array (page UG–519).

[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 FACTOR 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 "Technical Details and Choices for Providing Shocks".
Impulse

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.

Technical Details and Choices for Providing Shocks

In the moving average representation,

\[ y_t = \sum_{s=0}^{\infty} \Psi_s u_{t-s} \]

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.

IMPULSE allows the shock to the system to take one of several forms:

1. First period shock which is a unit shock to an orthogonalized innovation of the process. If \( \text{Var}(u) = \Sigma = FF' \), then \( u = Fv \) 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 \( F \). Implement by setting COLUMN to the component you want shocked. \( F \) will be a Cholesky factor by default, so use the FACTOR option if you want a different factor matrix.

2. General first period shocks (\( z \) vector).

Implement using either the SHOCKS option (preferred) or the INPUT option (including an additional supplementary card with the shocks).

3. Paths of shocks to one or more equations.

Implement using either the PATHS option (including an additional supplementary card listing the series of shocks) or the MATRIX option.
Examples

These examples use the six variable VAR from the example program IMPULSES.RPF. The interest rate is the third variable in the system, a fact which is used in several of the examples.

\texttt{impulse(model=canmodel, steps=20, results=impulses)}

computes twenty steps of the impulse responses to all the orthogonalized shocks to the equations in \texttt{CANMODEL}. \texttt{IMPULSES(i,j)} is a series defined from entries 1 to 20 which has the response of the \texttt{i}th dependent variable to a shock in the \texttt{j}th.

\texttt{impulse(model=canmodel, steps=24, col=3, window="Shock to Rate" )}

shocks the third orthogonalized component (the rate shocks) and puts 24 step responses out to a window.

\texttt{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 \texttt{f1}, \texttt{f2}, \texttt{r1}, \texttt{r2}, \texttt{n1} and \texttt{n2}.

\texttt{impulse(model=canmodel, steps=24, shock=%unitv(6,3), noprint, results=torate)}

computes the response to a unit shock in the interest rate alone (not an orthogonalized component). So the impact on the interest rate with be 1.0 initially, while all other variables will have an impact shock of 0. \texttt{TORATE(i, 1)} will be the response of variable \texttt{i} to the shock. Note that you must be very careful with the scaling of shocks like this. A unit shock to a variable in log's means an impact the same as multiplying the data by 2.718. Unit shocks in orthogonalized components as in the previous examples all adjust automatically to the scale of the variables.

\texttt{set shockr 1 3 = 1.00}
\texttt{set shockr 4 10 = 0.00}
\texttt{impulse(model=canmodel, paths, steps=10, noprint, results=persist)}

# * * shockr * * *

gives shocks of size 1.00 to the interest rate in each of the first three (of ten) periods.
Impulse

Older Syntax

```
impulse(options)  equations  steps  shockto  VCVmatrix
# 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)
```

Parameters

- **equations**: Number of equations in the system.
- **steps**: number of steps to compute. Prefer the newer `STEPS` option.
- **shockto**: variable/component to shock. Prefer the newer `COLUMN` option.
- **VCVmatrix**: covariance matrix for Cholesky factor. Prefer the newer `CV` option.

Equation 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. 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.
- **column**: (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.

Examples with Supplementary Cards

```
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, noprin) 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 saves the responses in RESP1, RESP2 and RESP3.

**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,

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:

list ieqn = 1 2 3 4
impulse(steps=20, column=1, cv=v) 4
cards ieqn resp(ieqn) 1 ieqn

**See Also ...**

*UG*, Section 7.5 Orthogonalization.
*UG*, Section 7.6 Using IMPULSE and ERRORS.
ERRORS Decomposition of Variance.
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" (optional) 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 it 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. Once the operation has taken longer than ten seconds, an estimate of the time to completion is added to the box. This assumes that the time required from lower bound to upper bound is roughly linear.

```
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. 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.

```plaintext
infobox(action=define, progress, lower=1, upper=ndraws) "$Monte Carlo Simulation"
do draw=1, ndraws
    *** Monte Carlo simulation instructions here:
    ...
    infobox(current=draw)
end do draw
infobox(action=remove)
```

The messagestring can be created to show information which might be helpful in deciding to stop the calculations. The code below will display a running estimate of the acceptance rate of a Metropolis-Hastings Markov Chain. If the rate is lower than you would like, you can cancel the process and adjust the way the draws are done.

```plaintext
compute accept=0
infobox(action=define, progress, lower=1, upper=ndraws) "Metropolis"
do draw=1, ndraws
    ... if alpha>1.0.or.%uniform(0.0,1.0)<alpha {
        compute accept=accept+1
    }
    infobox(current=draw) $
        "Acceptance Rate "+%strval(100.0*accept/draw, "###.##")
end do draw
infobox(action=remove)
```

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.

\[
\text{initial( options ) equation start end}
\]

Parameters

- **equation**: Equation whose initial estimates are to be computed.
- **start** and **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 \textit{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 \textit{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, and Reinsel (2008), 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 handle many types of data files. The FIXED instruction is useful for setting arrays of fixed values in PROCEDUREs and FUNCTIONs.

```
input( option )  arrays, variables, array elements
```

Parameters

arrays,...

These are the objects for which data is to be read. You can use any combination of variables. You can use arrays of arrays, but 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 array type)**
- are written and read as row vectors.

**General two-dimensional arrays (RECTANGULAR array type)**
- are written and read by rows (the natural fashion), even though internally they are stored by columns.

**Specialized two-dimensional arrays (SYMMETRIC and PACKED array types)**
- 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:

\[
\begin{bmatrix}
1.0 \\
2.0 & 3.0 \\
4.0 & 5.0 & 6.0
\end{bmatrix}
\]

\[
Y = 7.3
\]

\[
c = \begin{bmatrix} 0.0 + 1.0i \\ 1.0 + 0.0i \end{bmatrix}
\]

\[
I = 5
\]

Notes
You can also use **COMPUTE** for the initialization above:

```
compute [symmetric] v=||1.0|2.0,3.0|4.0,5.0,6.0||
compute y=7.3
compute [vector[complex]] c2=||%cmplx(0.0,1.0),%cmplx(1.0,0.0)||
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

dec vector v
input(varying) v
  1  5 10 25 50 100 250 500 1000

VARYING is useful if you would rather not 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 — 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.

\[ \text{inquire( options ) value1<<p1 \hspace{1em} value2<<p2} \]

# list of variables in regression format (only with the REGRESSORLIST option)

**Parameters**

\[ \text{value1 \hspace{1em} value2} \]

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.

\[ \text{p1 \hspace{1em} p2} \]

(Optional) Use these 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.

**Options (Mutually Exclusive)**

\[ \text{series=} \hspace{1em} \text{series name or number} \]

returns the defined range of the indicated series.

\[ \text{regressorlist/ [noregressorlist]} \]

\[ \text{equation=} \hspace{1em} \text{EQUATION supplying list of variables} \]

\[ \text{model=} \hspace{1em} \text{MODEL supplying list of variables} \]

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 (both dependent and explanatory) in the equation you supply. MODEL determines the range based upon all variables in all (linear) equations in the MODEL.

\[ \text{valid=} \hspace{1em} \text{dummy series to create} \]

Used in conjunction with SERIES, LASTREG, REGRESSORLIST, EQUATION or MODEL, 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.
smpl/[nosmpl]
    value1 and value2 are set to the starting and ending entries of the current SMPL.

seasonal/[noseasonal]
    This returns as value1 the current calendar seasonal: for instance, 4 for a quarterly CALENDAR, 12 for a monthly CALENDAR.

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.

matrix= matrix name
    This returns the dimensions of the indicated matrix. value1 and value2 are set to the number of rows and columns, respectively. However, it's simpler to use the %ROWS(matrix) and %COLS(matrix) functions instead.

lastreg/[nolastreg]
    value1 and value2 are set to the starting and ending entries of the last regression. However, it's simpler to use the %REGSTART() and %REGEND() functions.

Examples

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 gdp82
@test gdp82 1947:1 2017:4
@test gdp82 1955:1 *

In the first, STARTL and ENDL will be the start and end of the series GDP82. In the second, they will be 1947:1 and 2017:4 and in the third, 1955:1 and the end of GDP82.

inquire(equation=vareqn) rstart rend

This returns in RSTART and REND the range allowed by the variable in VAREQN.
Inquire

instruments ablogc dlogp dlogpn dlogy treduced
inquire(valid=absample,reglist)
# dlogc{0 1} dlogp dlogpn dlogy
mcov(instruments, lwform=||-1.0,2.0,-1.0||, smpl=absample)

This determines the set of entries at which all the series and lags listed in the supplementary card for the INQUIRE are defined, returning that as the dummy series ABSAMPLE.

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 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.
%SIZE(matrix) Returns total number of elements 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 $2 \times K$ 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.
%MODELDEP_VARS(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.

\[ \text{instruments(options) } \text{ exogenous variables in regression format} \]

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
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 variables list.

Options
\begin{align*}
\text{drop/} & \text{[nodrop]} \\
\text{add/} & \text{[noadd]}
\end{align*}

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.

\begin{align*}
\text{print/} & \text{[noprint]}
\end{align*}

Use PRINT to list the current set of instruments.

Example
\begin{align*}
\text{instruments constant trend govtwage taxes govtexp } \$ \\
\text{capital\{1\} production\{1\} profit\{1\}}
\end{align*}

sets the instruments list for Klein’s Model I.
Notes

The instruction \texttt{NLSYSTEM} has a special option \texttt{MASK} which allows a different set of instruments to be used for each equation in the system. You provide a \texttt{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:

\begin{verbatim}
  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) / frm1 frm2
\end{verbatim}

would use \texttt{CONSTANT}, lags of \texttt{CSZ} and lags of \texttt{PCS} for the first formula and \texttt{CONSTANT}, lags of \texttt{CSZ} and lags of \texttt{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 \texttt{%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:

\begin{verbatim}
  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})
\end{verbatim}

and then use $Y1 \ Y2 \ Y3 \ Y4$ on the instrument list.

Variables Defined

\texttt{%INSTR} \hspace{1cm} number of variables on the list (\texttt{INTEGER})

See Also . . .

\texttt{SWEEP} \hspace{1cm} Projects a set of target variables on a set of other variables.
\texttt{UG}, Section 2.3 \hspace{1cm} Instrumental Variables and Two-Stage Least Squares.
\texttt{UG}, Section 4.9 \hspace{1cm} Method of Moments Estimators (Univariate).
\texttt{UG}, Section 4.10 \hspace{1cm} Non-Linear Systems Estimation.
\texttt{%instlist()} \hspace{1cm} Function returning current instruments as a regressor list
\texttt{%insttable()} \hspace{1cm} Function returning current instruments as a table
\texttt{%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 7.14) 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 )
```

Basic Options

print/[noprint]
ftests/[noftests]

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” on page RM–273 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 current 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. 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 ("SMPL Option" on page RM–546)

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=(output) 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.

discount=discount value [not used]

This allows for multiplicative (rather than additive) changes to the variance of the states. The update for the covariance matrix takes the form:

$$\Sigma_{t|t-1} = A_{t} \Sigma_{t-1|t-1} A_{t}' \frac{1}{\text{discount}}$$
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 \( v \) option 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**

\%

- **NREG** number of regressors in the first equation (INTEGER)
- **XX** the \( X'X^{-1} \) matrix (SYMMETRIC)

**Examples**

```plaintext
system(model=yrmp)
variables gdp cpr ipd m1
lags 1 to 4
det constant
end(system)
estimate 1948:1 2011:4
do time=2012:1,2017:4
    kalmann(print=(time==2017:4))
end do
```

**ESTIMATE** computes the regressions through 2011:4 and prints them with the \( F \)-tests. The loop then executes **KALMAN** twenty-four times (quarterly data) so that the last produces estimates using data through 2017:4. This only prints the estimates for this final period because the argument for **PRINT** will be zero until **TIME** is 2017:4.
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).

Older Syntax (with parameters)

\[ \text{kalman( options ) residuals coeffs printflag testflag} \]

Parameters

- **residuals, coeffs** first series in a block of series for the residuals and coefficients, respectively (use VECTORS of SERIES or numbered series). The new RESIDUALS and COEFFS options are generally more convenient for saving these values.

- **printflag, testflag** KALMAN prints output and F-tests (respectively) when these are equal to 1 (usually set with logical expressions). These override the NOPRINT and NOFTESTS options.

See Also ...

- **UG, Section 7.14** 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 7.14 of the *User’s Guide* for more on the use of **KFSET** and related instructions.

```
**kfset**( options )
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>list of covariance matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># innovation variances</td>
</tr>
<tr>
<td></td>
<td>(if VARIANCE=KNOWN, CONSTANT and no VECTOR)</td>
</tr>
<tr>
<td></td>
<td># list of variance series</td>
</tr>
<tr>
<td></td>
<td>(if VARIANCE=KNOWN and NOCONSTANT)</td>
</tr>
</tbody>
</table>

**Parameters**

| 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, in which case 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 $\mathbf{\beta}_t$ is the vector of “states” at time $t$.
- The measurement equation is $y_t = \mathbf{X}_t \mathbf{\beta}_t + u_t$, where the variance of $u_t$ is $n_t$.
- The state vector follows the process: $\mathbf{\beta}_t = \mathbf{\beta}_{t-1} + \mathbf{v}_t$, with $\operatorname{Var}(\mathbf{v}_t) = \mathbf{M}_t$.
- $u_t$ and $\mathbf{v}_t$ are independent.

Let $\mathbf{\beta}_{t|t}$ be the estimates of $\mathbf{\beta}_t$ using information through $t$, and let the covariance matrix of $\mathbf{\beta}_{t|t}$ 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

\texttt{\textbf{variance=\{concentrated\}/known}}

\texttt{\textbf{constant/\{noconstant\}}}  \textit{(with VARIANCE=KNOWN only)}

\texttt{\textbf{v=VECTOR of variances}}  \textit{(with VARIANCE=KNOWN, CONSTANT only)}

Use these options for setting the variances of the measurement equation errors \((n_i)\). Note that constant variances are provided in a \texttt{VECTOR}. This is done because a \texttt{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 \texttt{||variance||}. (The \texttt{VARIANCE} option replaces the \texttt{SCALE} option used before version 7).

\texttt{\textbf{likelihood=2 x Equations RECTANGULAR array}}

This allows computation (by \texttt{KALMAN} instructions) of a pseudo-likelihood function for Kalman filter estimates stored in the array. See Doan, Litterman and Sims (1984), page 10 and page UG–267 of the \textit{User’s Guide} for more information.

\texttt{\textbf{names=\"base for array names\"}}

This is an alternative to using the \textit{list of matrices} parameter. When you use \texttt{NAMES}, \texttt{KFSET} forms names formed by appending the numbers 1,2,3,etc. to the \textit{base} label you supply. That is, if you use \texttt{NAMES=\"COVMAT\"}, \texttt{KFSET} will create \texttt{COVMAT1}, \texttt{COVMAT2}, etc.; as many as are needed. \texttt{NAMES} is largely obsolete, because you now use a \texttt{VECTOR} of \texttt{SYMMETRIC} arrays for the list of matrices.

Description

\texttt{KFSET} uses the \textit{list of matrices} in two ways:

- If you execute \texttt{KALMAN} without an \texttt{ESTIMATE}, \textit{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 \texttt{KALMAN}.
- If you \textit{do} execute an \texttt{ESTIMATE}, \texttt{RATS} fills the \textit{list of matrices} with the estimated covariance matrices. For this use, you do not need to dimension the arrays because \texttt{ESTIMATE} does it for you.

If you are using \texttt{KALMAN} without \texttt{ESTIMATE}, the initial coefficients also need to be set. This is usually done with the \texttt{ASSOCIATE} instruction.

If you want time-varying parameters \((M_i\text{, non-zero})\), you need to use \texttt{TVARYING} as well as \texttt{KFSET}.

Notes

If you want to write a single set of code which can handle \texttt{VAR}s of various sizes, we recommend that you use a \texttt{VECTOR[SYMMETRIC]} in place of a list. For instance,

\begin{verbatim}
  dec vector[symmetric]  kfs(neqn)
  kfset kfs

  makes KFS(1), KFS(2),..., KFS(NEQN) the list of matrices.
\end{verbatim}
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 `VARIANCE=CONCENTRATED` (default). RATS computes new estimates for the variances whenever you request output from `ESTIMATE` or `KALMAN`. *If you are using time-varying parameters (\(M_i\) 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 `KFSET` since all equations have the same \((X'X)^{-1}\) matrix. See the example below.

**Constant over time and known**

Use `VARIANCE=KNOWN` with `CONSTANT`. You can supply the values in one of two ways:

- with the option `V=VECTOR` of variances
- with a supplementary card

**Changing over time**

Use `VARIANCE=KNOWN` and `NOCONSTANT`, and include a supplementary card which lists series whose values are the desired variances.

**Examples**

```plaintext
system(model=rmpy)
variables cpr ml ppi ip
lags 1 to 12
kfset xxsy
end(system)
estimate 1948:1 2010:12
ESTIMATE dimensions and sets XXSYS
```

uses `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 `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 `XXMAT`, `TVMAT` and the coefficients of `KFEQ` before you can use `KALMAN`. 
Labels

LABELS — Setting Output Labels for Series

LABELS attaches an output label to a series. RATS uses these output labels whenever it needs to display the name of a series. They are also used when you read data from a file that includes series names, including RATS, portable, and spreadsheet formats.

```
labels list of series  (usually by number)
# "labels" and label expressions
```

Parameters

list of series  List of series to be given labels.

Supplementary Card

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 Time Series—VAR (Setup/Estimate) wizard 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 six-variable VAR with four 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 — Limited Dependent Variable Estimation

The LDV instruction implements limited dependent variable estimation techniques, for models with censored or truncated data.

```
ldv(options) depvar start end residuals
# list of explanatory variables in regression format
```

Wizard

The Statistics—Limited/Discrete Dependent Variables wizard 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 (“SMPL Option” on page RM–546)
unravel/[nounravel] (User’s Guide, Section 2.10)
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 4 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 λ 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 LAGS), as well as clustered standard errors. See Sections 2.2, 2.3, and 2.4 of the User’s Guide and “Long-Run Variance/Robust Covariance Calculations” on page RM–540.

Note, however, that the estimates themselves may be inconsistent if the distributional assumptions are incorrect.

weight=series of entry weights (“WEIGHT option” on page RM–549)
Use this option if you want to weight the observations unequally.
Description

The technical information on these models is provided in Section 12.3 of the User’s Guide. All the models are based upon the standard

\[ y_i = X_i \beta + u_i ; u_i \sim N(0, \sigma^2) i.i.d. \]

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 one 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

\[
\log L = \begin{cases} 
\log(1 - \Phi(L_i h - X_i \gamma)) & \text{if unbounded above} \\
\log(\Phi(U_i h - X_i \gamma)) & \text{if unbounded below} \\
\log(\Phi(U_i h - X_i \gamma) - \Phi(L_i h - X_i \gamma)) & \text{otherwise}
\end{cases}
\]

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.

Hypothesis Testing

You can apply the hypothesis testing instructions (EXCLUDE, TEST, RESTRICT and MRESTRICT) to estimates from LDV and you can use the Statistics—Regression Tests wizard to set those up. They compute the “Wald” test based upon the quadratic approximation to the likelihood function. RATS uses the second set of formulas in Section 3.3 of the User’s Guide to compute the statistics. Note that you cannot use the CREATE or REPLACE options on RESTRICT and MRESTRICT.
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 \textit{AFFAIR}.

Variables Defined by LDV

\%BETA \hspace{1cm} \text{VECTOR of coefficients}
\%SEESQ \hspace{1cm} \text{value of } \sigma^2 \text{ (use SQRT(\%SEESQ) to get value of SIGMA) (REAL)}
\%XX \hspace{1cm} \text{covariance matrix of coefficients (SYMMETRIC)}
\%STDERRS \hspace{1cm} \text{VECTOR of coefficient standard errors}
\%TSTATS \hspace{1cm} \text{VECTOR of } t\text{-statistics of the coefficients}
\%NFREE \hspace{1cm} \text{number of free parameters (INTEGER)}
\%NOBS \hspace{1cm} \text{number of observations (INTEGER)}
\%NREG \hspace{1cm} \text{number of regressors (INTEGER)}
\%NFREE \hspace{1cm} \text{number of free parameters (INTEGER)}
\%LOGL \hspace{1cm} \text{log likelihood (REAL)}
\%CVCRIT \hspace{1cm} \text{value of the convergence criterion (REAL)}
\%ITERS \hspace{1cm} \text{number of iterations completed (INTEGER)}
LINREG — Linear Regressions

LINREG computes a single linear regression using least squares, weighted least squares, or instrumental variables. We discuss many of the options in Chapter 1 of the Introduction and Chapter 2 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 Statistics—Linear Regressions wizard, 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 (page Int–77 of the Introduction).

- **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.

- **smpl=SMPL series or formula** (“SMPL Option” on page RM–546)
  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=standard SPREAD option ("SPREAD Option" on page RM–547)
Use SPREAD for weighted least squares. The residual variances are assumed to be proportional to the indicated series.

weight=series of entry weights ("WEIGHT option" on page RM–549)
Use this option if you want to weight the observations unequally.

shuffle=SERIES[INTEGER] for remapping ("SHUFFLE Option" on page RM–548)
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 (Introduction, Section 1.5.4)
frml=formula to define
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 2.10). RATS does not print the intermediate regression (in terms of encoded variables).

cmom/[nocmom]
Use the CMOM option together 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 (Section 1.2 in Additional Topics PDF).

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 1.4 in the Additional Topics PDF for details.

create/[nocreate]
If you use CREATE, LINREG does not estimate the regression. Instead, it generates the standard regression output and statistics using information that you provide. See “Options Used with CREATE” for details.
**Linreg**

**dfc=Degrees of Freedom Correction (Additional Topics, Section 1.4)**

Corrects for degrees of freedom lost in the dependent variable in processing prior to the regression.

**robusterrors/[norobusterrors]**

**lags=correlated lags [0]**

**lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/**

**panel/white**

**damp=value of \( \gamma \) for lwindow=damped [0.0] (overrides LWINDOW setting)**

**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 ROBUSTERERS) or serial correlation (with LAGS), or clustering based upon some other set of categories. Sections 2.2, 2.3, and 2.4 of the User’s Guide and “Long-Run Variance/Robust Covariance Calculations” on page RM–540. Note especially the possible computational problems that may arise when using LAGS.

**Options for Two-Stage Least Squares (GMM)**

**instruments/[noinstruments]**

Use the INSTRUMENTS option to do two-stage least squares. You must set your instruments list first using the instruction INSTRUMENTS.

**wmatrix=weighting matrix \([Z'Z]^{-1}\)**

**iwmatrix=inverse weighting matrix [not used]**

**optimalweights/[nooptimalweights]**

**update=[none]/once/continuous**

These control the updating of the weighting matrix. You can directly input a weight matrix with the WMATRIX or IWMATRIX 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.

**[zudep]/nozudep**

**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 lag window form [not used]**

**cluster=SERIES with category values for clustered calculation**

Use these to select the type of correlation assume for the \( Z'u \) process. See “Long-Run Variance/Robust Covariance Calculations” on page RM–540 for more information.
iterations = iteration limit \([1]\)
cvcrit = convergence limit \([0.0001]\)

trace/\[\text{notrace}\]
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 \([\text{zeros}]\)
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/\[\text{nocenter}\]
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. For more information, see “ZUMEAN and CENTER options” on page RM–541.

jrobust = statistic/\[\text{distribution}\]
You can use this option to adjust the \(J\)-statistic specification test when the weighting matrix used is not the optimal one. See Section 4.9 in the User’s Guide for more information.

Examples

linreg(define=foodeq) foodcons / resids
# constant dispinc trend

regresses FOODCONS on DISPINC and TREND, saves the residuals as RESIDS, and creates equation FOODEQ.

linreg(robusterrors) gdp 1955:1 2009:4
# constant m1\{-4 to 8\}

regresses GDP on 4 leads to 8 lags of M1 and corrects the covariance matrix for possible heteroscedasticity.

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

%BETA Coefficient VECTOR

%DURBIN Durbin-Watson statistic (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)

%RESIDS Series containing the residuals (SERIES)

%RSS Residual sum of squares (REAL)

%SEESQ Standard error of estimate squared (REAL)

%STDERRS VECTOR of coefficient standard errors

%TSTATS VECTOR containing the t-stats for the coefficients

%RHO First lag correlation coefficient (REAL)

%VARIANCE Variance of dependent variable (REAL)

%XX Covariance matrix of coefficients, or \((X'X)^{-1}\) (SYMmetric)

Least Squares Estimators Only

%LOGL Normal log likelihood (REAL)

%R0BAR SQ Adjusted \(R^2\) (REAL)

%RSQUARED Centered \(R^2\) (REAL)

%RSQ No. of observations times raw \(R^2\) (REAL)

%TRSQUARED No. of observations times centered \(R^2\) (REAL)

Instrumental Variables Estimators Only

%JSIGNIF Significance of %JSTAT (REAL)

%JSTAT Test statistic for overidentification for instrumental variables (REAL)

%NDFQ Degrees of freedom for %JSTAT (INTEGER)

%UZWZU \(u'ZW'Z'u\) for instrumental variables (REAL)

%WMATRIX Final weight matrix for GMM (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 those formulas 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**

```plaintext
m cov(matrix=b,lastreg) / f
m cov(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,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.

In older RATS programs, these were often used extensively for handling forecasting and other analysis instructions (FORECAST, IMPULSE, ERRORS, etc.) for Vector Autoregressions, as they could be defined as a set of equations. A more modern handling of these uses the MODEL options which group all the equations into a single object.

```
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(cv=vcv, steps=48, col=1)
# 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(cv=vcv, steps=48, col=1)
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,

```plaintext
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
```

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,

```plaintext
declare vector[series] forecast(8)
list ieqn = 1 to 8
forecast(input,print,steps=24,from=2010:1) 8
cards ieqn forecast(ieqn)
# .05 .03 0.0 0.0 0.0 0.0 0.0 0.0
```

which replaces

```plaintext
declare vector[series] forecast(8)
list ieqn = 1 to 8
forecast(input,print,steps=24,from=2010:1) 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
```

**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, GBOX, SCATTER, SUR**, and **SMODIFY**.

You can also use **CARDS** to supply the supplementary card for an **ENTER** instruction (in a **PROCEDURE**, for example). Use the **SEQUENCE** option on **ENTER** to indicate the value for the **LIST** index that should be used in evaluating the **CARDS** instruction.
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**

- **datatype**
  - Variable type that you want to assign to these variables. Local variables can be any of the data types supported by RATS. See Section 1.4 of the User's Guide.

- **list of names**
  - The list of variables that will have datatype. Separate the names with blanks. Symbolic names must be unique—if you attempt to assign a new type to a name already in use as a local variable, 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. The dimensions can use **PROCEDURE** or **FUNCTION** parameters or options.

**Description**

Any variable (other than a **PROCEDURE** parameter or option 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`.

**Example**

These are the first actual program lines for the `@DFUNIT` procedure. With the exception of `DFUNIT` itself and `%%AUTOP`, all the variables defined by the `TYPE`, `OPTION` and `LOCAL` instructions are part of the `DFUNIT`’s “namespace”, that is, they are recognized only here and can be used in other procedures or also as global variables. `%%AUTOP` is intentionally defined using `DECLARE` so it will be visible outside the procedure.

```plaintext
procedure dfunit series start end
  type series    series
  type integer   start end
  *
  option choice det 2 none constant trend
  option switch ttest 1
  option switch print 1
  option string title

  option integer lags
  option integer maxlags
  option choice method 1 input aic bic hq ttest gtos
  option real signif .10

  option switch intercept 1
  option switch trend 0
  *
  local integer startl endl nobs tag
  local series  s sdiff strend
  local real   teststat fiddle
  local string descrip lmethod
  local vect   critvalues
  local integer dettrend detconstant
  local string ltitle
  local integer maxlag znobs bestlag lag tag
  local real   ic icmin icmult
  local report dfreport
  local vect[int] detlist
  declare integer %%autop
```
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 Data/Graphics—Transformations wizard.

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.

```
log  gnp  /  loggnp
log  forecast
```

Here is how you would do these same operations using SET:

```
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.

```
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 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. A NEXT will jump back up to the top of the loop from wherever it is executed.

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 . . .

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAK</td>
<td>Breaks control out of a loop.</td>
</tr>
<tr>
<td>NEXT</td>
<td>Skips to the top of a loop.</td>
</tr>
<tr>
<td>DO</td>
<td>Looping over an index.</td>
</tr>
<tr>
<td>DOFOR</td>
<td>Looping over a list of items.</td>
</tr>
<tr>
<td>WHILE</td>
<td>Conditional looping.</td>
</tr>
<tr>
<td>UNTIL</td>
<td>Conditional looping.</td>
</tr>
<tr>
<td>UG, Section 15.1</td>
<td>The RATS Compiler</td>
</tr>
</tbody>
</table>
LQPROG — Linear and Quadratic Programming

LQPROG solves linear and quadratic programming problems.

\[ \text{lqprog( options ) } \quad x \]

The linear programs take the form:

minimize \[ c'x \]
subject to \[ A_x = b_e, \ A_x \leq b_l, \ A_x \geq b_g \]
\[ x_i \geq 0 \text{ for all } i=1,...,nvar \]
where, by design, \[ b_e \geq 0, b_l \geq 0, b_g \geq 0 \]

\( nvar \) 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 \( Q \).

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.

\( c, A, b, Q \)

These are for an older form of the instruction. They are the 2nd through 5th parameters in order and provide the same information as the newer options of the same names. We strongly recommend using the option form.

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, \( \text{LQPROG} \) 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 \( \leq \) constraints, and finally any \( \geq \) constraints.
**equalities=number of equality constraints [0]**

**ge=number of ≥ constraints [0]**

By default, all constraints are assumed to be of the form $Ax \leq b$. You can use the **EQUALITIES** and **GE** options if you wish to include equality constraints and constraints of the form $Ax \geq b$. Any equality constraints should be listed first in the $A$ and $b$ arrays, followed by any $Ax \leq b$ constraints. List any $Ax \geq b$ 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 **LQPROG** determines whether or not the process has converged to a solution. **LQPROG** 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**, **LQPROG** 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, **LQPROG** 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 13.2 in the **User’s Guide**.

Note that, while **LQPROG** 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

%FUNCVAL Final value of the function (REAL)
%LAGRANGE VECTOR of Lagrange multipliers on the constraints

Linear Programming Example

minimize \(-2x_1 - 4x_2 - x_3 - x_4\)
subject to
\[\begin{align*}
    x_1 + 3x_2 + x_4 &= 4 \\
    2x_1 + x_2 &\leq 3 \\
    x_2 + 4x_3 + x_4 &\leq 3 \\
    x_i &\geq 0
\end{align*}\]

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:

\[
\begin{align*}
\text{minimize} & \quad x_1^2 + x_2^2 + x_3^2 + x_1x_3 - 10x_1 - 4x_3 \\
\text{subject to} & \quad 2x_1 + x_2 + x_3 = 2 \\
& \quad x_1 + 2x_2 + 3x_3 = 5 \\
& \quad 2x_1 + 2x_2 + x_3 \leq 6 \\
& \quad x_i \geq 0
\end{align*}
\]

\[\text{declare rectangular amat(3,3)}\]
\[\text{declare symmetric qmat(3,3)}\]
\[\text{declare vector cvec(3) bvec(3)}\]
\[\text{compute amat = } \begin{bmatrix} 2, 1, 1; 1, 2, 3; 2, 2, 1 \end{bmatrix}\]
\[\text{compute bvec = } \begin{bmatrix} 2, 5, 6 \end{bmatrix}\]
\[\text{compute cvec = } \begin{bmatrix} -10, 0, -4 \end{bmatrix}\]
\[\text{compute qmat = } \begin{bmatrix} 2 & 0 & 2 \\ 0 & 0.01 & 0.1 \\ 2 & 0 & 1 \end{bmatrix}\]

\[\text{lqprog(equalities=2,c=cvec,a=amat,b=bvec,q=qmat) results}\]

which produces the output:

\[\text{QPROG converges at 3 iterations}\]
\[\text{minimum = -5.480000}\]
\[\text{solution x = }\begin{bmatrix} 0.2000 \\ 0.0000 \\ 1.6000 \end{bmatrix}\]

The example below demonstrates a simple portfolio optimization routine (see example \text{PORTFOLIO.RPF} 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.

\[\text{compute N=3}\]
\[\text{compute [vect] ExpRet = } \begin{bmatrix} 0.15, 0.20, 0.08 \end{bmatrix}\]
\[\text{compute [symm] CovMat = } \begin{bmatrix} 0.20 & 0.05 & 0.01 \\ 0.05 & 0.30 & -0.01 \\ 0.01 & -0.01 & 0.1 \end{bmatrix}\]

\[\text{* Compute the minimum variance portfolio, and its expected return}\]
\[\text{compute units=} \% \text{fill(1,N,1.0)}\]
\[\text{lqprog(q= CovMat, a=units, b=1.0, equalities=1) x}\]
\[\text{compute minr=} \% \text{dot(x,ExpRet)}\]
\[\text{compute maxr=} \% \text{maxvalue(ExpRet)}\]
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 %NOBS 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**

- **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** (“SMPL Option” on page RM–546)

  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.

- **panel/[nopanel]**

  You can use this option to split one or more panel data series into a \( T \times (NK) \) array, where \( T \) is the number of time periods per individual, \( N \) is the number of individuals, and \( K \) is the number of series.
**Make**

`mult=` series by which to multiply observations  
`spread=` series by which to divide observations (square root)  
`scaле=` value by which to scale matrix

These three options are mutually exclusive.

- With `MULT`, `MAKE` multiplies each observation in the matrix by the corresponding entry from the series.
- With `SPREAD`, it divides each observation in the matrix by the square root of the entry of the spread series. `SPREAD` has the same basic function as it does for `LINREG`.
- With `SCALE`, it multiplies all entries by the single value.

**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`.

**Examples**

```plaintext
make(lastreg) x %regstart() %regend()  
compute m=%identity(%nobs)-x*inv(tr(x)*x)*tr(x)
```

This takes the matrix of explanatory variables from the previous regression and creates the matrix `X` from them. That is used to compute the residual projection matrix:

\[
I - X(X'X)^{-1}X'
\]

```plaintext
make(panel) ut  
# %resids  
compute tdim=%rows(ut),ndim=%cols(ut)
```

This creates a `T × N` matrix `UT` from the residuals series `%RESIDS` where `T` is the number of observations per individual in the panel set and `N` is the number of individuals.

```plaintext
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)
```

This computes the residual vector. It does a least squares regression using matrix instructions. We include the `SMPL` option because each `MAKE`, on its own, might determine a different default range.
"Unmaking" a Matrix

Use SET or a sequence of SET instructions to set series equal to the rows (or columns) of matrices. For example,

```
set x2 1 100 = a(t,1)
```

sets $x2$ equal to the first 100 entries of column 1 of $A$. Remember, however, that the $T$ subscript runs over the start to end range on SET, which is not necessarily 1,...,dimension. The following (trivial) example shows how to fix the subscript in the SET to keep the output series aligned with the input. ($xx$ will be the same as $x$).

```
make r 1948:1 2017:12
# constant x
set xx 1948:1 2017:12 = r(t-(1948:1-1),2)
```

If the size of the matrix can change from one use to another, you should “unmake” into a VECTOR of SERIES. For instance, the following creates a vector of NCOLS series from the columns of $R$.

```
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, MAKE drops that entry from the constructed array. If you are using MAKE to create several arrays which must have identical sets of entries, use a common SMPL to deal with missing values.

```
smpl(reglist)
# constant v1{0 to 3} includes all variables
make x
# constant v1{1 to 3}
make y
# v1
```

Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations</td>
<td>INTEGER</td>
</tr>
<tr>
<td>%NVAR</td>
<td>Number of variables</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>
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.

```
maximize(options) frml start end funcval
```

**Description**

**MAXIMIZE** finds (or attempts to find) the $\beta$ which solves:

$$\max_{\beta} G(\beta) = \sum_{i=1}^{T} f(y_i, X_i, \beta)$$

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** and **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

\textit{\texttt{parmset=PARMSET} to use [default internal]}

This option selects the parameter set to be estimated (User’s Guide, page UG–129). RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.

\textit{\texttt{method=bhhh/[bfgs]/simplex/genetic/annealing/ga/evaluate}}

Chooses the estimation method. See Chapter 4 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 and are the only ones that can compute standard errors. The others are derivative-free, make less stringent assumptions and are generally used as a PMETHOD choice to refine initial guess values rather than as the final estimation method.

\textit{\texttt{METHOD=EVALUATE}} simply evaluates the function at the current settings of the parameters and produces no output other than the variables like \%LOGL and \%FUNCVAL.

\textit{\texttt{iterations=iteration limit [100]}}

\textit{\texttt{subiterations=subiteration limit [30]}}

\textit{\texttt{cvcrit=convergence limit [.00001]}}

\textit{\texttt{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 \texttt{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.

\textit{\texttt{pmethod=bhhh/bfgs/[simplex]/genetic/annealing/ga}}

\textit{\texttt{piters=number of PMETHOD iterations to perform [none]}}

Use \texttt{PMETHOD} and \texttt{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 \texttt{PMETHOD=SIMPLEX}, \texttt{PITERS=10}, and \texttt{METHOD=BFGS}.

\textit{\texttt{startup=FRML evaluated at period "start"}}

\textit{\texttt{onlyif=expression tested before calculating start option [not used]}}

You can use the \texttt{START} option to provide an expression which is computed once per function evaluation, before any of the regular formulas are computed. This allows you to do any time-consuming calculations that depend upon the parameters, but not upon time. It can be an expression of any type. \texttt{ONLYIF} calculates the expression provided; if it evaluates to a \textit{zero} value, the function evaluation doesn’t continue, and the function is assigned the missing value. \texttt{ONLYIF} is examined \textit{before} doing \texttt{START}, unlike \texttt{REJECT} (below), which is done \textit{after} the \texttt{START}.
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.

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 User’s Guide, page UG–116.

[print]/noprint
vcv/[novcv]
title="title for output" [“MAXIMIZE”]
These are the same as for other regressions.

smpl=SMPL series or formula (“SMPL Option” on page RM–546)
weight=series of entry weights (“WEIGHT option” on page RM–549)
These are the standard options for skipping data points or reweighting observations. Note that MAXIMIZE has no SPREAD option.

robusterrors/[norobusterrors]
lags=correlated lags [0]
lwindow=newey/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 (Section 4.5 in the User’s Guide) or clustered standard errors if you are doing Quasi Maximum likelihood estimation. With MAXIMIZE, the LAGS, LWINDOW, LWFORM and CLUSTER options are used for dealing with correlation in the gradient elements. See “Long-Run Variance/Robust Covariance Calculations” on page RM–540

derives=(output) VECTOR[SERIES] for partial derivatives
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.

Hypothesis tests
If you have used METHOD=BFGS or METHOD=BHHH, you can test hypotheses on the coefficients using TEST, RESTRICT and MRESTRICT and compute standard errors for functions of the coefficients using SUMMARIZE. Coefficients are numbered by their positions in the parameter set. See “Technical Information” regarding the validity of these test statistics.

Missing Values
MAXIMIZE drops any entry which cannot be computed because of missing values.
Maximize

Technical Information

Chapter 4 in the User’s Guide describes the estimation methods and how the instruction NLPAR controls some of the finer adjustments of them. MAXIMIZE uses numerical derivatives to compute the gradient. After the first iteration, the perturbations used in computing these adapt to the estimates of the dispersion of the parameters.

As discussed in Section 4.5, 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 monotonic 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 ROBUSTERRORS.

Variables Defined by MAXIMIZE

MAXIMIZE defines the following standard estimation variables (see LINREG)

\%NREG, \%FREE, \%NOBS (INTEGER)
\%BETA, \%TSTATS, \%STDERRS (VECTOR)
\%XX (SYMMETRIC)

and the following other variables

\%FUNCVAL Final value of $G(\beta)$ (REAL)
\%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)
\%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 (X1 and X2), 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)) , $

compute b0=b1=b2=0.0
maximize probit
Box-Cox models can take several forms. The only one which requires use of `MAXIMIZE` rather than `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 `%BOXCOX`. The following estimates using maximum likelihood the model

\[ y_i^{(\lambda)} = \beta_0 + \beta_1 X_i^{(\lambda)} + u_i ; u_i \sim N(0, \sigma^2) \text{ i.i.d} \]

Initial guess values are obtained using a linear regression, which (with an adjustment to the intercept) is a special case for \( \lambda = 1 \).

```r
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 `SIGSQ`, while the regression parameters come off the vector `B` generated using `FRML` with the `LASTREG` option.

```r
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.

```r
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.

```
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, it’s treated as a series of 1’s.

**Options**

See “Long-Run Variance/Robust Covariance Calculations” on page RM–540 for a more complete explanation of the calculations and the main options (*LAGS*, *LWINDOW*, *CLUSTER*).

`lags=correlated lags  [0]`

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 which typically require a choice for *LWINDOW* other than the default of *LWINDOW=FLAT*. For the quadratic spectral window (*LWINDOW=QUADRATIC*), *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=newey/bartlett/damped/parzen/quadratic/[flat]/panel/white`

`damp=value of γ for lwindow=damped  [0.0]`

*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. *DAMP* gives the parameter $\gamma$ of the window with *LWINDOW=DAMPED* (if *DAMP* is set to something other than 0.0, *LWINDOW* is automatically set to *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

These are for a general clustered calculation. CLUSTER and GROUP are synonyms.

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. For more information, see “ZUMEAN and CENTER options” on page RM–541.

[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.

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.

meanvector=(output) 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.
MCov

**opgstat=value for OPG statistic [not used]**
Use this to get the OPG (outer-product gradient) statistic used in many LM tests. See “OPGSTAT Option (LM Tests)”.

**matrix=SYMMETRIC array for computed matrix [%CMOM]**
**rmatrix=RECTANGULAR array for computed matrix [none]**
Use **MATRX** to put the computed matrix into a specific SYMMETRIC array. By default, **MCOV** uses the array %CMOM. If you’ve put in an asymmetric window using LWFORM option, the resulting matrix won’t be a SYMMETRIC, so you **must** use RMATRIX.

**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 (Introduction, page Int–77).

**spread=standard SPREAD option (“SPREAD Option” on page RM–547)**
**smpl=SMPL series or formula (“SMPL Option” on page RM–546)**
These are the same as for regressions instructions.

**weight=series of entry weights (“WEIGHT option” on page RM–549)**
Use this option if you want to give the observations unequal weights.

**[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.

**OPGSTAT Option (LM Tests)**
The test statistics for many LM tests take the form

\[
(1) \quad \left( \sum Z'u \right) \left( \sum Z'u^2Z \right)^{-1} \left( \sum Z'u \right)
\]

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. Note that it’s up to you to determine the proper degrees of freedom of the test (usually a count of the variables added that weren’t in the original regression).
Examples

```plaintext
mcov(lwindow=bartlett, lags=lags) start1+1 endl resid
# constant
```

computes an estimate of the spectral density of RESIDS at frequency zero, using “LAGS” lags. The computed matrix (which here will be $1 \times 1$) will be in %CMOM.

```plaintext
mcov(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 — Spreadsheet Array Editor/Viewer

**MEDIT** opens a spreadsheet-style editing window that allows you to view, edit, or enter data into RECTANGULAR, SYMMETRIC, or VECTOR type arrays. You can also use *File—Export...* to save the contents of the window to a file in various formats. 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 code for data [none]
A picture code 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 the “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. The option PICTURE="*.*##" will limit the display to three digits right of the decimal. See “Picture Codes” on page RM–545 for more information.

prompt="string for window prompt"
This puts the text string you supply in 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
stype=[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:

```
declare rectangular r(10,10)
memit r
```

The 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. (This can also be done with **REPORT**).

```
do mas=0,3
   do ars=0,3
      boxjenk(constant,diffs=1,ma=mas,ar=ars,maxl, $  
         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
memit(hlabels=||"0","1","2","3"||,vlabels=||"0","1","2","3"||, $  
   picture="*.###", noedit) aic sic
```

The resulting **MEDIT** window will look something like this:

![MEDIT Window Example](image)
**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.

```
messagebox( 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. The buttons shown and the returned status are shown in the table below.

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 shown in the table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Button</th>
<th>Status Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>STYLE=ALERT</td>
<td>OK</td>
<td>1</td>
</tr>
<tr>
<td>STYLE=OKCANCEL</td>
<td>OK</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Cancel</td>
<td>0</td>
</tr>
<tr>
<td>STYLE=YESNO</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>STYLE=YNCANCEL</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Cancel</td>
<td>-1</td>
</tr>
</tbody>
</table>
Example

messagebox(style=yesno,status=yn) "View the residuals?"
if yn==1
  {  
    graph(header="Regression Residuals") 1  
    # %resids  
  }

This displays the following dialog:

![Messagebox 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.

```plaintext
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 initiates the process of modifying the structure of an EQUATION. VREPLACE and VADD do the actual changes, which can include substituting one variable for another, directly or in differenced form, or swapping a right-side variable for the dependent variable.

```
modify( option )   oldequation   newequation
```

Parameters

- **oldequation**: Source equation for the modification.
- **newequation**: (output) The target for the modified equation. By default, this is the same as oldequation.

Option

- **print/[noprint]**: Use PRINT to display the form of oldequation. Note that DISPLAY is a simpler way to do that.

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

Once you do a MODIFY, all subsequent VREPLACE and VADD instructions will apply to oldequation, until you do another MODIFY. If you do a series of VREPLACE or VADD operations to a single EQUATION, do only one MODIFY at the start of the sequence.

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 3 of the User’s Guide for more information.

mrestrict(options) restrictions Rmatrix rvec

Wizard

You can use the Statistics—Regression Tests wizard to do restriction tests, though it will produce a RESTRICT instruction rather than an MRESTRICT.

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.

rvec A VECTOR with dimension equal to restrictions. You can omit this if rvec is the zero vector. Use a * to skip rvec if you want to use either of the next two parameters.

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/[nvccv]
define= equation to define
frml= formula to define
unravel/[nounravel]
When used with CREATE, these perform the same task they do for LINREG.

form=f/chisquared
This affects only the test statistic, not the restricted regression. See page UG–64 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 rvec parameter fields.

Example
This does the polynomial distributed lag from Section 1.1 in the Additional Topics PDF 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

- %CDSTAT: the computed test statistic (REAL)
- %SIGNIF: the marginal significance level (REAL)
- %NDFTEST: (numerator) degrees of freedom for the test (INTEGER)
- %RESIDS: SERIES of residuals (if CREATE or REPLACE)
- %BETA: VECTOR of coefficients (if CREATE or REPLACE)
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**.

```
mvstats (options)  series  start  end  newseries  newstart
```

**Wizard**

You can use the **Data/Graphics—Moving Window Statistics** wizard 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**

- **width**=
  - *Width of the moving window [5]*
- **span**=
  - *same as **WIDTH** (used in older versions of **RATS**)*
- **centered**/[**nocentered**]
  - The **WIDTH** 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. 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.

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/shorten (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 (EXTEND=SHORTEN means the same) 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 (Quantiles)

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 $0.25X_{(3)} + 0.75X_{(4)}$.

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

`mvstats(variances=volatile,width=20) price`

sets the series `VOLATILE` at each period equal to the sample variance of the twenty entries ending at that period. `VOLATILE` is defined beginning at entry 20.

`mvstats(max=mhigh,min=mlow,width=13,center,extend=rescale) price`

sets the series `MHIGH` to the highest value achieved by `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.

`mvstats(fractiles=|.1,.25,.75,.9|,results=pricefract) price`

computes four different fractiles of `PRICE`, storing the results in a vector of series called `PRICEFRACT`.

`mvstats(iqrange=iqr,centered,width=21) x`

sets `IQR` to a moving interquartile range of series `x`, using current and ten entries on either side.

See Also . . .

<table>
<thead>
<tr>
<th>STATISTICS</th>
<th>With the fractiles option, computes a fixed list of fractiles.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTREMUM</td>
<td>Finds the maximum and minimum values of a series.</td>
</tr>
<tr>
<td>%MINVALUE (x)</td>
<td>Returns the minimum value of an array (or series).</td>
</tr>
<tr>
<td>%MAXVALUE (x)</td>
<td>Returns the maximum value of an array (or series).</td>
</tr>
<tr>
<td>%FRACTILES (x, f)</td>
<td>Computes a collection of fractiles for an array (or series).</td>
</tr>
</tbody>
</table>
NEWTYPE — Defining New Data Type

NEWTYPE allows you to define your own type name as shorthand for a more complicated data type.

\[
\text{newtype} \quad \text{newtype} \quad \text{datatype}
\]

Parameters

- **newtype**: The identifier that you want to use as shorthand for a more complicated data type. Note that, unlike standard RATS data types, you need to use the entire name, not just the first three letters.

- **datatype**: The data type that newtype can now be used to represent.

Example

```
new GARCHSwitch function[vector](integer)
declare GARCHSwitch msgarchfnc
```

creates the type name GARCHSwitch to represent a function of an INTEGER that returns a VECTOR.
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 15.1 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

(1) \[ y_t = f(X_t, \beta) + u_t \]

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) depvar start end residuals

**Parameters**

- **depvar**
  - the dependent variable. You can use an asterisk (*) for this parameter if the model is $f(X_t, \beta) = u_t$ so there is no “dependent variable.” If you use *, the regression output will exclude $R^2$ and similar summary statistics.

- **start end**
  - the estimation range. If you have not set a **SMPL**, this defaults to the maximum range over which the residuals can be computed, and the instruments are defined (if you are doing instrumental variables).

- **residuals**
  - (Optional) Series for the residuals at the final estimates.

**General Options**

- **frml=formula name** *(This option is required)*
  - The name of the formula for the function $f$.

- **parmset=PARMSET to use [default internal]**
  - This option selects the parameter set to be estimated. See page UG–129 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.

- **[print]/noprint**
- **vcv/[nocrv]**
- **title="title for output"** *(“Nonlinear Least Squares” or “Nonlinear Instrumental Variables”)*

  These are the same as for other regressions.
method=[gauss]/simplex/genetic/annealing/ga/evaluate

METHOD sets the estimation method to be used—see Chapter 4 of the User’s Guide for descriptions. The default (GAUSS) is Gauss–Newton, which requires that the formula be twice continuously differentiable and is the only choice that can compute standard errors. The others make weaker assumptions, and can compute point estimates only—they are more often used as a PMETHOD to refine initial guesses before using Gauss–Newton to finish the estimation.

EVALUATE simply evaluates the formula given the initial parameter values and produces no output.

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 4 in the User’s Guide for more details.

pmethod=gauss/[simplex]/genetic/annealing/ga
piter=n=number of PMETHOD iterations to perform [none]

Use PMETHOD and PITER if you want to do preliminary iterations using one method, and then switch to another method for final estimates. For example, to do 10 simplex iterations before switching to Gauss-Newton, you can use the options PMETHOD=SIMPLEX, PITERS=10, and METHOD=GAUSS.

startup=FRML evaluated at period “start”
onlyif=expression tested before calculating start option [not used]

You can use the START option to provide an expression which is computed once per function evaluation, before any of the regular formulas are computed. This allows you to do any time-consuming calculations that depend upon the parameters, but not upon time. It can be an expression of any type. ONLYIF calculates the expression provided; if it evaluates to a zero value, the function evaluation doesn’t continue, and the function is assigned the missing value. ONLYIF is examined before doing START, unlike REJECT (below), which is done after the START.

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.

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.
NLLS

\texttt{smpl=} SMPL series or formula ("SMPL Option" on page RM–546)
\texttt{weight=} series of entry weights ("WEIGHT option" on page RM–549)
\texttt{dfc=} Degrees of freedom correction (Additional Topics, Section 1.4)

These are the standard options for skipping data points, reweighting observations or correcting for degrees of freedom lost in preliminary processing.

Options for Non-Linear Least Squares

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

When used without the \texttt{INSTRUMENTS} option, these permit calculation of a consistent covariance matrix allowing for heteroscedasticity (with \texttt{ROBUSTERRORS}) or serial correlation (with \texttt{LAGS}), or clustering based upon some other set of categories. For more information, see Sections 2.2, 2.3, and 2.4 of the \textit{User's Guide} and "Long-Run Variance/Robust Covariance Calculations" on page RM–540. Note especially the possible computational problems that may arise when using \texttt{LAGS}.

\texttt{spread=} standard \texttt{SPREAD} option ("\texttt{SPREAD Option}" on page RM–547)
\texttt{sv=} Residual variance formula

Use one of these for weighted least squares. The difference is that \texttt{SPREAD} is used when the variances don't depend upon the parameters (so simple weighted least squares can be used) and \texttt{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.

\texttt{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)

\texttt{instruments/ [noinstruments]}

Use the \texttt{INSTRUMENTS} option to do two-stage least squares or GMM. You must set your instruments list first using the instruction \texttt{INSTRUMENTS}.

\texttt{wmatrix=} weighting matrix \([\{Z'Z\}^{-1}]\)
\texttt{iwmatrix=} inverse weighting matrix \(\text{[not used]}\)
\texttt{optimalweights/ [nooptimalweights]}
\texttt{update=} [none]/once/continuous

This controls the updating of the weighting matrix. You can directly input a weight matrix with the \texttt{WMATRIX} or \texttt{IWMATRIX} 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.

```
zudep/nozudep
lags=correlated lags [0]
lwindow=newey/bartlett/damped/parzen/quadratic/[flat]/panel/white
damp=value of γ for lwindow=damped [0.0]
lwform=VECTOR with the lag window form [not used]
cluster=SERIES with category values for clustered calculation
    Use these to select the type of correlation assumed for the $Z'u$ process. Note that the default for the ZUDEP (ZU Dependence) option is different from the NLSYSTEM instruction. For more information, see the "Long-Run Variance/Robust Covariance Calculations" on page RM–540.

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. For more information, see "ZUMEAN and CENTER options" on page RM–541.

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 page UG–140 in the User’s Guide for more information.
```

Description
By default, NLLS estimates using the Gauss–Newton algorithm with numerical partial derivatives. See User’s Guide Sections 4.2 and 4.8 for more on this.

Hypothesis tests
You can test hypotheses on the coefficients with TEST, RESTRICT and MRESTRICT and you can use the Statistics—Regression Tests wizard to set those up. 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* $ 
\quad \log( \delta \times k^{(-\rho)} + (1-\delta) \times L^{(-\rho)} )
compute lgamma=1.0, delta=0.4, nu=0.8, rho=.6
nlls(frml=ces,trace) q

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**)

\%BETA, \%DURBIN, \%MEAN, \%NDF, \%NFREE, \%NOBS, \%NREG, \%RESIDS, \%RSS, 
\%SEESQ, \%RHO, \%STDERRS, \%TSTATS, \%VARIANCE, \%XX

for least squares only

\%LOGL, \%RBARSQ, \%RSQUARED, \%TSQ, \%TRSQUARED

for instrument variables only

\%JSIGNIF, \%JSTAT, \%NDFJ, \%UZWZU, \%WMATRIX

and the following other variables:

\%FUNCVAL Final value of \( G(\beta) \) (**REAL**)
\%ITERS Iterations taken (**INTEGER**)
\%CONVERGED \( = 1 \) or \( 0 \). Set to \( 1 \) if the process converged, \( 0 \) if not (**INTEGER**)
\%CVCRT 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.
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).

\texttt{nlpars( options ) (no parameters)}

Description

The instructions \texttt{BOXJENK} and \texttt{ITERATE} (ARMA estimation), \texttt{NLLS} and \texttt{NLSYSTEM} (Non-Linear Least Squares and systems estimation), \texttt{DDV} and \texttt{LDV} (discrete and limited dependent variables), \texttt{ESMOOTH} (exponential smoothing), \texttt{MAXIMIZE}, \texttt{FIND}, \texttt{DLM} (dynamic linear models) and \texttt{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 those 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 \texttt{NLPAR} provide control over these settings.

In general, there are five main methods of optimization used by RATS: the hill-climbing methods (Section 4.2 of the \textit{User's Guide}), and the simplex, genetic, simulated annealing and genetic annealing (Section 4.3). Of these, the ones most likely to benefit from “tuning” are the last three, which are all designed to deal with surfaces with possibly multiple peaks. If you do have multiple peaks, it requires quite a bit more effort (bigger populations, slower speed of convergence) to allow all of those to be explored properly. The genetic algorithm has four controls and the annealing algorithm does as well (and genetic annealing uses both sets).

Options

The following is a general description of the \texttt{NLPAR} options. Please see Chapter 4 in the \textit{User's Guide} for technical details on the parameters controlled via \texttt{NLPAR}, as well as general information on the algorithms and methods used.

\texttt{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 4.2 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.

derive=[first]/second/fourth
This determines the method used for computing numerical derivatives. The default is a right arc derivative. DERIVE=SECOND and DERIVE=FOURTH use more accurate but slower formulas. In most cases, you won't need those.

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 4.3)

aschedule=fraction by which temperature is reduced on each iteration [.85]
ainner=number of inner repetitions before $V$ is adjusted [20]
aouter=scale of the number of parameters for the number of outer repetitions [3]
avadjust=adjustment control for the $V$ values [2.0]
These parameters control details of the ANNEALING estimation method (added with version 9.2)

marquardt/[nomarquardt]
For Gauss-Newton estimation, you can use MARQUARDT to select the Levenberg–Marquardt subiteration algorithm (Marquardt, 1963). We generally don’t recommend this, as we have generally obtained better results using the default method.
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 4.3 in the *User’s Guide* for more information.

Sets the default convergence limits and subiteration limits. Since the estimation instructions to which these apply have their own `CVCRIT` and `SUBITERS` options, you should use those instead.
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 $u_i$ either as $y_i = f(X_i, \beta) + u_i$ or as $f(X_i, \beta) = u_i$. **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 **MODEL** option to provide a full model, or the **FRMLVECT** option to provide the formulas as a **FRML** which returns a **VECTOR**.

### 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) \( u_t = (u_{1t}, \ldots, u_{nt})' \) 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) \( Z_t = (z_{1t}, \ldots, z_{rt})' \) as 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{parmset=PARMSET to use [default internal]}

This option selects the parameter set to be estimated. See Section 4.6 of the \texttt{User's Guide} for more on \texttt{PARMSET}s. RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.

\texttt{smpl=SMPL series or formula ("SMPL Option" on page RM–546)}

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.
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 4 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 $\Sigma$ or weight matrix) are re-estimated.

startup=FRML evaluated at period “start”
onlyif=expression tested before calculating start option [not used]

You can use the START option to provide an expression which is computed once per function evaluation, before any of the regular formulas are computed. This allows you to do any time-consuming calculations that depend upon the parameters, but not upon time. It can be an expression of any type. ONLYIF calculates the expression provided; if it evaluates to a zero value, the function evaluation doesn’t continue, and the function is assigned the missing value. ONLYIF is examined before doing START, unlike REJECT (below), which is done after the START.

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.

[print]/noprint
vcv/[novcv]
title="title for output" [depends upon options]

These are the standard output control options.

sigma/[nosigma]

This controls the printing of the final estimate of the residual covariance matrix.

residuals=(output) 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 $\Sigma$ matrix (SYMMETRIC)
cvout=Output $\Sigma$ 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. 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, prior to version 7.
**derives**=RECTANGULAR[SERIES] for partial derivatives

This saves 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.

**Options for Multivariate Least Squares**

**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.

**robusterrors**/[norobusterrors]

**lags**=correlated lags [0]

**lagswindow**=neweywest/bartlett/damped/parzen/quadratic/[flat]/

**panel**=white

**damp**=value of $\gamma$ for **lagswindow**=damped [0.0]

**lwform**=VECTOR with the window form [not used]

**cluster**=identifying SERIES for clustered std. errors [not used]

When you use these without the **INSTRUMENTS** option, they allow you to calculate a consistent covariance matrix allowing for heteroskedasticity (with **ROBUSTERRORS**), serial correlation (with **ROBUSTERRORS** and **LAGS**), or clustered standard errors (with **ROBUSTERRORS** and **CLUSTER**). For more information, see Sections 2.2, 2.3, and 2.4, and 4.5 of the *User's Guide* and "Long-Run Variance/Robust Covariance Calculations" on page RM–540.

Note that none of these options affect the parameter estimates. Just as with **LINREG** and **NLLS**, these options come into play when the covariance matrix of the estimates is computed. These behave differently when you are using **NLSYSTEM** with the **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.
zudependent/[nozudep]

wmatrix=SYMMETRIC weighting matrix for instruments \([Z'Z]^{-1}\]
iwmatrix=SYMMETRIC inverse weighting matrix
sw=SYMMETRIC grand weighting matrix [not used]
isw=SYMMETRIC inverse grand weighting matrix
swout=(output) SYMMETRIC grand weighting matrix [not used]

NOZUDEP (the default) is the special case for the SW matrix. We call it 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 NOZUDEP, you can use the WMATRIX or IWMATRIX option (whichever is more convenient) 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 or ISW 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

\[
\frac{1}{T} \sum (u_t \otimes Z_t)(u_t \otimes Z_t)'
\]

(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.

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 or ISW option, the default is UPDATE=NONE.
- If you use the CV option with NOZUDEP, the default is UPDATE=ONCE.

This option replaces the SECONDSTEP option used before version 7.

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 2.4 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 page UG–140 in the User’s Guide for more information.

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. For more information, see "ZUMEAN and CENTER options" on page RM–541.

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.

mask=RECTANGULAR masking array for instruments [not used]
Normally, NLSYSTEM uses the same set of instruments (those listed on the INSTRUMENTS instruction) for all equations. 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.
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, \%NFREE is number of free parameters (including estimated covariance matrix), \%NOBS is the number of observations, \%NVAR the number of equations. Other variables defined are:

\%LOGDET log determinant of the estimate of $\Sigma$ (REAL)
\%SIGMA final covariance matrix of the residuals (SYMMETRIC)
\%LAGRANGE VECTOR of Lagrange multipliers if estimating with constraints.

For multivariate least squares only
\%LOGL (normal) log likelihood (REAL)

For instrumental variables/GMM only
\%JSTAT Test statistic for overidentification for instrumental variables (REAL). 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)
\%NDFJ Degrees of freedom of \%JSTAT (INTEGER)
\%UZWZU criterion function for instrumental variables (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 SUR iterated to convergence.

The second NLSYSTEM uses an alternative investment equation which includes first order autocorrelated errors. This is done by adding a lagged dependent variable and a lag of the base investment formula to create the formula 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 ITERS option has been greatly increased for this last estimator. Estimating with ZUDEP (which is implicit when you use 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.

```
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,lwindow=newey,)$
    iters=400) 1950:1 1985:4 investnl consnl ratenl
```
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 13.3 for more on neural networks.

```
nlearn(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 that 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 page UG–444 of the User’s Guide.

mode=[epoch]/example
This controls how often new weights are computed. With EPOCH, NNLEARN 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).

**smpl=SMPL series or formula**  
("SMPL Option" on page RM–546)  
This series or formula should return 0 for entries you want to omit from the training, and non-zero values for the other entries.

**iters=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 **MAXIMIZE** instruction, but also has less of a chance to produce an improvement. You might need an iteration limit in the thousands.

**cvcrit=convergence criterion**  
[.00001]  
**rsquared=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 **CVCRT** option, **NNLEARN** 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 **CVCRT** value.

If you use the **RSQUARED** option, **NNLEARN** will train the model until the mean square error is less than $(1-R^2)s^2$, where $R^2$ is the value specified in the **RSQUARED** option, and $s^2$ is the smallest of the output series variances.

The default setting is **CVCRT**=.00001. If you specify both options, **NNLEARN** will use the **CVCRT** setting.

**trace/ [notrace]**  
If you turn on the **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 **TRACE**, particularly when developing new models.

**theta=theta parameter**  
[0.7]  
(must be in the range $0 \leq \theta < 1$)  
**kappa=kappa parameter**  
[0.1]  
(must be in the range $0 \leq \kappa < 1$)  
**phi=phi parameter**  
[0.5]  
(must be in the range $0 < \phi < 1$)  
**mu=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 **THETA** value affects how derivatives are averaged over recent derivatives. **KAPPA** affects how a weight is increased if the current derivative has the same direction as recent derivatives, while **PHI** affects how the weights change if the derivative changes direction. A non-zero value for the **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 $X_1, X_2, \ldots, X_n$ and the output series are $Y_1, Y_2, \ldots, Y_m$, this fits:

$$Y_i \sim F(X_j), \text{ for } i=1,2,\ldots,m; \ j=1,2,\ldots,n$$

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.

```r
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:

\[
12 + (I + 1) \times H + (H + 1) \times O + 8 + (2 \times O)
\]

(with NODIRECT option), or

\[
12 + (I + 1) \times H + (H + 1) \times O + (I \times O) + 8 + (2 \times O)
\]

(with DIRECT option)

The first 12 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:

memory(13,...,P) = \( \alpha_{ij} \), for \( i=1,...,H; j=0,...,I \) (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 \( j \).

memory(P+1,...,Q) = \( \beta_{ij} \), for \( i=1,...,O; j=0,...,H \)

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 \( j \).

memory(Q+1,...,R) = \( \delta_{ij} \), for \( i=1,...,O; j=1,...,I \) (only when using the DIRECT option)

If you use DIRECT, 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 \( i \).

memory(R+1,...,S=R+9) = Eight zero/one flags relating to the scaling options used.

memory(S+1,...,S+(2 \times O)) = Pairs of values for each output node, containing the lower and upper bounds when using YMAX or YMIN, or the lower and upper scale factors when using PAD.
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 13.3 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 or formula ("SMPL Option" on page RM–546)
```

If you use the SMPL option, **NNTEST** will only compute output for entries where the SMPL series or formula has a non-zero value. No output will be calculated for entries where it's 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 **NLLS, NLSYSTEM, MAXIMIZE, FIND, CVMODEL,** and **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 **NONLIN.** **NONLIN** creates a data type called a **PARMSET** which allows you to switch between parameter sets quickly. **PARMSETS** can even be combined, making it easy to adjust parameter sets. **NONLIN** can also impose constraints on the parameters.

```
nonlin(options) parameterfields (separated by blanks)
```

Wizard

You can define a list of nonlinear parameters when using the **Statistics—Equation/FRML Definition** wizard to define a formula.

Parameters

**parameterfields**  A parameterfield is one of the following:

- a simple **REAL** variable, such as B1
- a real array (**VECTOR, RECTANGULAR, SYMMETRIC, PACKED**)
- an array of arrays, such as a **VECTOR** of **VECTORS**.
- a substitution operation: \( B3 = B1 \times B2 \)
- an equality constraint: \( B3 = B1 \times B2 \)
- an inequality constraint: \( B3 \geq 0.0 \). You cannot use a strict inequality here. For instance, \( B3 > 0.0 \) is illegal.

See the notes below on the use of these.

Options

**parmset=PARMSET to define [default internal]**

Using the **PARMSET** option, you can define or redefine a parameter set. By using the **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.

**zeros/[nozeros]**

If you use the **ZEROS** option, the elements of the **PARMSET** are initialized to zero. By default, whatever values they previously had are retained—a new variable name will be initialized to **NA**. If you use the **PARMSET** without initializing some of the values, you will get a warning message like:

```
## NL6. NONLIN Parameter C1 Has Not Been Initialized. Trying 0
```
Thus, zero is (in effect) the default guess value for any parameter—the warning is there in case you run into numerical problems related to not having proper guess values. If zeroes are a reasonable set of guesses (for instance, linear functions aren't sensitive to guess values), then the ZEROS option will allow you to use the PARMSET without explicit guess values and without getting the warnings.

```
add/[noadd]
```

This allows you to add parameters or constraints to a parameter set without reentering the full set. This is largely obsolete because you can now separate the parameter set into different parts and “add” them using the + operator when you need to use them for estimation. That is, MAXIMIZE (PARMSET=MODELPARMS+GARCH) will combine the MODELPARMS and 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 should 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
```

You can use the DISPLAY instruction to display a list of the variables in a parameter set, along with the current values of those variables. This only works for “named” parameter sets—you cannot display the contents of the (unnamed) default internal set.
Examples

\begin{verbatim}
nonlin alpha lambda sig_eta sig_eps

defines the default parameter set as having the five named parameters.

dec vect b(n)
nonlin(parmset=meanparms) b

defines the \texttt{PARMSET} named \texttt{MEANPARMS} as a \texttt{VECTOR} of dimension \texttt{N}.

nonlin(parmset=base) mu1 mu2 mu3 $
  p11 p12 p13 p21 p22 p23 p31 p32 p33
nonlin(parmset=relax) ep14 ep24 ep34
nonlin(parmset=symmetry) p12=p21 p13=p31 p23=p32

This defines three \texttt{PARMSETS}. \texttt{BASE} has a base set of parameters, \texttt{RELAX} has some added ones and \texttt{SYMMETRIC} forces equality constraints among three pairs. Note that these use \texttt{=} which means that they are forced to be equal.

nonlin(parmset=lrrest) lr(1)(2,3)==0.0

This defines a \texttt{PARMSET} called \texttt{LRREST}. This imposes an implicit constraint (note the \texttt{==}). This is used because the \texttt{LRF(1)(2,3)} is a matrix calculation of other parameters.
\end{verbatim}
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.

```plaintext
npreg( options )   Y series   X series   start   end   grid   fit
```

Wizard

The Statistics—Nonparametric Regression wizard provides access to the major features of NPREG.

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

**bandwidth=** kernel bandwidth [see below]

TYPE selects the kernel type for the Nadaraya-Watson estimator. BANDWIDTH specifies the bandwidth for the kernel. The default value is 

\[
\left(0.79 \text{ IQR}\right)/N^{1/5}
\]

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.

**smoothing=** smoothing scale vector [1]

You can supply a real value (bigger than 0) to adjust the amount of smoothing. Use a value bigger than 1 for more smoothing than the default, values less than 1 for less smoothing.

**smpl=** SMPL series or formula ("SMPL Option" on page RM–546)

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.

**weight=** series of entry weights ("WEIGHT option" on page RM–549)

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.
Technical Information

The Nadaraya-Watson estimator (METHOD=NADARAYA) 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 kernels have the 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**  \[ K(v) = e^v/(1 + e^v)^2 \]

**FLAT**  \[ K(v) = 0.5 \text{ if } |v| \leq 1, \text{ and } 0 \text{ otherwise} \]

**PARZEN**  \[ K(v) = 4/3 - 8v^2 + 8v^3 \text{ if } |v| \leq 0.5, \]

\[ 8(1 - |v|^3)/3 \text{ if } 0.5 \leq v \leq 1, \]

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** (METHOD=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:

$$1 - \left(\frac{|x_i - x|}{D(x)}\right)^3$$

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.

Finally, METHOD=NN takes the simple average of $y$ for the requested fraction of the data closest to each value of $x$. 

---

*RATS Reference Manual  RM–353*
Examples

This example implements several non-parametric regressions from the NIST Engineering Statistics Handbook. See the file LOWESS.RPF for the complete code and the web link for the NIST document.

The first estimates using LoWess with \( \text{FRAC} = .33 \). The default \( \text{FRAC} = .5 \) 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.

All of the estimated functions are graphed using SCATTER, with the original data done as dots, and the functions done as lines. Note the use of 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.RPF 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,smoothing=.75) y x / xv yx
scatter(style=dots,overlay=line,ovsame,$
   header="Kernel Fit, Smaller Bandwidth") 2
   # x y
   # xv yx
```

Variables Defined

\%EBW The computed bandwidth
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]/binary/cdf/dbf/dif/html/matlab/portable/prn/rats/rtf/tex/tsd/wf1/xls/xlsx/’(FORTRAN)’**

- **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.

- **ODBC**
  Opens a connection to an ODBC/SQL database.
Open

PLOT  Intermediate file for high-resolution graphics.
KEYBOARD  The keyboard.
SCREEN  The output screen or window.
PRINTER  The printer.

The last three are predefined “hardware” units. You can never “OPEN” them. You can open files as INPUT and OUTPUT, but there are superior alternatives for those units. For instance, the instruction SOURCE switches the INPUT unit to a different file, but 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 also define your own names for I/O units. By defining your own units, you can keep several data or output files open at a time and switch between them.

Examples

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.

compute fname=thiscountry+.rat"
open data &fname
data(format=rats)

opens as the DATA unit a file whose name is created from appending “.rat” to the current contents of the string THISCOUNTRY and reads its contents. (DATA is the default I/O unit for the DATA instruction).

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 CHANGE to redefine an I/O unit, which can be particularly useful for redirecting output to different files.

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 page UG–475 of the User’s Guide for more information.

<table>
<thead>
<tr>
<th>option switch</th>
<th>optionname</th>
<th>default</th>
<th>(1 for ON, 0 for OFF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>option choice</td>
<td>optionname</td>
<td>default number</td>
<td>list of choices</td>
</tr>
<tr>
<td>option</td>
<td>datatype</td>
<td>optionname</td>
<td>default value</td>
</tr>
</tbody>
</table>

Parameters

`optionname` This is the name which you are assigning to the option. Within the procedure, you must use its full name. However when you execute 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, page UG–475).

`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 description of the instruction.

`datatype` 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.
- for **FUNCTIONS**, a **FUNCTION** which has the identical return type and parameter list.

**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=optionname ; for instance, 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 %CHOICE Function

If you are trying to pass information to a choice option on a RATS instruction, you can also use the function %CHOICE(choicelabel). For instance,

```plaintext
compute fchoice=%if(spencer,"spencer","henderson")
filter(type=%choice(fchoice)) ...
```

will pick TYPE=SPENCER if the SPENCER variable is "true" and TYPE=HENDERSON if not.

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 final example below.

Examples

```plaintext
option  choice  type 1  flat tent
```

The TYPE option has choices FLAT and TENT with FLAT (choice 1) the default.

```plaintext
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. This procedure requires that the user supply a series of threshold values using 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.

```plaintext
if .not.%defined(threshold).or..not.%defined(depvar)
{
    display $
    "Syntax: @tsaytest(threshold=threshold series) depvar start end"
    display " # list of regressors"
    return
}
```
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. Since it generally rearranges the elements of series, it may be a good idea to apply this to a copy of the data.

order( options )  series  start  end  list of series

Parameters

series  Series to sort or rank.
start  Range of entries to sort or rank. If you have not set a SMPL, this defaults to the defined range of series.
end
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  ("SMPL Option" on page RM–546)
Limits the sort to the observations for which the series or formula returns a non-zero or true value.

index=SERIES[INTEGER] for index series  [not used]
This save 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

```r
order pop / spend tax aid
```
reorders POP, SPEND, TAX and AID based upon POP.

```r
order(rank=xrank) xseries
order(rank=yrank) yseries
linreg yrank
# 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 one of a number of important transformations required for operating with panel or generally grouped data series. You can use it to implement regressions on panel data which can’t be done by using PREGRESS. See Section 12.5 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.

```
panel(options)  series  start  end  newseries  newstart
```

Parameters

- **series**: Source series.
- **start** and **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 count per individual [0.0]
- **tcount**: Weight on the count per time period [0.0]
- **isum**: Weight on the individual sum [0.0]
- **tsum**: Weight on the time sum [0.0]
- **mean**: Weight on the series mean [0.0]

These supply the weights on the various components. See "Description" for details.

- **effects=[individual]/time/both**

  This indicates whether to allow for INDIVIDUAL effects, TIME effects or BOTH. This applies when you use the GLS and related options or the DUMMIES option.

- **group=SERIES or FRML with values defining individuals**

  This is an alternative to a panel data setup for data. This defines the individuals. If you use GROUP, you can’t do any calculations which require identifying specific time periods (TIME, TCOUNT or TSUM components, EFFECTS=TIME or EFFECTS=BOTH).
vrandom= (input) variance of the random component
vindiv= (input) variance of the individual component
vtime= (input) variance of the time component
gls=[standard]/forwards/backwards
  If you’re doing a GLS transformation for random effects, using options like INDIV and TIME requires you to take the estimated component variances and convert them into the proper linear combination of entries and averages. And a single linear combination will only work if your data set is a balanced panel. As an alternative, you can input the component variances using the VRANDOM, VINDIV and (if necessary) VTIME options and let PANEL do the calculations.

There are several ways to “factor” the random effects covariance matrix to get this transformation. Which one is chosen is controlled by the GLS option. GLS=STANDARD does the standard symmetrical transformation using individual and time means. That can be used with any choice for EFFECTS. The other two choices for GLS can be used only with EFFECTS=INDIV. GLS=FORWARDS does a transformation using only “forward” calculations of means, while GLS=BACKWARDS uses only “backwards” means. The forward mean at \( t \) for individual \( i \) is the mean using only \( x(i,s) \) for \( s \geq t \).

The choice for the GLS option won’t affect the results if you use the transformed data in a standard LINREG instruction. It will matter if you are using it as an input to (for instance) an instrumental variables estimator.

kr= across time periods covariance matrix of residuals [not used]
  This is used to do the Keane-Runkle (1992) transformation, which is a transformation within each individual using a forwards factorization of a general covariance matrix across time periods. This requires a balanced panel data set.

smpl=SMPL series or formula ("SMPL Option" on page RM–546)
  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 \( N \times T \)).

id=VECTOR of (sorted) individual or group values [not used]
identries=VECTOR[VECTOR[INTEGER]] with the list of entries for each individual in ID [not used]
  If you have to do any calculation over the entries covered by an individual that is more complicated than the means and variances done with the other options of PANEL, you will need a simple way to organize those calculations. If you have a balanced panel data set, that’s fairly easy. It’s harder if you have an unbalanced
panel data set or general grouped data. For those situations, you can use the ID
and IDENTRIES options. ID returns a VECTOR of values of the grouping variable
(in sorted order), while IDENTRIES provides (for each value in ID) the corre-
responding set of entry numbers. These would be used something like the example
below, which walks through the individuals (the I loop) and then the entries for
that individual (the J loop, with IT being the entry number).

\begin{verbatim}
panel(group=p_cusip, id=vid, identries=identries)
do i=1, %size(vid)
do j=1, %size(identries(i))
   compute it=identries(i)(j)
   ...
end do j
end do i

dummies=VECT[SERIES] of dummies [not used]
This generates a VECT[SERIES] of dummy variables for EFFECTS=INDIV or
EFFECTS=TIME. (If you need both, do two separate PANEL instructions).

spread=(output) 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 (User’s Guide, Section 2.3). Note
that this computes a centered variance.

Description
With \( y_{it}; i=1,...,N; t=1,...,T \) representing series, for entry it:

\begin{description}
\item[ENTRY] is \( y_{it} \)
\item[INDIV] is \( y_{i\cdot} \), the mean of \( y \) for individual \( i \), averaged across \( t \)
\item[TIME] is \( y_{\cdot t} \), the mean of \( y \) for time \( t \), averaged across \( i \)
\item[MEAN] is the mean across all entries
\item[ISUM] is the sum across \( t \) of \( y \) for individual \( i \)
\item[TSUM] is the sum across \( i \) of \( y \) for time period \( t \)
\item[ICOUNT] is the number of valid observations (time periods) in the current individual.
   That is, for \( y_{it} \), ICOUNT returns the count of valid observations across all \( t \) for
   individual \( i \).
\item[TCOUNT] is the number of valid observations across individuals for the current time
   period. That is, for \( y_{it} \), TCOUNT returns the count of valid observations across all \( i \)
   at time period \( t \).
\end{description}

For example, you would use the following options to create the indicated series:

For \( y_{it} - y_{i\cdot} \), use options ENTRY=1.0, INDIV=-1.0
For \( y_{it} - 0y_{\cdot t} \), use options ENTRY=1.0, TIME=-THETA
Missing Values

PANEL removes any missing values from any average it calculates.

Examples

panel(entry=1.0,time=-1.0,smpl=oecd) lnrxrate / cxrate

Computes the deviations from time period means for LNRXRATE, using only the entries for OECD countries.

panel(effects=time,dummies=tdummies) constant

Creates TDUMMIES as a set of time period dummies.

panel(group=townid,icount=1.0) %resids / bcount
dofor ss = mv crim zn indus chas nox rm age dis rad tax ptratio b lstat

panel(group=townid,indiv=1.0) ss / %s("p_"+%l(ss))
end dofor ss

This creates BCOUNT as a series of counts of (valid) data points for each individual, and P_MV to P_LSTAT as the individual averages of a group of data series. All these use the TOWNID series to identify the individuals.

linreg(robust) lgaspcar
# lincomep lrpmg constant idummies
*
panel(spreads=countryvar) %resids
linreg(spread=countryvar) lgaspcar
# lincomep lrpmg constant idummies

This does a two-step feasible weighted least squares allowing for each individual to have a different variance.

Variables Defined

%NGROUP number of individuals or groups used in the computation that actually contain data. Does not include individuals/groups where all time periods are empty (missing values).
**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 12.5 of the *User's Guide* for general information on using panel data. See the "Description" segment 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** and **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**

- **input=[indiv]/time**
  - The default behavior for **PFORM** is to take a set of series where each series contains the data for one individual (across time), and stack those “individual” series into a single “panel” series. Use **INPUT=TIME** if you instead have separate series for each time period (that is, a given series contains data for all individuals for a single time period). You can also use the **TRANSPOSE**, **INDIVIDUAL**, or **TIME** options described below to handle other situations—any of those options will over-ride the **INPUT=INDIV** default.

- **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.

repeat/[norepeat]
If you have a series which is common to all individuals (there should be only one input variable), and you have already set up a panel CALENDAR scheme, PFORM with REPEAT will replicate one set of data across all individuals.

Variables Defined
%NGROUP number of individuals in the created series (INTEGER)
%NOBS number of time periods in the created series (INTEGER)

Description
PFORM constructs new series 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 start and end to the range of each of these you want included in the concatenated series. If you leave start and 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.

Each time period is in a separate series
If you have a separate series for each time period, list the set of series on the supplementary card and use the INPUT=TIME option.

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 oc-
curs, 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
This creates a single panel series from eight separate time series and resets the CAL-ENDAR to the appropriate values.

```
open data panel.xls
calendar(q) 1980:1
data(format=xls.org=columns) 1980:01 2008:01 australia canada $
    france germany japan netherlands uk us
pform(exrate)
    # australia canada france germany japan netherlands uk us
* cal(panelobs=%nobs,q) 1980:1
all %ngroup//2008:1
```

This reblocks an unbalanced sample using the ID and TIME options to identify the two dimensions.

```
open data abdata.dta
data(format=dta) 1 1031 ind year emp-wage-cap indoutpt n w k $
    ys rec yearm1 id n11 n12 w11 k11 k12 ys11 ys12 yr1976 yr1977 $
* pform(indiv=id,time=year) p_n
    # n
pform(indiv=id,time=year) p_w
    # w
pform(indiv=id,time=year) p_k
    # k
pform(indiv=id,time=year) p_ys
    # ys
pform(indiv=id,time=year) p_year
    # year
* cal(panel=%nobs)
all %ngroup/%panelobs()
```
**POLAR: Polar Decomposition**

**POLAR** computes the polar decomposition of a complex series. A complex number may be decomposed as

\[ z = |z| \exp(i\theta), \quad -\pi < \theta < \pi \]

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 π, 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 π’s for the axis labels. This doesn’t affect the numbers.

grpam(font="Symbol") axislabels 14
   These are good labels for monthly data. For quarterly data, labeling at 0, π/4, π/2, 3π/4 and π 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
```
PREGRESS — Panel Data Regressions

PREGRESS estimates a linear regression on a panel or grouped data set using one of several specialized techniques. See page UG–413 of the User’s Guide for more on panel data.

\[ \text{pregress (options) depvar start end resids} \]
# list of explanatory variables in regression format

Wizard

The Statistics—Panel Data Regressions wizard 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

\[ y_{it} = X_{it}\beta + u_{it}, \quad \text{where} \]

\[ u_{it} = \varepsilon_i + \lambda_t + \eta_{it}, \quad \text{unless METHOD=SUR} \]

$\varepsilon$ is the individual effect, $\lambda$ is the time effect and $\eta$ the purely random effect. If you use the option EFFECTS=INDIV, or METHOD=FD, the decomposition only includes the $\varepsilon$ and $\eta$ components. With EFFECTS=TIME, it only includes $\lambda$ and $\eta$.

METHOD=POOLED just estimates (1) by least squares, with no panel effects. METHOD=BETWEEN estimates (1) by least squares on individual averages. If you use METHOD=FIXED, $\varepsilon_i$ and $\lambda_t$ are treated as constants and are “swept” out. With METHOD=RANDOM, they are treated as part of the error term and $\beta$ is estimated by GLS. If METHOD=FD (first difference), the data are differenced to eliminate $\varepsilon_i$.

METHOD=SUR 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 VRANDOM, VINDIV and VTIME options, or you can allow PREGRESS to estimate them. There are many ways to estimate consistently these variances; most of the commonly used choices can be implemented by a combination of the VCOMP and CORRECTION options.
Options

`effects=[individual]/time/both`
This indicates whether to allow for INDIVIDUAL effects, TIME effects or BOTH.

`method=[fixedeffects]/randomeffects/fd/sur/between/pooled`
This chooses the estimation method: fixed or random effects, first difference, cross-section SUR, the “between” estimator, or pooled panel regression.

`group=SERIES or FRML with values defining individuals`
This is an alternative to a panel data setup for data. This defines the individuals.
If you use GROUP, you can only do EFFECTS=INDIV.

`vrandom=variance of the random component [estimated]`
`vindiv=variance of the individual component [estimated]`
`vt ime=variance of the time component [estimated]`
`vcomp=[wk]/sa/wh/ml/greene/wooldridge`
`correction=[full]/degrees/none`
You can input the component variances for random effects using VRANDOM, VINDIV and (if necessary) VTIME. If you don’t use these, PREGRESS will estimate them. The algorithm used for this is controlled by the VCOMP and CORRECTION options. With the exception of VCOMP=ML (maximum likelihood), these all solve a set of equations using quadratic forms in residuals for various estimators with different choices for the quadratic forms, estimators, and level of detail in the multipliers in the equations. Any of these will give consistent estimators. VCOMP=WK (the default) is Wansbeek-Kapteyn, VCOMP=SA is Swamy-Arora, VCOMP=WH is Wallace-Hussain, VCOMP=ML is maximum likelihood. VCOMP=GREENE and VCOMP=WOOLDRIDGE are (simpler) estimators proposed in the Greene (2012) and Wooldridge (2010) textbooks. The CORRECTION option controls the level of detail used in computing the coefficients in the quadratic forms. See the "Technical Information" for more.

`indiv=(output) series of individual effects [not used]`
`time=(output) 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.

`instruments/ [noinstruments]`
Use the INSTRUMENTS option to do instrumental variables. You must set your instruments list first using the instruction INSTRUMENTS. This can be used with any of the METHOD options except SUR.
hausman/nohausman
If you do METHOD=RANDOM, the HAUSMAN option requests that a Hausman test (for random vs fixed effects) be done. Computing this requires the fixed effects estimate. Because VCOMP=WK or VCOMP=SA each do a fixed effects regression anyway, if you choose one of those, PREGRESS will do the Hausman test. This option is necessary only if you want the Hausman test and you are using a different choice for the component variances.

robusterrors/ [norobusterrors]
cluster=SERIES with category values for clustered calculation
The combination of ROBUSTERRORS and CLUSTER allows the calculation of coefficient standard errors which are robust to arbitrary correlation within groups defined by the CLUSTER expression. This can be applied to any choice of METHOD except SUR. CLUSTER=INDIV(T) (or LWINDOW=PANEL) would be used for standard errors clustered by individuals.

[print]/noprint
vcv/ [novcv]
smpl=SMPL series or formula ("SMPL Option" on page RM–546)
unravel/ [nounravel] (User’s Guide, Section 2.10)
define= equation to define (User’s Guide, Section 1.5.4)
fml= formula to define
equation= equation to estimate
dfc= Degrees of Freedom Correction (Additional Topics, Section 1.4)
title= title to identify estimation method [depends upon options]
See LINREG for details. If you use EQUATION, omit the supplementary card.

Variables Defined
Besides the usual regression variables (%XX, %BETA, %LOGL and %RSS, etc.) PREGRESS defines:

%VRANDOM variance of η: the random component (REAL)
%VINDIV variance of ε: the individual component (REAL)
%VTIME variance of λ: the time component (REAL)
%NGROUP number of individuals or groups (INTEGER)
%SIGMA covariance matrix for METHOD=SUR (SYMMETRIC)
%NFREE number of freely estimated parameters. This includes any fixed effects coefficients that aren't reported directly and variance or covariance matrix parameters (INTEGER)

Technical Information
The different choices for VCOMP use different quadratic forms in residuals to estimate the component variances. VCOMP=WK (Wansbeek-Kapteyn) estimates fixed effects and uses its residuals several ways. VCOMP=SA (Swamy-Arora) uses residuals from fixed effects and “between” estimators. VCOMP=WH (Wallace-Hussain) uses the OLS residuals several ways. See Baltagi (2008) for more detail.
To demonstrate how the \texttt{CORRECTION} option works, we’ll look at part of the calculation for \texttt{VCOMP=WH}. The OLS residuals will be \((I - XX')^{-1}X'u\) where \(u\) is the vector of true residuals. The expected value of the sum of squared residuals will be

\[
E'\big(I - XX'^{-1}X'\big)\big(I - XX'^{-1}X'\big)u = \text{trace}(Euu') \big(I - XX'^{-1}X'\big)
\]

\text{trace}(Euu') will be a matrix with elements which are linear combinations of the component variances. \texttt{CORRECTION=NONE} ignores the \(X\) matrices (in effect, acting as if the OLS residuals are the true residuals) and uses as one of its conditions that the sum of squared OLS residuals is equal to \text{trace}(Euu'). \texttt{CORRECTION=DEGREES} uses a correction based on the number of regressors in \(X\) rather than their specific values. \texttt{CORRECTION=FULL} computes the exact correction using the structure of \text{trace}(Euu') and \(X\). (This can be rearranged using the properties of the trace to avoid multiplying the potentially very large matrices in (3)).

\begin{itemize}
    \item \texttt{preg (method=between) invest}
    \# constant value cap
    \item \texttt{preg (method=fixed) invest}
    \# constant value cap
    \item \texttt{preg (method=random,vcomp=wh) invest}
    \# constant value cap
    \item \texttt{preg (method=random,vcomp=wk) invest}
    \# constant value cap
    \item \texttt{preg (method=random,vcomp=sa) invest}
    \# constant value cap
    \item \texttt{preg (method=random,vcomp=ml) invest}
    \# constant value cap
\end{itemize}

This estimates an equation, allowing for individual effects only, using the between estimator, fixed and random effects, done with several variance components estimators. Note that we left the \texttt{CONSTANT} in the fixed effects equation. This will show a zero coefficient with zero standard error.

\begin{itemize}
    \item \texttt{preg vfrall}
    \# beertax
    \item \texttt{preg (effects=both) vfrall}
    \# beertax
\end{itemize}

This estimates an equation (by fixed effects, which is the default), 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

PRINT — Display Data Series

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.

\[
\text{print( options ) } \quad \text{start} \quad \text{end} \quad \text{list of series}
\]

Wizard

You can also view series by using the View—Series Window operation to open a Series Window, selecting the series you want to view, and selecting View—Data Table or clicking on the View Data toolbar icon: 

Parameters

\[
\text{start} \quad \text{end} \quad \text{Range of entries to print. If you have not set a SMPL, PRINT uses the smallest range required to show all defined data for the series. Any undefined data are treated as missing values.}
\]

\[
\text{list of series} \quad \text{The list of series to print. If you omit the list, all current series are printed.}
\]

Options

\[
\text{[dates]/nodates}
\]

By default, PRINT labels entries with their dates if you have set a CALENDAR. Use NODATES to get just the entry number instead.

\[
\text{number=labeling number for first entry}
\]

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.

\[
\text{picture=picture code for data}
\]

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 code 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 needed to the left. For details, see "Picture Codes" on page RM–545.
smpl=SMPL series or formula ("SMPL Option" on page RM–546)
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. (If you use the WINDOW option, however, they will always be in a single block).

Missing Values
Missing values and data outside the range of a series display 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.

```plaintext
prj( options )   series   start   end
```

**Wizard**

You can use the Time Series—Single-Equation Forecasts wizard to forecast univariate models, including static forecasts as produced by **PRJ**, though it will produce a **UFORECAST** instruction to do that.

**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 12.3).

**Fitted Values**

**PRJ** takes the coefficients ($\beta$) and the regressors ($x$) from the most recent regression and computes the fitted values ($x_\beta$) over entries **start** to **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 \texttt{AR1} and \texttt{BOXJENK} because the covariance matrix for those is from a restricted non-linear model, so these formulas won’t apply.

\textbf{stderr=} series for standard errors of projection

This computes the series of standard errors of projection: \( s \sqrt{1 + x_i (X'X)^{-1} x_i'} \)

\textbf{xvx=} series for variance of fitted values

This computes the series of (unscaled) variances (leverage statistics) for the in-sample fitted values. These are useful in various diagnostic tests. The formula is: \( x_i (X'X)^{-1} x_i' \)

\textbf{resids=} series for residuals

This computes and saves the residuals, as the actual dependent variable values minus the fitted values computed by \texttt{PRJ}.

\textbf{coeffs=} VECTOR of coefficient values to use

You can use this option to compute the fitted values based on the supplied coefficients, rather than the coefficients from the original regression.

Examples with Fitted Values

\begin{verbatim}
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 \texttt{PRJ} computes fitted values over 1923:1 to 1941:1. The \texttt{SCATTER} instruction does an actual vs. fitted plot. The second \texttt{PRJ} forecasts \texttt{FOODPROD} over the period 1942:1 to 1945:1 using the four input values for \texttt{AVGPRICE}.

linreg employ 1947:1 1961:1 resid
# constant year price gnp armed
prj(xvx=px)
set stdresids = resid/sqrt(%seesq*(1-px))
graph(style=symbols,vlabel="Standardized Residual")
# stdresids
\end{verbatim}

This uses \texttt{PRJ} with the \texttt{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:

$$z_i = \begin{cases} 
\text{Bottom truncation} & \left( X_i \hat{\beta} - T_i \right)/\sigma \\
\text{Top truncation} & \left( T_i - X_i \hat{\beta} \right)/\sigma 
\end{cases}$$

**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** ("SMPL Option" on page RM–546)
- 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.
tion 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 $x_i$ [unused]
```

The `ATMEAN` and `XVECTOR` options allow you to compute the index, density, standard error, and predicted probability for a single input set of X's. The values are returned as the variables `%PRJFIT`, `%PRJDENSITY`, `%PRJSTDERR` 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.

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 the `MCOV` instruction 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><code>%MEAN</code></td>
<td>vector of means of the explanatory variables (<code>VECTOR</code>)</td>
</tr>
<tr>
<td><code>%PRJCDF</code></td>
<td>predicted probability produced by <code>XVECTOR</code> or <code>ATMEAN</code> (<code>REAL</code>)</td>
</tr>
<tr>
<td><code>%PRJDENSITY</code></td>
<td>density produced by the <code>XVECTOR</code> or <code>ATMEAN</code> options (<code>REAL</code>)</td>
</tr>
<tr>
<td><code>%PRJFIT</code></td>
<td>fitted value produced by the <code>XVECTOR</code> or <code>ATMEAN</code> options (<code>REAL</code>)</td>
</tr>
<tr>
<td><code>%PRJSTDERR</code></td>
<td>standard error of prediction produced by the <code>XVECTOR</code> or <code>ATMEAN</code> options (<code>REAL</code>)</td>
</tr>
</tbody>
</table>

**Example with Distribution Statistics**

This estimates a “Tobit II” model, using the Mills ratio from a preliminary probit model as a regressor in a secondary linear regression.

```
  ddv(noprint) choice1
  # constant age nadults nkids nkids2 lnx $
  agelnx nadlnx bluecol whitecol
  prj(mills=lambda)
  linreg(smpl=share1>0,title="Tobit II") share1
  # constant age nadults nkids nkids2 lnx agelnx nadlnx lambda
```
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. Use TYPE to use other RATS Data types or to pass by address.

Description

The most powerful compiler structure in RATS is the procedure. A procedure is similar to subroutines of FORTRAN, the functions of C or the procedures of Java. 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 15 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)

```plaintext
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

    DISTRIBUT 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. It’s most useful to name is `procname.src` which will allow RATS to locate it automatically when you use it.

If you have a procedure on a separate file, bring it into your current RATS program using the instruction `SOURCE`. The typical instruction is

```plaintext
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:

```plaintext
@cumpdgm x 1980:1 2017:12
```

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.

```
prtdata( options )  list of data file series (optional)
```

**Wizard**

If you open a RATS format file using File—Open..., 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, it will print all the series on the file.

**Options**

- `format=[portable]/binary/cdf/dbf/dif/free/html/prn/rats/rtf/tex/tds/wks/xls/xlsx/"(FORTRAN format)"
  
  This selects the desired format for the output. The “spreadsheet” formats (XLS, XLSX, WKS, RTF, DBF, PRN, CDF, DIF, HTML and TEX) 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, XLSX, WKS, RTF, PRN, CDF, DIF, HTML and TEX, 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 2.8 in the Introduction.

- `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 code for data
  
  This allows you control the formatting of the numeric values in the output in PRN, CDF, RTF, DIF, HTML, TEX and FREE formats. (Actual spreadsheets like XLS always use the full precision). A picture code 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 "Picture Codes" on page RM–545 for more information.
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.

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,
   SALEY45 and SALEY66.

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 12.5 and 12.6 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**

- **effects=[individual]/time/both**
  This indicates whether to allow for **INDIVIDUAL** effects, **TIME** effects or **BOTH**.

- **smpl=SMPL series**
  This is the standard **SMPL** option ("SMPL Option" on page RM–546).

- **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.

- **group=SERIES or FRML with values defining individuals**
  This is an alternative to a panel data setup for data. This defines the individuals. If you use **GROUP**, you can’t do any calculations which require identifying specific time periods (**EFFECTS=TIME** or **EFFECTS=BOTH**).
Missing Values

RATS drops missing values from the computation.

Variables Defined

- \%NGROUP: number of individuals/groups that actually contain data (omits individuals where all time periods are empty) (INTEGER)
- \%VRANDOM: variance of \( \eta \): the random component (REAL)
- \%VINDIV: variance of \( \epsilon \): the individual component (REAL)
- \%VTIME: variance of \( \mu \): the time component (REAL)

Example

This is a variation on the code presented in the PANEL.RPF example program.

```
    cal(panelobs=20) 1935
    all 10//1954:1
    open data grunfeld.dat
    data(format=prn,org=cols)
    linreg invest
    # firmvalue cstock
    pstats(tests,effects=both) %resids
    preg(method=fixed) invest
    # firmvalue cstock
    pstats(tests,effects=time) %resids
    pstats(spread) %resids
```

The first PSTATS tests the residuals from an OLS regression for time and individual effects, the second tests the residuals from a fixed effects regression for time effects. The third PSTATS tests for equal variances.

Output

The output from the second and third PSTATS instructions above is:

```
Analysis of Variance for Series %RESIDS
Source Sum of Squares  Degrees  Mean Square   F-Statistic  Signif Level
TIME    55556.59368270      19   2924.03124646        1.125     0.329613
ERROR  467921.55370355     180   2599.56418724
TOTAL  523478.14738625     199

Test of Equal Variances for Series %RESIDS
Chi-Squared(9)=    281.229443 with Significance Level 0.00000000
```

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.
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.

```
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=** "string" or STRING variable ["Enter Information"]
  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.

- **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 valid input [none]
  You can use VERIFY to 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/[noinitilize]**
  If INITIALIZE, the text box is filled with the current value of the variable. This only works if there is only one 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.

```plaintext
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.

```plaintext
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.

```plaintext
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

\[ \text{qz ( options ) } \quad A \quad B \]

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 \(QAZ' = \Lambda\) 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 \text{BLOCK} and \text{CUTOFF} options. The \text{SIZE} option allows you to find out how big the “upper” block is.

Options

\text{block=below/above/real/imag}

Indicates the criterion used for determining which eigenvalues go into the upper block. \text{BELOW} and \text{ABOVE} are based upon absolute values, and use the \text{CUTOFF} value. \text{BLOCK=BELOW, CUTOFF=1.0} will put all eigenvalues with absolute value less than 1.0 in the upper block. \text{REAL} and \text{IMAG} partition the eigenvalues into real and complex, moving the indicated group into the upper block.

\text{cutoff=value or formula giving the cutoff value [1.0 by default]}

This supplies the cutoff value for \text{BLOCK=BELOW} or \text{BLOCK=ABOVE}.

The following are used to provide names for the variables computed by \text{QZ}:

\text{q=(output) } Q \text{ matrix}
\text{z=(output) } Z \text{ matrix}
\text{lambda=(output) } \Lambda \text{ matrix}
\text{omega=(output) } \Omega \text{ matrix}
\text{cvalues=(output) VECTOR[COMPLEX] of generalized eigenvalues}
\text{evalues=(output) VECTOR of real parts of generalized eigenvalues}
\text{size=(output) size of the upper block}

Example

\[ \text{qz(q=q,z=z,lambda=lambda,omega=omega,block=below,}$
\quad \text{cutoff=1.0/beta) g0 g1} \]

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 page UG–212.

```
ratio(designes=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 to 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 as 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.

```
mcorr=multiplier correction [0]
```
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.

```
smpl=SMPL series or formula ("SMPL Option" on page RM–546)
```
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.

```
[print]/noprint
```

```
title="string for output title"
```
Use NOPRINT to suppress the printing of the test results. If you are showing the output, you can use the TITLE option to provide your own title for the output.
spread=standard SPREAD option ("SPREAD Option" on page RM–547)
weight=series of entry weights ("WEIGHT option" on page RM–549)
The residuals which are generated by the regression commands are "unweighted." So, if you used a SPREAD or WEIGHT option to compute residuals, you should use the same option on RATIO.

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
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
%CDSTAT the computed test statistic (REAL)
%SIGNIF the marginal significance level (REAL)
%NOBS number of observations (INTEGER)
%NVAR number of variables (INTEGER)

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. You can use arrays of arrays, but any arrays must be dimensioned ahead of time (unless you use the option VARYING, or are reading from a MATLAB file).

Options

format=[free]/binary/cdf/matlab/prn/tsd/wks/xls/xlsx/

“( FORTRAN format )”

This tells READ the format of the data. See below for information on how the format option affects the way that READ fills arrays.

sheet=“worksheet or matrix name”

When reading an Excel file with multiple worksheets or a MATLAB file with several data series on a single matrix, you can use SHEET to identify the which you want to read. READ reads the first by default.

unit=input/[data]/other unit

READ reads the data from the specified I/O unit. By default, this is the DATA unit.

top=top line to process [1]
bottom=top line to process [last]
left=leftmost column to process [1]
right=rightmost column to process [last]

You can use these if you want to ignore text above, below, to the left, or to the right of the data you want to read.

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** or **PACKED** arrays by the rows of the lower triangle. It can read more than one row from a single line.

- With **FORMAT=CDF, TSD, PRN, WKS, XLS, and XLSX**, **READ** fills cells in the target variables based on the arrangement of the data on the file. If you don’t have enough values in a row, the remaining elements get missing value codes.

- With **FORMAT=MATLAB**, **READ** will read (and dimension) only full arrays which exist with the same name on the file.

- With **FORMAT=BINARY**, **READ** processes arrays in their internal order. Arrays which are **RECTANGULAR** are read by columns, **SYMMETRIC** or **PACKED** by the rows of the lower triangle.

- With a **FORTRAN** format (**Additional Topics**, Section 3.9), **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** or **PACKED** arrays by the rows of the lower triangle. A **VECTOR** is treated like a single row.

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
open data arrayin.mat
read(format=matlab) a
```

This reads the array **A** from a MATLAB file. It will take its dimensions from the matrix **A** on the file.
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.

**Example with VARYING**

```plaintext
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, though with modern computers, these are generally not large enough to matter. 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 CMO- MENT.

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.
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.

\begin{verbatim}
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)
\end{verbatim}

Parameters

variables/expressions

With ACTION=MODIFY, this list of variables and expressions provides the information which is to be inserted into the report.

Options

\begin{verbatim}
action=define/[modify]/format/show/sort
\end{verbatim}

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.

\begin{verbatim}
use=name of report
\end{verbatim}

USE allows you to define a new report object (when used with ACTION=DEFINE), or work with an existing report object (when used with the other choices for the ACTION option: MODIFY, FORMAT, SHOW, and SORT). If you omit the USE option, RATS uses the default internal 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

**title=**"title for report" ["User Report"]

This option allows you to supply your own descriptive title for the report. This is used to identify it in the Window-Restore Reports menu operation.

**hlabels=VECTOR of STRINGS to provide the column (header) 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.

**vlabels=VECTOR of STRINGS to provide row labels**

**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/threestars/parens/brackets
   You can use SPECIAL to enclose cells in parentheses (), brackets [], or tag them with one star (*), two stars (**), or three stars (***)..

prefix="prefix formatting string" [not used]
suffix="suffix formatting string" [not used]
   These can be used as an alternative to the SPECIAL option if you need some other form of tagging (for eventual use in TeX). You can use these alone or together. For instance, SUFFIX="^**" will add ** as a superscript (ordinarily ** is on the same line at normal text size), and PREFIX="\lbrace" combined with SUFFIX="\rbrace" will enclose the element in {}.

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 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 “Picture Codes” on page RM–545 for more information), 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.

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.

tag=maximum/minimum

   special=[none]/onestar/twostars/threestars/parens/brackets
   TAG identifies the maximum or minimum value in the range specified by the ATROW, TOROW, ATCOL, and TOCOL options. SPECIAL determines how the cell will be identified: enclosed in parentheses or brackets, or tagged with one star (*), two stars (**), or three stars (***).. For example, the following “stars” the maximum value in the first column:

   report(act=format,atcol=1,tocol=1,tag=max,special=onestar)
These set the range of rows and columns over which the PICTURE and WIDTH formatting will be applied, or over which TAG will search to identify a maximum or minimum.

Options Used With ACTION=SORT

bycol=column number
This sorts the report by the values in the selected column.

atrow=starting row number
torow=ending row number
You can use ATROW and TOROW to limit the sorting to a specific range of rows.

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=cdf/dbf/dif/html/prn/rtf/tex/wks/xls/xlsx/xml [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

Below (and continued on the next page) is an excerpt from example 10.3 from Verbeek (2008). This uses the REGRESS option to build a report presenting the coefficients and standard errors from four different regressions (three panel data regressions and one OLS estimation).

Define the report and provide column headers:

report(action=define,hlabels=||"Variable","Between","Fixed Effects","OLS","Random Effects"||)

Perform the regressions, following each with REPORT(REGRESS) to add the results to the report:
preg(method=between) wage
# constant school exper expersq union mar black hisp pub
report(regress)

preg(method=fixed) wage
# exper expersq union mar pub
report(regress)

linreg wage
# constant school exper expersq union mar black hisp pub
report(regress)

preg(method=random,vindiv=.1055,vrand=.1234) wage
# constant school exper expersq union mar black hisp pub
report(regress)

Format the cells to use 3 decimals places and display the table:

report(action=format,picture="*.###")
report(action=show)

Here's the resulting output:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Between Fixed Effects</th>
<th>OLS</th>
<th>Random Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>0.490</td>
<td>-0.034</td>
<td>-0.104</td>
</tr>
<tr>
<td></td>
<td>(0.078)</td>
<td>(0.065)</td>
<td>(0.111)</td>
</tr>
<tr>
<td>SCHOOL</td>
<td>0.095</td>
<td>0.099</td>
<td>0.101</td>
</tr>
<tr>
<td></td>
<td>(0.004)</td>
<td>(0.005)</td>
<td>(0.009)</td>
</tr>
<tr>
<td>EXPER</td>
<td>-0.050</td>
<td>0.116</td>
<td>0.112</td>
</tr>
<tr>
<td></td>
<td>(0.018)</td>
<td>(0.008)</td>
<td>(0.010)</td>
</tr>
<tr>
<td>EXPERSQ</td>
<td>0.005</td>
<td>-0.004</td>
<td>-0.004</td>
</tr>
<tr>
<td></td>
<td>(0.001)</td>
<td>(0.001)</td>
<td>(0.001)</td>
</tr>
<tr>
<td>UNION</td>
<td>0.274</td>
<td>0.081</td>
<td>0.106</td>
</tr>
<tr>
<td></td>
<td>(0.017)</td>
<td>(0.019)</td>
<td>(0.017)</td>
</tr>
<tr>
<td>MAR</td>
<td>0.145</td>
<td>0.045</td>
<td>0.063</td>
</tr>
<tr>
<td></td>
<td>(0.014)</td>
<td>(0.018)</td>
<td>(0.016)</td>
</tr>
<tr>
<td>BLACK</td>
<td>-0.139</td>
<td>-0.144</td>
<td>-0.144</td>
</tr>
<tr>
<td></td>
<td>(0.017)</td>
<td>(0.024)</td>
<td>(0.048)</td>
</tr>
<tr>
<td>HISP</td>
<td>0.005</td>
<td>0.016</td>
<td>0.020</td>
</tr>
<tr>
<td></td>
<td>(0.015)</td>
<td>(0.021)</td>
<td>(0.043)</td>
</tr>
<tr>
<td>PUB</td>
<td>-0.056</td>
<td>0.035</td>
<td>0.030</td>
</tr>
<tr>
<td></td>
<td>(0.038)</td>
<td>(0.039)</td>
<td>(0.037)</td>
</tr>
</tbody>
</table>
The next set of code is from example 2.7 from Verbeek (2008):

```
cal(m) 1960:1
open data capm2.dat
data(format=prn,org=columns) 1960:1 2002:12 $
   rfood rdur rcon rmrf rf jandum

Define a report, and provide a set of row labels for the first column:

```
report(action=define)
report(atrow=1,fillby=cols) $
   "Company" "Excess Returns" '"" Uncentered R^2" "s"

Now do the first regression and add four numerical results to the report.
COL=NEW puts these in a new column. ATROW=1 starts the information in row one. FILLBY adds items going down the column, rather than across in rows:

```
linreg rfood
# rmrf
report(col=new,atrow=1,fillby=cols,align=center) $
   "Food" %beta(1) %stderrs(1) %trsq/%nobs sqrt(%seesq)

Repeat process for two more regressions

```
linreg rdur
# rmrf
report(col=new,atrow=1,fillby=cols) $
   "Durables" %beta(1) %stderrs(1) %trsq/%nobs sqrt(%seesq)

linreg rcon
# rmrf
report(col=new,atrow=1,fillby=cols) $
   "Construction" %beta(1) %stderrs(1) %trsq/%nobs sqrt(%seesq)

Generate the report:

```
report(action=show)

And here is the output:

<table>
<thead>
<tr>
<th>Company</th>
<th>Food</th>
<th>Durables</th>
<th>Constuction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excess Returns</td>
<td>0.790380</td>
<td>1.112622</td>
<td>1.156058</td>
</tr>
<tr>
<td></td>
<td>0.028397</td>
<td>0.028953</td>
<td>0.025148</td>
</tr>
<tr>
<td>Uncentered R^2</td>
<td>0.600674</td>
<td>0.741429</td>
<td>0.804054</td>
</tr>
<tr>
<td>s</td>
<td>2.902180</td>
<td>2.959008</td>
<td>2.570095</td>
</tr>
</tbody>
</table>
Restrict

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
# list of coefficients entering the restriction
# coefficient weights followed by restricted value
```

**Wizard**

Use [Statistics—Regression Tests](#) and select “General Linear Restrictions”.

**Parameters**

- **restrictions**  
The number of linear restrictions.
- **residuals**  
(Optional) With the **CREATE** option only, this is a series for the residuals from the restricted regression. Note that the standard %RESIDS series is set to the residuals as well, so you will rarely need this.

**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
\]

\[
\beta_1 - 2\beta_2 + \beta_5 = 0.0\
\]

```

```
# 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 parameter 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 GARCH, DDV and similar instructions which do maximum likelihood estimation 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:

form=\text{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 page UG–65 in the User's Guide.

print/\text{[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.

title="\text{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 `RESTRICT` 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 3.3 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 for doing restricted regressions:

- `create/[nocreate]`
- `replace/[noreplace]`

`CREATE` does the restricted regression. `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 `RESTRICT(CREATE)` can only print single equations.

- `[print]/noprint`
- `vcv/[novecv]`

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.
**Restrict**

**form=f/chisquared**
Described earlier—it affects only the test statistic, not the restricted regression.

**coeff=(output) VECTOR for restricted coefficients**

**covmat=(output) 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:

\[ y_t = \beta_0 + \beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t} + u_t \]

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

\[
\log(C/Q) = \alpha + \sum_i \alpha_i \log w_i + \sum_j \sum_{j<i} \gamma_{ij} (\log w_i)(\log w_j)
\]

and tests (jointly) the following combination of restrictions

\[
\alpha_1 + \alpha_2 = 1.0
\]

\[
\gamma_{11} + 0.5 \gamma_{12} = 0.0
\]

\[
\gamma_{22} + 0.5 \gamma_{12} = 0.0
\]

```
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

Variables Defined

%CDSTAT   the computed test statistic (REAL).
%SIGNIF    the marginal significance level (REAL).
%NDFTEST   (numerator) degrees of freedom for the test (INTEGER)

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 page UG–62 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. For a procedure, execution continues with the instruction following the EXECUTE that invoked the procedure. For a FUNCTION, the calculation in which the function was invoked is continued.

RATS automatically puts a return at the end of a PROCEDURE or FUNCTION, so you only need RETURN if you want to exit before hitting the end of the subprogram.

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 15.2 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. Most formats, such as spreadsheets, are always read as a whole each time.

If you write to a file using (for instance) COPY and want to read it back within the same program, you need to CLOSE the file then re-open it rather than rewinding.

```
rewind RATS I/O unit
```

Parameters

`RATS I/O unit` The I/O unit to rewind.

Example

data(format=free,org=columns,missing=-88.888) * 1991:2 $
   year month notmiss regnobs stderr taxrate $
   r0m r1m r2m r3m r4m r5m r6m r7m r8m r9m r10m r11m r12m $
   r13m r14m r15m r16m r17m r18m r21m r24m r30m r36m r48m $ 
   r60m r72m r84m r96m r108m r120m r132m r144m r156m r168m $ 
   r180m r192m r204m r216m r228m r240m r252m r264m r276m $ 
   r288m r300m r312m r324m r336m r348m r360m r372m r384m $ 
   r396m r408m r420m r480m
   *
```

rewind data
data(format=free,org=columns,missing=-88.888) 1991:3 531 $
   year month notmiss regnobs stderr taxrate $
   r0m r1m r2m r3m r4m r5m r6m r7m r8m r9m r10m r11m r12m $
   r13m r14m r15m r16m r17m r18m r21m r24m r30m r36m r48m $ 
   r60m r72m r84m r96m r108m r120m r132m r144m r156m r168m $ 
   r180m r192m r204m r216m r228m r240m r252m r264m r276m $ 
   r288m r300m r312m r324m r336m r348m r360m r372m r384m $ 
   r396m r408m r420m r480m

This was for duplicating a paper which had an error in data handling. There weren’t 531 data points on the data file and the program used (which was not RATS) rewound the data file to fill the request.

See Also . . .

OPEN Opens a RATS I/O unit.
CLOSE Closes a RATS I/O unit.
RLS — Recursive Least Squares

RLS uses the Kalman filter to perform least squares regressions over a range of entries, generating various 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
```

Wizard

You can use Statistics—Recursive Least Squares operation to do recursive least squares.

Parameters

- `depvar` Dependent variable.
- `start` 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` [“Recursive Least Squares”]
  These control the printing of regression output and the printing of the estimated covariance/correlation matrix of the coefficients (Introduction, page Int–77), and the title used in labeling the output.

- `smpl=SMPL series or formula` (“SMPL Option” on page RM–546)
  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=standard SPREAD option` (“SPREAD Option” on page RM–547)
  Use SPREAD for weighted recursive least squares. The residual variances are assumed to be proportional to the indicated series.

- `weight=series of entry weights` (“WEIGHT option” on page RM–549)
  Use this option if you want to give unequal weights to the observations.
**equation** = *equation to estimate*

**lastreg**/[**nolastreg**]

Use the **EQUATION** option to estimate a previously defined equation. **LASTREG** re-estimates the most recent regression using recursive least squares. If you use either, don’t include a supplementary card.

**cohistory** = **VECTOR**[/**SERIES**] of coefficient estimates [not used]

**sehistory** = **VECTOR**[/**SERIES**] of coefficient standard errors [not used]

Respectively, these save 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:

```plaintext
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, 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 CONDITION option, in which case RLS uses the number of entries you specify). This will give a coefficient estimate $\beta_t$ and $(X'X)^{-1}$ matrix $\Sigma_t$. The residual, SIGHIST and 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. $\hat{e}_t = \frac{(y_t - X_t\beta_{t-1})}{\sqrt{1 + X_t\Sigma_{t-1}X'_t}}$

2. $\beta_t = \beta_{t-1} + \Sigma_{t-1}X'_t\left(\frac{y_t - X_t\beta_{t-1}}{1 + X_t\Sigma_{t-1}X'_t}\right)$

3. $\Sigma_t = \Sigma_{t-1} - \frac{\Sigma_{t-1}X'_tX_t\Sigma_{t-1}}{(1 + X_t\Sigma_{t-1}X'_t)}$

where $\hat{e}_t$ is the recursive residual at $t$. The estimated variance of the regression through $t$ is

4. $\sigma_t^2 = \sum_{s=T_{t+1}}^{t} \hat{e}_s^2/(t-T_t)$

and the standard errors of the coefficient estimates are square roots of the diagonal elements of $\sigma_t^2\Sigma_t$.

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 JOHN4P121.RPF.

*The COHIST and SEHIST options provide a VECTOR[SERIES] with the “histories” of the coefficients and the standard errors of the coefficient estimates, while SIGHIST gives the standard errors of the regression. The CSQUARED option returns the sum of squared recursive residuals, which will also be the sequence of sums of squared residuals from the regressions.*

```r
rls(sehist=sehist,cohist=cohist,sighist=sighist, $ 
  dfhistory=dfhist,csquared=cusumsq) y 1959:1 1973:3 rresids
# constant x2 x3
```

```r
set lower = -2*sighist
set upper = 2*sighist
graph(header="Recursive Resids and S.E. Bands for Gasoline") 3
  # rresids
  # lower
  # upper / 2
```
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||)
# seqfcval
```

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 residu-
als. Note that it doesn’t have the distribution that a \( \text{dw} \) would have when applied to
regular regression residuals, though it will still be near 2 if the residuals aren’t seri-
ally correlated.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BETA</td>
<td>Coefficient vector (VETTOR)</td>
</tr>
<tr>
<td>%XX</td>
<td>((X'X)^{-1}) matrix (SYMMETRIC)</td>
</tr>
<tr>
<td>%TSTATS</td>
<td>Vector containing the ( t )-stats for the coefficients (VECTOR)</td>
</tr>
<tr>
<td>%STDERRS</td>
<td>Vector of coefficient standard errors (VECTOR)</td>
</tr>
<tr>
<td>%MEAN</td>
<td>Mean of dependent variable (real)</td>
</tr>
<tr>
<td>%NDF</td>
<td>Degrees of freedom (INTEGER)</td>
</tr>
<tr>
<td>%NOBS</td>
<td>Number of observations (INTEGER)</td>
</tr>
<tr>
<td>%NREG</td>
<td>Number of regressors (INTEGER)</td>
</tr>
<tr>
<td>%NFREE</td>
<td>Number of free parameters (INTEGER)</td>
</tr>
<tr>
<td>%RSQUARED</td>
<td>Centered ( R^2 ) (REAL)</td>
</tr>
<tr>
<td>%RBARSQ</td>
<td>Adjusted ( R^2 ) (REAL)</td>
</tr>
<tr>
<td>%TRSQ</td>
<td>No. of observations times raw ( R^2 ) (REAL)</td>
</tr>
<tr>
<td>%DURBIN</td>
<td>Durbin-Watson statistic (REAL)</td>
</tr>
<tr>
<td>%RSS</td>
<td>Residual sum of squares (REAL)</td>
</tr>
<tr>
<td>%SEESQ</td>
<td>Standard error of estimate squared (REAL)</td>
</tr>
<tr>
<td>%QSTAT</td>
<td>Q-statistic (REAL)</td>
</tr>
<tr>
<td>%QSIGNIF</td>
<td>Significance level of Q-statistic (REAL)</td>
</tr>
<tr>
<td>%RHO</td>
<td>First lag correlation coefficient (REAL)</td>
</tr>
<tr>
<td>%VARIANCE</td>
<td>Variance of dependent variable (REAL)</td>
</tr>
</tbody>
</table>
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.

\[
\text{rreg( options ) depvar start end residuals}
\]

# explanatory variables in regression format

Wizard

Use the Statistics—Linear Regressions wizard and select “Robust Regression” as the technique.

Parameters

- **depvar**: Dependent variable.
- **start** and **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.

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 (*Introduction*, page Int–77), and the title used in labeling the output.

- `smpl=SMPL series or formula` *(“SMPL Option” on page RM–546)*  
  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**

**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 “Technical Information” for details.

**Iter**

**iterations=** number of iterations

- Allows the user to control the number of iterations used for the linear programming algorithm. The default value depends on the number of parameters, but is generally set to 100.

**Bandwidth**

**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 “Technical Information” for a description of the default values for these.

**Last Regression**

**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 either, omit the supplementary card.

- **define=** equation to define (Introduction, page Int–53)
  - **frml=** formula to define
    - These define an equation and formula, respectively, using the results of the estimation. You can use the equation/formula for forecasting or other purposes.

- **weight=** series of entry weights (“WEIGHT option” on page RM–549)
  - Use this option if you want to give unequal weights to the observations.

---

## Technical Information

LAD chooses $\beta$ to minimize

$$
(1) \quad \sum |y_i - X_i \beta|
$$

while the quantile regression minimizes

$$
(2) \quad \sum_{y_i - X_i \beta > 0} \alpha |y_i - X_i \beta| + \sum_{y_i - X_i \beta < 0} (1 - \alpha) |y_i - X_i \beta|
$$

where $\alpha$ is the quantile requested. 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.
The functions being minimized aren’t differentiable, so $\beta$ 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 than least squares 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

$$
\frac{0.25}{f(0)^2} \text{ for } \text{LAD and}
$$

$$
\frac{\alpha(1-\alpha)}{f(x_\alpha)^2} \text{ where } x_\alpha \text{ is (an) } \alpha \text{ quantile of the residuals.}
$$

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 computes $f$ using a Gaussian kernel (see DENSITY), and bandwidth

$$
\frac{0.79\text{IQR}}{N^{1/5}}
$$

where $\text{IQR}$ 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

This is part of an example from Greene (2012). 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.

```
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
```

Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BETA</td>
<td>Coefficient vector (VECTOR)</td>
</tr>
<tr>
<td>%XX</td>
<td>Covariance matrix of coefficients, or ((X'X)^{-1}) (SYMMETRIC)</td>
</tr>
<tr>
<td>%TSTATS</td>
<td>Vector containing the (t)-stats for the coefficients (VECTOR)</td>
</tr>
<tr>
<td>%STDERRS</td>
<td>Vector of coefficient standard errors (VECTOR)</td>
</tr>
<tr>
<td>%NOBS</td>
<td>Number of observations (INTEGER)</td>
</tr>
<tr>
<td>%NREG</td>
<td>Number of regressors (INTEGER)</td>
</tr>
<tr>
<td>%NFREE</td>
<td>Number of free parameters (INTEGER)</td>
</tr>
<tr>
<td>%NDF</td>
<td>Degrees of freedom (INTEGER)</td>
</tr>
<tr>
<td>%FUNCVAL</td>
<td>Minimized values of equations (1) or (2) (REAL)</td>
</tr>
<tr>
<td>%MEAN</td>
<td>Mean of dependent variable (REAL)</td>
</tr>
<tr>
<td>%RESIDS</td>
<td>Series containing the residuals (SERIES)</td>
</tr>
<tr>
<td>%DURBIN</td>
<td>Durbin-Watson statistic (REAL)</td>
</tr>
<tr>
<td>%RHO</td>
<td>First lag correlation coefficient (REAL)</td>
</tr>
<tr>
<td>%VARIANCE</td>
<td>Variance of dependent variable (REAL)</td>
</tr>
<tr>
<td>%EBW</td>
<td>Bandwidth used in estimate of the density (REAL)</td>
</tr>
</tbody>
</table>
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 RTOC (Real TO Complex) to transfer values to a complex series. CTOR (Complex TO Real) transfers data in the other direction.

```
rtoc(options) start end newstart
# list of real series to copy
# list of complex series to copy to
```

Parameters

- **start end**: Range of entries to transfer. If you have not set a SMPL, this defaults to the defined range of each real series, determined separately for each series.

- **newstart**: Entry in the complex series for the start entry of the real series. By default, same as start. The default starting entry may change from series to series if you use the default start and end range.

Description

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 14.5 of the User’s Guide for a discussion of preparing data for frequency domain analysis.

Options

- **[pad]/nopad**: By default, RTOC 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 NOPAD option.

RTOC 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 RTOC parameters, it will transfer the defined portion of the real series and set the initial undefined observations to zero.
even/[noeven]

With `EVEN`, `RTOC` 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.

```plaintext
frequency 5 256
rtoc
#    resids1  resids2
# 1 2
```

Using Matrices

It is possible to read and write complex series with the matrix input instruction `READ`. 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`:

```plaintext
freq 5 128
open data freqs.dat
declare vector[complex] cseries
overlay %z(1,2) with cseries(128)
read cseries
```

Variables Defined

```plaintext
%NOBS  Number of observations transferred (taken from the last series on the list) (INTEGER)
```
SAMPLE — Extracting Data and Changing Frequencies

SAMPLE creates one series by selecting entries from another one either at regular intervals, or under control of a dummy variable (or expression).

```
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 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 page Int–104 in the Introduction.

- **smpl=SMPL series or formula**  (“SMPL Option” on page RM–546)

  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

```
set filter_missing = %valid(sp500)
sample(smp1=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**.

```
calendar(q) 1980:1
all 2010:4
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.

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 16.2 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 programmatically (that is, starting with DEDIT). 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, adds three series to it, then saves the changes to the file.

See Also . . .

Intro, Section 2.7 RATS format data 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  stylenum
```

**Wizard**

Use *Data/Graphics—Scatter (X-Y) Graph*. Note that in order to keep the Wizard from getting too complicated, some of the `SCATTER` options have been omitted in the wizard. To further customize your graph, you can edit the `SCATTER` instruction generated by the wizard.

**Positioning**

If you’re using `SPGRAPH` to put multiple graphs on a single page, by default, the fields are filled by column, starting at the top left (field 1,1). If you want to fill a particular field instead, use either the combination of `ROW` and `COL` options or `hfield` (for the column) and `vfield` (for the row) parameters.

**Parameters**

- **pairs**
  Number of pairs of series to plot against each other.

- **hfield**  **vfield**
  See “Positioning” above.

**Supplementary Cards (One for Each Pair of Series)**

- **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.*

- **stylenum**
  (Optional) This lets you use an integer number between 1 and 30 to select the style (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. A style number over 30 is “wrapped” to a number in range (41 is the same as 11).

  Normally, you can just omit this parameter. RATS will automatically assign a different color or pattern to each pair.
## Scatter Options

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>none/vertical/horizontal/both</td>
<td>Draw x=0 and/or y=0 axes</td>
</tr>
<tr>
<td>extend</td>
<td>[none]/vertical/horizontal/both</td>
<td>Extend grid lines across graph</td>
</tr>
<tr>
<td>hgrid</td>
<td>VECTOR of grid values</td>
<td>Sets position of grid lines</td>
</tr>
<tr>
<td>hlabel</td>
<td>horizontal scale label</td>
<td>Adds a label to the x-axis</td>
</tr>
<tr>
<td>hlog</td>
<td>base for a log scale</td>
<td>Selects a log scale for x-axis</td>
</tr>
<tr>
<td>hmax</td>
<td>value for right boundary</td>
<td>Sets the maximum x-axis value</td>
</tr>
<tr>
<td>hmin</td>
<td>value for left boundary</td>
<td>Sets the minimum x-axis value</td>
</tr>
<tr>
<td>hpicture</td>
<td>picture code for x-axis</td>
<td>Format of x-axis scale values</td>
</tr>
<tr>
<td>hscale</td>
<td>[lower]/upper/both/none</td>
<td>Placement of x-axis scale</td>
</tr>
<tr>
<td>hshade</td>
<td>RECTANGULAR with shading zones</td>
<td>Shading zones for x-axis</td>
</tr>
<tr>
<td>hticks</td>
<td>max number of horizontal ticks</td>
<td>Number of tick marks on x-axis</td>
</tr>
<tr>
<td>lines</td>
<td>RECTANGULAR with intercept/slope</td>
<td>Draw lines given slope/intercept</td>
</tr>
<tr>
<td>omax</td>
<td>max value for overlay scale</td>
<td>Sets max value of overlay scale</td>
</tr>
<tr>
<td>omin</td>
<td>min value for overlay scale</td>
<td>Sets min value of overlay scale</td>
</tr>
<tr>
<td>ovcount</td>
<td>number of series for overlay</td>
<td># of series using overlay scale</td>
</tr>
<tr>
<td>overlay</td>
<td>dots/symbols/line/bar/poly/filled/spike/step</td>
<td>Style used for overlay series</td>
</tr>
<tr>
<td>ovkey</td>
<td>[nooverkey]</td>
<td>Adds a key for overlay series</td>
</tr>
<tr>
<td>ovlabel</td>
<td>label for the overlay scale</td>
<td>Label for the overlay scale</td>
</tr>
<tr>
<td>ovsmescale</td>
<td>[noovsmescale]</td>
<td>Same scale for regular &amp; overlay</td>
</tr>
<tr>
<td>style</td>
<td>dots/[symbols]/line/bar/poly/filled/spike/step</td>
<td>Style of graph</td>
</tr>
<tr>
<td>vgrid</td>
<td>vector of grid line values</td>
<td>Sets grid for vertical axis</td>
</tr>
<tr>
<td>vlabel</td>
<td>vertical scale label</td>
<td>Label for vertical axis</td>
</tr>
<tr>
<td>vlog</td>
<td>base for a log scale for y-axis</td>
<td>Selects a log scale for y-axis</td>
</tr>
<tr>
<td>vmax</td>
<td>value for upper boundary</td>
<td>Sets maximum y-axis value</td>
</tr>
<tr>
<td>vmin</td>
<td>value for lower boundary</td>
<td>Sets minimum y-axis value</td>
</tr>
<tr>
<td>vpicture</td>
<td>picture code for y-axis</td>
<td>Formatting of y-axis scale values</td>
</tr>
<tr>
<td>vscale</td>
<td>[left]/right/both/none</td>
<td>Placement of vertical scale</td>
</tr>
<tr>
<td>vshade</td>
<td>RECTANGULAR with shading zones</td>
<td>Shading zones for y-axis</td>
</tr>
<tr>
<td>vticks</td>
<td>max number of vertical ticks</td>
<td>Number of tick marks on y-axis</td>
</tr>
<tr>
<td>xlabels</td>
<td>VECT[STRING] for x-axis labels</td>
<td>Strings for labeling x-axis</td>
</tr>
</tbody>
</table>

### Options Common to GRAPH and SCATTER

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[box]</td>
<td>nobox</td>
<td>Superseded by <code>FRAME</code> option</td>
</tr>
<tr>
<td>col</td>
<td>column number</td>
<td>Column in the <code>SPGRAPH</code> matrix</td>
</tr>
<tr>
<td>footer</td>
<td>footer label</td>
<td>Adds a footer label below graph</td>
</tr>
<tr>
<td>frame</td>
<td>[full]/half/none/bottom</td>
<td>Controls frame label around the graph</td>
</tr>
</tbody>
</table>
Scatter

header=header string for graph  
Adds a header to the graph
height=height in inches  
Sets height of graph
[kbox]/nokbox
[draws a box around the key
key=[none]/upleft/upright/loleft/
  lortight/above/below/left/right  
Location of Key
kheight=fraction of graph height  
Sets height of the key box
klabel=VECTOR of LABELS or STRINGS  
Use to supply custom key labels
kwidth=fraction of overall graph width  
Sets width of the key box
[ksample]/noksample
patterns/[nopatterns]
row=row number  
Row in the SPGRAPH matrix
smpl=series or formula for smpl  
Selects subset of entries to graph
subhead=subheader string  
Adds a sub-header to the graph
width=width in inches  
Sets width of graph
window=string for window title  
Custom title for graph window

General 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 increas
  ing order. A bar graph draws a filled rectangle centered at each x value to its
  corresponding y value. POLYGON 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. Or
dinarily, 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.

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.
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 BOTH, or centered at the top above the tick marks if you use HSCALE=UPPER. The VLABEL is centered at the left for VSATE=LEFT or NONE, and centered at the right for VSATE=RIGHT. It appears on both sides with VSATE=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.
\textbf{hticks=maximum horizontal ticks or VECTOR with specific values [7]}

\textbf{vticks=maximum vertical ticks or VECTOR with specific values [9]}

These set the maximum number of labeled tick marks on the horizontal scale and vertical scales, respectively. With either, you can provide a \texttt{VECTOR} to override the automatic assignment.

\textbf{hpicture=picture code for horizontal axis [shortest that works]}

\textbf{vpicture=picture code 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, \texttt{VPicture="*##"} will show the numbers on the vertical axis with two digits right of the decimal. By default, \texttt{SCATTER} chooses the shortest representation that can show all values accurately. See “Picture Codes” on page RM–545 for more information.

\textbf{extend=[none]/both/vertical/horizontal}

Normally, \texttt{RATS} marks the vertical and horizontal axis with small tick marks outside the graph. You can use the \texttt{EXTEND} option to have \texttt{RATS} draw dotted grid lines from each tick mark all the way across or down the graph. \texttt{HORIZONTAL} draws horizontal grid lines, \texttt{VERTICAL} draws vertical grid lines, and \texttt{BOTH} draws horizontal and vertical grid lines.

\textbf{hgrid=VECTOR of grid line values for the horizontal axis [unused]}

\textbf{vgrid=VECTOR of grid line values for the vertical axis [unused]}

\textbf{hshade=RECTANGULAR with shading zones for horizontal axis [unused]}

\textbf{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 \texttt{VECTOR} with the x (y for \texttt{VGRID}) values at which a vertical (horizontal) line will be placed. The “shade” options take a \texttt{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 \texttt{HSHADE}) or side to side (for \texttt{VSHADE}).

\section*{Two-Scale Graph Options}

\textbf{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 \texttt{STYLE} option for a description of the choices). The \texttt{OVCOUNT} option tells how many of the series are given this treatment—their scale is placed on the right vertical axis. With \texttt{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. \texttt{OMAX}, \texttt{OMIN}, \texttt{OVLABEL}, and \texttt{OVKEY} control other aspects of the overlay-graph presentation.

\textbf{ovcount=Number of series for right-side scale [1]}

The last \textit{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.

\textbf{omax=}Maximum value for right-side scale [largest value]
\textbf{omin=}Minimum value for right-side scale [smallest value]
\textbf{ovlabel=}Vertical scale label (in quotes: "...") or STRING [none]

OMAX and OMIN allow you to set the maximum and minimum values, respectively, for the overlay 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.

\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 OVSAMESCALE to force both the regular and the overlay series to share a common scale.

**Missing Values**

RATS leaves out any entry for which either of the two series in the pair is missing.

**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.

```
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
```

This computes a histogram for the series \texttt{AGES}, presented as a bar graph. The title on the graph is “Histogram of Ages” and it goes into a window labeled “Histogram”.

```
density(type=histogram) ages / grid density
scatter(style=bar,window="Histogram",header="Histogram of Ages")
# grid density
```
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. \texttt{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 CUMPGDM.SRC).

\texttt{ctor 1 half}
\texttt{\# 1 2}
\texttt{\# actual white\_noise}

\texttt{label actual white\_noise}
\texttt{\# "Actual" "White Noise"}

\texttt{set frqs 1 half = t-1}
\texttt{grp\_parm(font=\"Symbol\") axislabels *}
\texttt{spgraph}

\texttt{scatter(style=line,key=upleft, \$}
\texttt{\hspace{1cm} header=\"Cumulated Periodogram Test\", \$}
\texttt{\hspace{1cm} xlabels=||\"0\",\"p/4\",\"p/2\",\"3p/4\","\"p\"||) 2}
\texttt{\# frqs white\_noise 1 half}
\texttt{\# frqs actual \hspace{1cm} 1 half}
\texttt{display(store=gap\_label) \"gap = \" \#.#### \%maximum}

\texttt{if actual(%max\_ent)<white\_noise(%max\_ent) \hspace{1cm}}
\texttt{\hspace{1cm} grtext(align=left,x=%max\_ent-1,y=actual(%max\_ent)-.01) gap\_label}
\texttt{else}
\texttt{\hspace{1cm} grtext(align=right,x=%max\_ent-1,y=actual(%max\_ent)+.01) gap\_label}
\texttt{spgraph(done)
SEASONAL — Creating Seasonal Dummies

SEASONAL creates a seasonal dummy series.

```
seasonal series start end
```

Wizard

The Data/Graphics—Trend/Seasonals/Dummies wizard can create seasonal dummies.

Parameters

- **series**: Series to set as a seasonal dummy.
- **start** and **end**: Range of entries to set. If you have not set a SMPL, this defaults to the standard workspace plus seasonal–1. See the explanation on the next page.

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.
Examples

seasonal(period=1948:12) december 1948:1 2018:12
seasonal(period=1948:6) june 1948:1 2018:12

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 2018:4
seasonal seasons

defines a 4th quarter dummy over a period which runs 3 entries beyond 2018:4. It is equivalent to:

seasonal(span=4,period=1947:4) seasons 1947:1 2019:3

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 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:

```plaintext
dec vect[series] seasonals(12)
do i=1,12
   seasonal(period=i) seasonals(i) 1947:1 2010:12
end do i
```

creates a vector of series which have the twelve dummies as its twelve elements. You can then use SEASONALS in a regressor list to add the full set.

### 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: LINREG will drop entries at the end of each cross-section. 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 end of the previous segment.

### SEASONAL with Daily/Weekly Data

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 weekday, using as SPAN the number of days per week.

Use SET with logical and date functions to generate other types of calendar dummy variables. For instance,

```plaintext
cal(daily) 1990:1:1
all 2018:12:31
set monday = %weekday(t)==1
```

This uses the `%WEEKDAY` function (which returns “1” for Mondays) to create a MONDAY dummy variable.

You can use other date functions, such as `%MONTH` or `%PERIOD` as an alternative to SEASONAL for monthly, quarterly and other dummies:

```plaintext
set january = %month(t)==1
```

creates a January dummy variable (1 for all entries which fall in a January, 0 otherwise).
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 (1999). 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.

Normal and gamma deviates are generated from the uniforms by acceptance-rejection algorithms.

**See Also . . .**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMULATE</td>
<td>Forecasts a model with random shocks.</td>
</tr>
<tr>
<td>BOOT</td>
<td>Draws random entry numbers.</td>
</tr>
<tr>
<td>%RAN(x)</td>
<td>Function returning draws from Normal$(0, x^2)$ distribution.</td>
</tr>
<tr>
<td>%UNIFORM(x1, x2)</td>
<td>Function returning draws from Uniform$(x1, x2)$</td>
</tr>
<tr>
<td>%RANINTEGER(L, U)</td>
<td>Function returns integer in $[L, U]$</td>
</tr>
</tbody>
</table>
SELECT — User Selection from a List

**SELECT** displays a dialog box asking the user to select items from a list of integers, strings, or data series. You can allow the user to make only one choice or many. Note that you can also include selection boxes as an item in **DBOX**.

```
select(options) number
select(options) number selectlist
```

### Parameters

**number**

If you use just this one parameter, the user can only select one item from the list, and **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 **number** returns the number of items selected.

**selectlist**

To allow the user to select more than one item, supply a new variable name for the **selectlist** parameter. **RATS** will create this variable as a **VECTOR[INTEGERS]** containing the return values of the selected items.

### Options

**series**

**strings=VECTOR of STRINGS to list**

**list=VECTOR of INTEGERS to list**

**regressors**

These four (mutually exclusive) options determine the type of list displayed:

- **Use SERIES** to request a selection from the list of series currently in memory. When you use **SERIES**, the return values are the series numbers. You can use a **selectlist** array directly in a regression list and other locations where a **RATS** instruction requests a “list of series.” The special series **CONSTANT** is never displayed as one of the choices in the list.

- **STRINGS** requests a selection from a list of labels. You must declare and set the **VECTOR[STRINGS]** before executing **SELECT**. The return values are the selected positions (element numbers) in the **VECTOR[STRINGS]**.

- **LIST** to request a selection from an arbitrary list of integers. You must declare and set the **VECTOR[INTEGERS]** before executing **SELECT**. **SELECT** returns the actual integer values selected (not the positions in the array).

- **REGRESSORS** requests 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=(output) 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 (depending upon the option used) list all series in memory or all elements of the VECTOR [STRINGS] or VECTOR [INTEGER]. 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 — 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 Data/Graphics—Transformations for general transformations. Select the “General-Input Formula” action.

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 standard workspace, 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=**real expression/FRML for first entry set [not used]
  Use FIRST when the first entry of series requires a different formula than the remaining entries; for instance, a benchmark value. The option will accept a FRML (formula) so it can adapt to the value of start.

- **startup=**expression/FRML evaluated at period “start”
  You can use the START option to provide an expression which is computed just once, before the regular formula is computed. This allows you to do any time-consuming calculations that don’t depend upon time. It can be an expression of any type.
**Set**

````
smpl=SMPL series or formula  (“SMPL Option” on page RM–546)
    You can supply a series or a formula that can be evaluated across entry numbers.
    Only entries for which the SMPL series or formula are non-zero will be set.
````

````
scratch/[noscratch]
    Use SCRATCH when redefining 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 it if only one of these is true.
````

````
[panel]/nopanel
    When working with panel data, NOPANEL disables the special panel data treat-
    ment of expressions which cross individual boundaries.
````

**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:

````
set trendsq = t^2           Creates “time” squared.
set postwar = t>=1946:1     Creates a postwar dummy for 1946 on.
````

**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. Note that, compared to some
other software, RATS uses `{}` rather than `()` and uses `{1}` for the lag rather than `-1`.

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

```plaintext
set gnp = log(gnp)
```

replaces the series GNP by its log.

```plaintext
set xswitch 1951:1 1998:4 = %if(x>=0.0,xpos,xneg)
```

makes XSWITCH equal to either the current entry of XPOS or of XNEG, depending upon whether X is positive or not.

```plaintext
set(first=0.0) rw = rw{1}+%ran(1.0)
```

makes RW a “random walk” with an initial value of 0 at entry 1, adding N(0,1) increments. Without the FIRST option, this would create a series of missing values since RW{1} isn’t defined when T=1.

```plaintext
set u = %ran(1.0)
set(first=0.0) x = u+.5*u{1}
```

generates U as a series of random numbers, then makes X as a moving average of U. Again, without the FIRST option, the series would be all missing values. (Note, by the way, that you should discard the first few entries of a series generated this way since 0.0 is a convenient starting point but isn’t representative of the stationary MA(1) process).

```plaintext
set lag1 = %if(%period(t)==1,0.0,x{1})
set lag2 = %if(%period(t)<=2,0.0,x{2})
set lag3 = %if(%period(t)<=3,0.0,x{3})
```

This is for panel data. These generate three series called LAG1, LAG2 and LAG3 which are lags of X, padded with 0’s for the entries at the start of the individual’s record for which there is no data. X{1} by itself is a missing value for the first period in an individual’s record.

See Sections 1.4.3 and 1.5.9 of the Introduction for many more examples using SET.

Unnecessary / for Date Range

If you look at a program written for (much) older versions of RATS, you may see a SET instruction that looks like

```plaintext
set gnp / = log(gnp)
```

The added / (for default range) is harmless but unnecessary—if there aren’t start and end parameters, SET uses the default range.
SHOW — Run-Time Information

You can use the instruction `SHOW` to display different types of information.

```
show memory
show series
show files  file identifier
```

**Description**

**show memory**
This displays the amount of memory available to RATS for workspace, the number of bytes used and the number of bytes remaining.

**show series**
Lists the names of all the series currently in use.

**show files  file identifier**
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,

```
show *.rat
```

will list all the files in the current directory with .RAT extensions. If you omit `file identifier`, `SHOW` will list all files on the current directory.
**SIMULATE — Random Simulations of a Model**

**SIMULATE** solves a model, drawing shocks to the equations from a Normal distribution.

```plaintext
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.

**Options**

- **model=** model name
  - Of the two ways to input the form of the model to be solved (the other is with supplementary cards), this is the more convenient and is the only way to forecast with a set of **FRML**s. **MODELS** are usually created by **GROUP** or **SYSTEM**. *If the model includes any identities, those should be last in the model.* If you use this, omit the “equation” supplementary cards.

- **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)

- **results=** VECTOR[SERIES] for result series
  - This provides a **VECTOR** of **SERIES** which will be filled with the results. For instance, **RESULTS=SIMULS** will make **SIMULS(i)** the series of forecasts for the **i**th equation in the model.

- **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 **FRML**s and not just **EQUATION**s, and thus requires an iterative solution *(User’s Guide, Section 8.3).*
Simulate

cv=SYMMETRIC covariance matrix of residuals
factor=RECTANGULAR decomposition of $\Sigma$

You can use CV to supply a $\Sigma$ matrix. If you’re using the MODEL option and you either estimated the model with ESTIMATE or SUR, or you provided a covariance matrix when you set the model up with GROUP, you don’t need to use this. If you need to compute a covariance from a general set of residuals, use VCV and pass %SIGMA from that through using the CV option.

As an alternative, you can use the FACTOR option (the older DECOMP is also acceptable as a synonym) to provide a factorization of the covariance matrix of residuals—see orthogonalization. Note that the distribution (though not the actual simulations) of the simulations doesn’t depend upon which factorization is chosen for $S$.

See “Description” for details.

print/[noprint]

Use PRINT if you want RATS to print the forecasts; this is not done automatically.

Supplementary Cards

There is one supplementary card for each equation. Include these only if you are not inputting a MODEL. Identities should be listed last.

equation
forecasts
(newstart)
column

(Optional) The series for the forecasts of the dependent variable of equation. forecasts can be the same as the dependent variable.

(Optional) The starting entry for forecasts. By default, the same as FROM.

If you use the 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 equations matches the order of the columns in the FACTOR matrix.

Variables Defined

%FSTART Starting entry of forecasts (INTEGER)
%FEND Ending entry of forecasts (INTEGER)
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,\(\Sigma\)) distribution. You put the variance/covariance matrix \(\Sigma\) into **SIMULATE** in one of the following ways:

1. If you use **MODEL**, it will already be part of that if you estimated the model with **ESTIMATE** or **SUR**, or provided a covariance matrix with the **CV** option on **GROUP**.
2. 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.
3. You can supply a factorization of \(\Sigma\) using the **FACTOR** option.
4. In all other cases, \(\Sigma\) 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 16 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. (That is how bootstrapping is done).

To simulate an equation that includes moving average (MA) terms, the equation must have a series of residuals associated with it. This happens automatically for equations estimated using instructions like **BOXJENK**, but for equations defined using **EQUATION**, you must assign the residuals manually, using either **ASSOCIATE** or the %eqnsetresids() function. For example:

```plaintext
set armaresids = 0.0
equation(ar=lags,ma=1,noconstant,coeffs=coeffsvec,$
    variance=1) yeq y
compute %eqnsetresids(yeq,armaresids)
```
Examples

This computes and copies to a file a 100 element realization of the AR(2) process

\[ y_t = 6.0 + 1.4y_{t-1} - 0.8y_{t-2} + u_t \; ; \; \text{Var}(u_t) = 4.0 \]

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:

```r
allocate 150
set y = 0.0
equation(variance=4.0,coeffs=||6.0,1.4,-.8||) simeq y 2 0
simulate(from=3,steps=148) 1
# simeq y
open copy simul.dat
copy 51 150 y
```

This next example does 100 simulations of a six-variable VAR over the period 2011:1 to 2016:4 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.)

```r
system(model=canmodel)
variables usargdps canusxsr cancd90d canmls canrgdps cancpinf
lags 1 to nlags
det constant
end(system)
estimate(noprint)
declare vector minxrate(100) maxxrate(100)

do draw=1,100
   simulate(model=canmodel,results=simuls, $
      steps=24,start=2011:1)
   ext(noprint) simuls(2)
   compute minxrate(draw)=%minimum,maxxrate(draw)=%maximum
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** (Introduction, Section 1.6.2)
  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 1.6.2 of the Introduction.
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 \texttt{SMPL}. In both cases, the \texttt{DATA} instruction reads from 1955:1 to 2009:12, then the other computations are done from 1960:1 to 2009:12.

\begin{verbatim}
cal(m) 1955:1
data 1955:1 2009:12  inter infla
linreg inter 1960:1 2009:12
 # constant infla
set dif 1960:1 2009:12 = inter-infla
set lgs 1960:1 2009:12 = log(dif)
\end{verbatim}

\begin{verbatim}
cal(m) 1955:1
allocate 2009:12
data / inter infla
smpl 1960:1 2009:12
linreg inter
 # constant infla
set dif = inter-infla
set lgs = log(dif)
\end{verbatim}

In the next example, both regressions begin in 1950:1. The \texttt{LINREG} runs through the last entry for which both \texttt{CONS} and \texttt{GNP} are defined. The \texttt{AR1} runs through the last entry for which \texttt{INVEST}, \texttt{YDIFF}, \texttt{GNP\{1\}} and \texttt{RATE\{4\}} are defined.

\begin{verbatim}
smpl 1950:1 *
linreg(inst) cons
 # constant gnp cons\{1\}
ar1(inst,frml=investeq) invest
 # constant ydiff gnp\{1\} rate\{4\}
\end{verbatim}

The last example runs a series of \texttt{NLLS} instructions with increasing numbers of lags in the instrument list. By using \texttt{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.

\begin{verbatim}
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
\end{verbatim}
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 most of the time, this is unnecessary, as RATS 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 page UG–471 in the *User's Guide*.

```
source( options )    filename
```

**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 **SOURCE**ing 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  gammaparms.src
compute  gguess=%GammaParms(%mean,sqrt(%variance))
```

This example brings in the **FUNCTION %GammaParms** from the file **GAMMAPARMS.SRC** and then applies it to determine the parameters for a gamma distribution to hit a particular mean-standard deviation pair.
**Specify**

**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 page UG–253 of the *User’s Guide* for more on the use of priors.

```
specify(type=symmetric, other options)  other’s weight
specify(type=general, other options)  # weights (if you don’t use the MATRIX option)
```

**Wizard**

The *Time Series—VAR (Setup/Estimate)* wizard 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 **SYMMETRIC** 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), *γ*, *g*(l).

**Options**

*type=[symmetric]/general*

*matrix=RECTANGULAR array of weights for TYPE=GENERAL*

**TYPE** selects the *f*(i,j) function. **SYMMETRIC** provides a restricted form of the function, while **GENERAL** allows complete generality. See the discussions below.
tightness=

The overall tightness is the $\gamma$ parameter in the formula above.

mean=

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:

\[
g(l) = l^d \quad \text{for LAGTYPE=HARMONIC} \quad \text{and} \\
g(l) = d^{l^{-1}} \quad \text{for LAGTYPE=GEOMETRIC}
\]

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 "Full Specification of Priors".

The TYPE=SYMMETRIC Option

This specifies the $f(i,j)$ function as follows:

\[
f(i,j) = \begin{cases} 
1.0 & \text{if } i = j \\
w & \text{otherwise}
\end{cases}
\]

where $w$ is the other’s weight parameter. For example:

```plaintext
system(model=sixvar)
variables ip ml 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

The `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 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
dermend(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 $\beta_k \sim N\left(b, \lambda^2 \right)$ 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 (SSpecial 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.

```
spgraph( options )
   One or more graphics instructions
spgraph(done)       To signal end of special graphs.
```

Description

SPGRAPH tells RATS the overall structure of the graphics content. 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.

You must use SPGRAPH(DONE) at the end even if you only need a single graphing instruction.

Positioning

If the SPGRAPH is inside of another SPGRAPH, the fields in the outer SPGRAPH are, by default, filled by column, starting at the top left (field 1,1). If you want to fill a particular field instead, use the combination of ROW and COL options.

Options

```
hfields=number of horizontal fields [1]
vfields=number of vertical fields [1]
fillby=[columns]/rows
```

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 use the ROW and COL options (or hfield and vfield parameters) on GRAPH, GBOX, GCONTOUR, or SCATTER (or inner SPGRAPHs) to control the positioning of each graph. By default, the graph instructions fill the fields by columns, beginning with the top left, working down. Use FILLBY=ROWS to fill across in rows by default instead. Either can be overridden by ROW and COLUMN options on the graphing instructions.

```
done
   You must do SPGRAPH(DONE) after all the graphics instructions. RATS will draw the graph as soon as you execute the SPGRAPH(DONE).
```

```
row=row in the SPGRAPH matrix for this graph
col=column in the SPGRAPH matrix for this graph
```
If you have one SPGRAPH inside another, this allows you to override the positioning. See "Positioning".
samesize/[nosamesize]
Use SAMESIZE if you want all the graphs in the SPGRAPH to be the same size.

header=header string for graph page [none]
subheader=subheader string [none]
footer=footer string [none]
These are nearly identical to the options of the same name on GRAPH, SCATTER, 
GCONTOUR and GBOX, but they add labels for the entire SPGRAPH page, rather 
than the individual graphs. You can use a STRING variable or an explicit string 
in quotes for these. See the second example below. Use the analogous options on 
GRAPH, SCATTER, GCONTOUR or GBOX instructions to label the individual graphs.

window="Window title" [none]
The WINDOW option allows you to set a title for the graph window that will be as-
associated with the graph. By default, graph windows are titled using the HEADER 
or FOOTER options or, if those aren't used, as “Graph.01,” “Graph.02,” etc.

hlabel=horizontal scale label [none]
vlabel=vertical scale label [none]
xlabels=VECTOR[STRINGS] with labels for the horizontal fields
xpos=[upper]/lower/both/none
ylables=VECTOR[STRINGS] with labels for the vertical fields
ypos=[left]/right/both/none
yrotate/[noyrotate]
These options control the labelling on outside of the content (but between 
the headers and footer). HLABEL is a general horizontal label, centered be-
low the content. VLABEL is a general vertical label, written vertically on 
the left side. XLABELS and YLABELS are used to label the individual rows 
(YLABELS) and columns (XLABELS) in the fields. You can supply these as a 
variable of type VECTOR of STRINGS, or using in-line matrix notation (such as 
YLABELS=||"label1","label2"||). XPOS and YPOS determine the positioning 
of the XLABELS and YLABELS respectively. YROTATE determines the orientation 
of the YLABELS. See the examples.

height=height of graph page in inches
width=width of graph page in inches
Use these options to specify a fixed size for the SPGRAPH. You must use both op-
tions together. RATS will display the SPGRAPH at the specified size, with the “Fix” 
toolbar button turned on to freeze the proportions of the graph. It will maintain 
that size if you print or save the graph. Resizing the graph window will resize the 
graph proportionally. Click on the “Unfix” toolbar button if you want to change 
the proportions. To specify sizes in centimeters, use the %CM() function to con-
vert centimeters to inches.

rgf="name of RGF file to insert"
This inserts a saved graph file into an SPGRAPH matrix of graphs. This is the one 
use of SPGRAPH that does not require a closing SPGRAPH (DONE). Use ROW and 
COL, if needed, to override the standard order in which fields are filled.
Options for Adding a Key

If you would like to add a key to the outside of the SPGRAPH array (if the individual graphs use the same basic representation), you have to input all the information to define the key since, at the time the key is being created, none of the content has been added.

patterns/[nopatterns]
This chooses the way the SPGRAPH key distinguishes among series. Ordinarily, RATS will use a different color line or box. If you print the graph on a black and white printer, RATS uses a different type of symbol for each. 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.

key=[none]/above/below/left/right
KEY controls the placement of the key. The choices are:

- NONE: no key
- ABOVE: entered above the content (and any HEADER and SUBHEADER).
- BELOW: centered below the content, and any outer horizontal labeling
- LEFT: left side, centered vertically, outside the content and any outer vertical labeling
- RIGHT: right side, centered vertically, outside the content and any outer vertical labeling

style=[line]/polygon/bar/stackedbar/overlapbar/vertical/step/
symbol/midpolygon/fan/dots/spike
Choose the style that will be used for displaying the information in the graph. There are actually only three ways that a key sample is shown: a line, a filled rectangle or a symbol so you just have to pick a style in the correct "family".

klabel=VECTOR of STRINGS for KEY labels [required]
This is required so SPGRAPH will know how to describe the series. You can create the VECTOR[STRINGS] ahead of time, or enter it using the ||.|| notation.

symbols=VECTOR of INTEGERS supplying style numbers for SERIES
Use SYMBOLS to supply a VECTOR of INTEGERS with the style numbers you want to use for the corresponding series if you don't want the standard behavior of using styles 1, 2, 3, ... in order.

[kbox]/nokbox
This controls whether or not a box (border) is drawn around the key.

kheight=height of key box (between 0 and 1)
width of key box (between 0 and 1)
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 KEYLABELING parameter on GRPARM and so on. KHEIGHT and KWIDTH allow you to control the size and proportion of the box by specifying the height and width of the box as a fraction of the graph's overall height and width. You must specify both options.
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. Together, they 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 3×3 “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)
```

This is from the **BJTRANS.SRC** procedure file. It graphs three transformations of a time series (levels, square root and log). These are all put into the same graph box; **SPGRAPH** is used here to keep the graph “open” while **GRTEXT** applies labels.

```
display(store=header) "Transformations of" %label(series)
do i=1,3
   disp(store=klabels(i)) labels(i)+"\" *### value(i)
end do i

list s = transform
spgraph(window="Transformations")
   graph(scale=none,header=header,key=below,noksample,$
      klabels=klabels) 3
cards s nbeg nend 1
do i=1,3
   grtext(entry=nbeg,align=right,$
      y=%if(transform(i)(nbeg)>(i-1)+.50,$
         transform(i)(nbeg)-.25,transform(i)(nbeg)+.25)) labels(i)
   end do i

spgraph(done)
```
The example below is taken from the `MONTEVAR.SRC` procedure. Note the use of the labeling options available on `SPGRAPH`:

```rat
dec vect[strings] xlabel(nvar) ylabel(nvar)
dec vect[integer] depvars
compute depvars=%modeldepvars(varmodel)
do i=1,nvar
    compute ll=%l(depvars(i))
    compute xlabel(i)=ll, ylabel(i)=ll
end do i

grparm(bold) hlabel 18 matrixlabels 14
grparm axislabel 24
spgraph(header="Impulse responses", xpos=both, xlab=xlabel, ylab=ylabel, vlab="Responses of", vfields=nvar, hfields=nvar)
dec vect[series] upper(nvar) lower(nvar) resp(nvar)
do i=1,nvar
    compute minlower=maxupper=0.0
    smpl 1 nkeep
do j=1,nvar
    clear lower(j) upper(j) resp(j)
do k=1,nstep
        set work 1 nkeep = responses(t)((i-1)*nvar+j,k)
        compute frac=%fractiles(work, ||.16,.84||)
        compute lower(j)(k)=frac(1)
        compute upper(j)(k)=frac(2)
        compute resp(j)(k)=%avg(work)
    end do k
    compute maxupper=%max(maxupper, %maxvalue(upper(j)))
    compute minlower=%min(minlower, %minvalue(lower(j)))
end do j

smpl 1 nstep
do j=1,nvar
    graph(ticks, min=minlower, max=maxupper, number=0) 3 j i
    # resp(j)
    # upper(j) / 2
    # lower(j) / 2
end do j
end do i
spgraph(done)
```

`See Also ...`

- **GBOX** Generates box-and-whisker plots.
- **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, minimum or other quantile of a range of values from a series or formula. This is a very useful instruction. STATS and TABLE provide other ways to get statistical information, but SSTATS doesn’t require you to create a separate series for analysis.

\[
\text{sstats( options ) } \quad \text{start end (pairs of) expression } \Rightarrow \text{result}
\]

**Parameters**

- **start end** Range to use. If you have not set a SMPL, this defaults to the standard workspace. 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 (REAL) variable into which the computed result will be stored.

**Options**

- **smpl=SMPL series or formula** ("SMPL Option" on page RM–546)
  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.

- **mean**
- **product**
- **maximum**
- **minimum**

- **frac=desired fractile (quantile) [not used]**
  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. Use FRAC=.50 for the median value.

- **startup=FRML evaluated at period “start”**
  You can use the START option to provide an expression which is computed once per function evaluation, before the regular formula is computed. This allows you to do any time-consuming calculations that don’t depend upon time. It can be an expression of any type.

- **[panel]/nopanel**
  When working with panel data, NOPANEL disables the special panel data treatment of expressions which cross individual boundaries.

- **weight=series of entry weights** ("WEIGHT option" on page RM–549)
  Use this option if you want to give unequal weights to the observations.
SStats

Variables Defined

- **%NOBS**: Number of observations (INTEGER)
- **%MAXENT**: Entry number of maximum value (if MAXIMUM) (INTEGER)
- **%MINENT**: Entry number of minimum value (if MINIMUM) (INTEGER)

Examples

This is a common use for **SSTATS**. It does a comparison of a series of statistics with a critical value and computes the percentage of those that exceed it. **STATS>%CDSTAT** is either 1 or 0 depending upon whether or not the value of **STATS(T)** is bigger than **%CDSTAT** or not, so the mean of that over the sample will be the fraction for which that is true. The resulting value (called **PVALUE**) is an empirical significance level.

```plaintext
sstats(mean) 1 ndraws (stats>%cdstat)>>pvalue
disp "Bootstrapped p-value" pvalue
```

```plaintext
sstat(smpl=yesvm==0) / 1>>nos testvm<.5>>nnnos prfitted<.5>>prbnos
sstat(smpl=yesvm==1) / 1>>yes testvm>.5>>nnyes prfitted>.5>>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 (2007). Here, we compute pseudo out-of-sample forecasts and use **SSTATS** to compute the (uncentered) second moments of forecast errors. The **@UFOREERRORS** procedure uses an **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))
```
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 Statistics—Univariate Statistics wizard, choose "Basic Statistics".

Parameters

- **series**: Series to analyze.
- **start 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. Use TITLE to supply your own title to label the resulting output.
- **smpl=SMPL series or formula**
  - 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.
- **fractiles/[nofractiles]**
  - If you use the FRACTILES option, STATISTICS computes the maximum, minimum, median and a number of other sample fractiles (quantiles) (1%, 5%, 10%, 25%, 75%, 90%, 95% and 99%). If you want only the fractiles and not these moment-based statistics, use the option NOMOMENTS.
- **[center]/nocenter**
  - NOCENTER can be used if a mean of zero is assumed—it uses variants on the calculations which don't include subtracting off an unknown mean.
- **[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 in "Technical Information".
weight=series of entry weights ("WEIGHT option" on page RM–549)
Use this option if you want to give unequal weights to the observations. Note:
The WEIGHT option has no effect on the FRAC TILES computations.

shuffle=SERIES[INTEGER] for remapping ("SHUFFLE Option" on page RM–548)

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

Notes
RATS has a number of other related instructions which you may find useful. SSTATS
computes statistics on one or more general formulas. EXTREMUM computes the maxi-
mum and minimum values only. MVSTATS computes means, variances, and various
quantiles for a moving window on a series.

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.

open data haversample.rat
calendar(q) 1947:1
data(format=rats) 1947:01 2006:04 gdp gdph m x
*
set reltrade  = (x-m)/gdp
set gdpgrowth = 400.0*log(gdph/gdph{1})
stats gdpgrowth
stats(fractiles,nomoments) reltrade

<table>
<thead>
<tr>
<th>Statistics on Series GDPGROWTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarterly Data From 1947:02 To 2006:04</td>
<td></td>
</tr>
<tr>
<td>Observations</td>
<td>239</td>
</tr>
<tr>
<td>Sample Mean</td>
<td>3.334030</td>
</tr>
<tr>
<td>Variance</td>
<td>15.473951</td>
</tr>
<tr>
<td>Standard Error</td>
<td>3.933694</td>
</tr>
<tr>
<td>SE of Sample Mean</td>
<td>0.254450</td>
</tr>
<tr>
<td>t-Statistic (Mean=0)</td>
<td>13.102912</td>
</tr>
<tr>
<td>Signif Level (Mean=0)</td>
<td>0.000000</td>
</tr>
<tr>
<td>Skewness</td>
<td>-0.071198</td>
</tr>
<tr>
<td>Signif Level (Sk=0)</td>
<td>0.655213</td>
</tr>
<tr>
<td>Kurtosis (excess)</td>
<td>1.396066</td>
</tr>
<tr>
<td>Signif Level (Ku=0)</td>
<td>0.000014</td>
</tr>
<tr>
<td>Jarque-Bera</td>
<td>19.610701</td>
</tr>
<tr>
<td>Signif Level (JB=0)</td>
<td>0.000055</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statistics on Series RELTRADE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarterly Data From 1947:01 To 2006:04</td>
<td></td>
</tr>
<tr>
<td>Observations</td>
<td>240</td>
</tr>
<tr>
<td>Minimum</td>
<td>-0.060909</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.047833</td>
</tr>
<tr>
<td>01-%ile</td>
<td>-0.059073</td>
</tr>
<tr>
<td>99-%ile</td>
<td>0.042288</td>
</tr>
<tr>
<td>05-%ile</td>
<td>-0.046684</td>
</tr>
<tr>
<td>95-%ile</td>
<td>0.012138</td>
</tr>
<tr>
<td>10-%ile</td>
<td>-0.035987</td>
</tr>
<tr>
<td>90-%ile</td>
<td>0.010284</td>
</tr>
<tr>
<td>25-%ile</td>
<td>-0.015881</td>
</tr>
<tr>
<td>75-%ile</td>
<td>0.004513</td>
</tr>
<tr>
<td>Median</td>
<td>-0.002858</td>
</tr>
</tbody>
</table>
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 ($\bar{X}$)**

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

**Variance ($s^2$)**

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \bar{X})^2$$

**Standard Error of Mean**

$$\frac{s}{\sqrt{N}}$$

**$t$-statistic for mean=0**

$$\frac{\bar{X} \sqrt{N}}{s}$$

$m_k$ (used below)

$$m_k = \frac{1}{N} \sum_{i=1}^{N} (X_i - \bar{X})^k$$

**Skewness (Sk)**

$$Sk = \frac{N^2}{(N-1)(N-2)} \frac{m_3}{s^3}$$

**Sk=0 statistic**

$$z = Sk \sqrt{\frac{(N-1)(N-2)}{6N}}$$

**Kurtosis (Ku)**

$$Ku = \frac{N^2}{(N-1)(N-2)(N-3)} \frac{(N+1)m_4 - 3(N-1)m_2^2}{s^4}$$

**Ku=0 statistic**

$$z = Ku \sqrt{\frac{(N-1)(N-2)(N-3)}{24N(N+1)}}$$

**Jarque-Bera**

$$jb = N \left( \frac{(Ku)^2}{24} + \frac{(Sk)^2}{6} \right)$$

**Jarque-Bera test**

$jb$ 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

%NOBS Number of observations (INTEGER)

Variables Defined (MOMENTS)

%MEAN Sample mean (REAL)
%VARIANCE Sample variance (REAL)
%CDSTAT Test statistic for mean=0 (REAL)
%SIGNIF Significance level of %CDSTAT (REAL)
%SKEWNESS Skewness (REAL)
%KURTOSIS Kurtosis (REAL)
%JBSTAT Jarque–Bera statistic (REAL)
%JBSIGNIF Significance level of %JBSTAT (REAL)

Variables Defined (FRACTILES)

%FRACT01 1st percentile (REAL)
%FRACT05 5th percentile (REAL)
%FRACT10 10th percentile (REAL)
%FRACT25 25th percentile (REAL)
%FRACT75 75th percentile (REAL)
%FRACT90 90th percentile (REAL)
%FRACT95 95th percentile (REAL)
%FRACT99 99th percentile (REAL)
%MINIMUM minimum value (REAL)
%MAXIMUM minimum value (REAL)
%MEDIAN median (REAL)

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.

```
steps( options )  equations
#   equation  forecasts  newstart  errors   (one per equation)
```

**Parameters**

- **equations**: Number of equations in the system. You can use a * if you are using the **MODEL** option.

**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 **FRML**s. **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**, **FROM** and **TO** 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 forecasts, or
  - **FROM** and **STEPS** to set the starting date and number of steps (periods)

- **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.
Steps

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 containing FRMLs, not just EQUATIONS, and thus requiring an iterative solution. See page UG–281 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 i-th equation in the model, and STEPSFORE(2,i) the series of one-step forecast errors.

Supplementary Cards (if MODEL option isn’t used)

There is one supplementary card per equation if you don’t use the MODEL option. Note that even if you use supplementary cards for the model, you can still use the RESULTS option for the output information (rather than forecast and errors on the supplementary cards).

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 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.

Variables Defined

%FSTART

   Starting entry of forecasts (INTEGER)

%FEND

   Ending entry of forecasts (INTEGER)

Example

steps(model=canmodel,print,results=stepsfore,from=2000:1, to=2009:4)
do i=1,6
   graph(header="Forecasts of "+%modellabel(canmodel,i)) 2
     # stepsfore(1,i)
     # %modeldepvars(canmodel)(i) %fstart %fend
end do i

This computes and graphs one-step forecast errors for 2000:1 to 2009:4 for a six variable model. The graphs include both the forecasts and the actual data for comparison. Note that this type of graph is likely to be very uninteresting if plotted over a long time span—the forecast errors are generally quite small compared to the overall range of the series, so 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 a RATS file with File—Open, you can write series to the file by doing View—Series Window and dragging series from that window onto the RATS Data 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=binary/cdf/citibase/dbf/dif/dta/fame/free/haver/matlab/
odbc/portable/prn/rats/tsd/wks/xls/xlsx
refresh/[norefresh]

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 Introduction. Omit the list of series when using this option. 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.

request=VECT[STRINGS] with the database names to include

This applies to the CONVERT option and is mainly designed for data formats such as FRED, FAME or HAVER that have too many series to import in their entirety (or may have no option to run through the entire catalog anyway). You create a VECT[STRINGS] with the names that you want, and STORE will pull them off the data base (if they’re present) and add them to 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=\[\text{rows}\]/\text{columns}
For spreadsheet-style formats, 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=\text{"worksheet name"}
When converting data from a format that supports multiple pages or sheets, 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.

skiplines=\text{number of top lines to skip} [0]
top=\text{top line to process} [1]
bottom=\text{top line to process} [\text{last}]
left=\text{leftmost column to process} [1]
right=\text{rightmost column to process} [\text{last}]
You can use these if you have comments or other non-data information on a text or spreadsheet file. SKIPLINES or TOP can be used to skip over information at the top of the file, and the others can be used to prevent reading marginal comments at other locations.

sql=\text{"SQL string"}
Used with FORMAT=ODBC, this allows you to connect to a database and read data using SQL commands. See Section 3.15 of the Additional Topics PDF for more details.

[addates]/noadddates
For XLS, XLSX, DIF, PRN, WKS, ODBC and DBF only. If there are dates on the file, STORE 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.

dateformat=\text{"date format string"}
This can be used if dates on the file are text strings in a form other than “year (delimiter) month (delimiter) day”. In the date format string, use y for positions providing year information, m for positions providing month information, and d for positions with day information. Include the delimiters (if any) used on the file. For example: dateformat="mm/dd/yyyy" or dateformat="yyyymmdd".

missing=\text{code (numeric) for missing values}
With 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
   This allows you to selectively include series based on a series name template.
   It supports the standard filename wildcard characters "*" and "?". For example,
   LIKE="X*" will convert all series whose name begins with X, while LIKE="X?"
   converts only series whose names are two letters long, with the first letter X.

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 unit with the data.

Examples
   cal(q) 1947:1
   all 2018:4
   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
**Store**

**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 `RENAME`ing after you convert the file, since the data base names for series often are very cryptic.

**See Also ...**

*Intro, Section 2.7*  
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 Statistics—Linear Regressions wizard and choose “Stepwise Regression” as the technique.

Parameters

- **depvar**: Dependent variable in the regression.
- **start** and **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.

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. (Some software automatically includes an intercept.)

Options

- **method=[stepwise]/forward/backward/gtos**: 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.

  - **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 \texttt{METHOD=STEPWISE}, \texttt{STWISE} will not allow you to set \texttt{SLSTAY} to a lower value than \texttt{SLENTER}.

\texttt{force=} \textit{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 \texttt{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. For example, the following will include \texttt{CONSTANT} and \texttt{TREND} in all models:

\begin{verbatim}
  stwise(force=2) gnp_82
  # constant trend x1 x2 x3 ...
\end{verbatim}

The following options are the same as those for the \texttt{LINREG} instruction:

\begin{verbatim}
[print]/noprint
vvcv/[novcv]
smpl=SMPL series or formula ("SMPL Option" on page RM–546)
spread=standard SPREAD option ("SPREAD Option" on page RM–547)
dfc=Degrees of Freedom Correction (Additional Topics, Section 1.4)
define=Equation to define (Introduction, Section 1.5.4)
frml=Formula to define
unravel/[nounravel] (User’s Guide, Section 2.10)
weight=series of entry weights ("WEIGHT option" on page RM–549)
\end{verbatim}

**Stepwise Methods**

\texttt{STWISE} performs stepwise regressions using one of four methods, selected using the \texttt{METHOD} option:

**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 \texttt{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 \texttt{SLSTAY} threshold.
**Full stepwise**

This is the default method and combines the first two. At each stage in the forward selection procedure, the backward selection algorithm is run to delete variables which now have small $t$-statistics.

**General to specific**

Use GTOS (for General TO Specific) if you want STWISE to drop regressors starting at the end of the list on the supplementary card. This is useful for pruning lags from an autoregressive model.

**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                        0.2091332
R-Bar^2                             0.1647025
Uncentered R^2                      0.7086280
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             0.0007346
Log Likelihood                       -54.3965
Durbin-Watson Statistic                1.9187

<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>0.00505310</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

%EQNSIZE(0) number of regressors (could also use %NREG)
%EQNCOEFFS(0) coefficient vector (could also use %BETA)
%EQNTABLE(0) 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
%EQNREGLABELS(0) VECTOR[STRINGS] with the regressor labels

If we do the following after the STWISE instruction executed above,

For instance, after the example above

disp %regrlabels()

will show

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, %FREE

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 function of the coefficients of the regression.

```
summarize( options )
# list of variables in regression format (omit with VECTOR or ALL options)
summarize( options ) expression
```

**Parameter**

*expression* To analyze a linear or nonlinear combination of the coefficients, supply the desired expression as a parameter, using elements of the `%BETA` vector or nonlinear parameters (with `PARMSET` option) 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 `VECTOR` or `ALL` options, or if you are using the *expression* parameter to analyze your own function of the coefficients.

**Applicability**

**SUMMARIZE**, in the first 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 estimation instructions that use parameter sets (such as `MAXIMIZE`) with the use of the `VECTOR` option or by using the *expression* form with references to elements of `%BETA`. You can also use the original parameters in the *expression* form if you use the `PARMSET` option.

**Options**

```
[print]/noprint
title="string for output title"
```

- Use `NOPRINT` if you just need to compute results, and don’t need printed output.
- Use `TITLE` to add to the output a description of what is being computed.

```
all/[noall]
```

- Use `ALL` to test the sum of all the coefficients. Omit the supplementary card.
**SUMMARIZE**

`vector=VECTOR of weights`

VECTOR allows you to compute any linear combination of coefficients, not just the sum. The VECTOR should have dimension equal to the number of regressors and give the weights to apply to the regressors in computing the linear combination.

`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 VECTOR to input the weights on the function, and the VALUE option to input the value of that function.

`form=f/chisquared`

This determines the form of the test statistic. By default, RATS selects the appropriate form based on the estimation technique used last. 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 2.11 in the *User's Guide*.

`parmset=PARMSET to use [default internal]`

Use this option to apply the SUMMARIZE operation to the elements of a nonlinear parameter set, rather than to the coefficients in %BETA.

`derives=VECTOR of derivatives`

`numerical/[nonumerical]`

DERIVES returns the VECTOR of derivatives of the expression with respect to the (full set of) parameters in case you need them for further calculations. By default, SUMMARIZE uses analytical derivatives, which aren’t always available (for instance, RATS doesn’t differentiate matrix operations). With NUMERICAL, it does numerical derivatives instead.

**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).
Examples
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 `SHORTRATE`:

```
linreg longrate
# constant shortrate{0 to 24}
summarize
# shortrate{0 to 24}
```

Output
The example above produces the following output:

```
Summary of Linear Combination of Coefficients

<table>
<thead>
<tr>
<th>SHORTRATE</th>
<th>Lag(s) 0 to 24</th>
<th>Value</th>
<th>t-Statistic</th>
<th>Signif Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.97229804</td>
<td>96.53677</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>
```

Examples with VECTOR option
This is from example 5.4 in Greene (2012). 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 × (\text{LOGK coefficient}) + (\text{mean of LOGK}) × (\text{LOGK2 coefficient})
+ (\text{mean LOGL}) × (\text{LOGLK coefficient})
```

```
linreg logq
# constant logl logk logl2 logk2 loglk
sstats(mean) / logk>>logkmean logl>>loglmean
summarize(title="Estimate of K elasticity",$
     vector=||0.0,0.0,0.0,1.0,0.0,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 $b_2/(1 - b_3)$. This is from example 5.6 in Greene (2012).

```
linreg(vcv) logc
# constant logy logc{1}
summarize(title="Long-run MPC") %beta(2)/(1-%beta(3))
```
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) $\left(c\hat{\beta}\right)/\sqrt{c\Sigma_Xc'}$

is treated as a standard normal.
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 2.7 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.

```
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 are estimating. This is ignored if you use the **MODEL** option—you can use * in its place if you need any of the remaining parameters.

**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. If you use supplementary cards to input the model, you can still use the **RESIDS** and **COEFFS** options to save the residuals or coefficients, rather than the **resids** and **coeffs** fields on the supplementary cards.

**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**.

**coeffs**  
(Optional) The series of the coefficient estimates for **equation**.
General Options

**model**=

Model name

This is an alternative way to specify the system of equations to be estimated. **Models** are usually created by **GROUP** or **SYSTEM**. For **SUR**, the **MODEL** must contain only **EQUATIONS**, and no **FRMLs**.

**[print]/[noprint]**
**vcv/[novcv]**
**[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/correlation matrix, respectively.

**resids**=VECTOR[SERIES] for the residuals
**coeffs**=RECTANGULAR array of the coefficients

These allow you to save the estimated residuals and coefficients. Residuals are saved into a VECTOR of SERIES, with element $i$ of the array containing the residuals for equation $i$. Coefficients are saved into a RECTANGULAR array, with the coefficients for equation $i$ stored in column $i$.

**smpl**=SMPL series or formula
**spread**=standard SPREAD option
**dfc**=Degrees of Freedom Correction

These options are the same as for **LINREG**, but the single option applies to all equations. The **SPREAD** option implies a covariance matrix of disturbances of the form $\Sigma \otimes D$, with $D$ diagonal.

**weight**=series of entry weights

Use this option if you want to give unequal weights to the observations.

**shuffle**=SERIES[INTEGER] for remapping

**iterations**=Iteration Limit
**cvcrit**=Convergence criterion

If you give **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 Convergence criterion. **TRACE** prints the intermediate results.

**cv**=Input $\Sigma$ matrix (SYMMETRIC)
**cvout**=Output $\Sigma$ matrix (SYMMETRIC)

**CV** allows you to feed in an initial covariance matrix ($\Sigma$) and **CVOUT** allows you to save the final estimate of the covariance matrix. For **CVOUT**, you don’t need to **DECLARE** or **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 \texttt{SUR(INST)} with a \texttt{CV} of the identity matrix, but the covariance matrix will be incorrect.

\textbf{cmom/[nocmom]}

This pulls cross products out of the cross product matrix computed previously with a \texttt{CMOM} instruction. This can improve calculation time if the \texttt{SUR} is being executed many times with different input \texttt{CV} matrices.

\textbf{create/[nocreate]}

\textbf{setup/[nosetup]}

Use \texttt{CREATE} to print the output from the system if you recompute the coefficients and/or covariance matrix using an instruction other than \texttt{SUR}. This is the systems analogue of the \texttt{CREATE} option for \texttt{LINREG}. \texttt{SETUP} does no estimation: it sets up the \%BETA and \%XX arrays (described below) so that you can compute the coefficients and covariance matrix using matrix instructions. You can then use \texttt{SUR(CREATE...)} to get the output.

\textbf{unravel/[nounravel]}

Substitutes for \texttt{ENCODED} variables (\textit{User’s Guide}, Section 2.10). \texttt{RATS} does not print the intermediate regression (in terms of encoded variables).

\textbf{title="title for output" ["Seemingly Unrelated Regressions"]}

This option allows you to supply your own title to label the resulting output.

\textbf{robusterrors/[norobusterrors]}

\textbf{lags=correlated lags [0]}

\textbf{lwindow=newey/bartlett/damped/parzen/quadratic/[flat]/panel/white}

\textbf{damp=value of $\gamma$ for lwindow=damped [0.0]}

\textbf{lwform=VECTOR with the window form [not used]}

\textbf{cluster=SERIES for clustered std. errors [not used]}

When you use these \texttt{without} the \texttt{INSTRUMENTS} option, they allow you to calculate a consistent covariance matrix allowing for heteroscedasticity (with \texttt{ROBUST}), serial correlation (with \texttt{ROBUST} and \texttt{LAGS}), or clustered standard errors (with \texttt{ROBUST} and \texttt{CLUSTER}). For more information, see Sections 2.2, 2.3, and 2.4, and 4.5 of the \textit{User’s Guide} and “Long-Run Variance/Robust Covariance Calculations” on page RM–540.

None of these affect the parameter estimates. As with \texttt{LINREG} and \texttt{NLLS}, they only come into play when the covariance matrix of the estimates is computed. They behave differently when used with \texttt{INSTRUMENTS}. 
Instrumental Variables Options

instruments/[noinstruments]

Use the INSTRUMENTS option to do three-stage least squares or GMM. You must construct the instruments list in advance using the instruction INSTRUMENTS.

zudependent/[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 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 NOZUDEP, you can use WMATRIX to set the \(W\) part of the \(\Sigma^{-1}\otimes W\) and the CV option to set \(\Sigma\). Otherwise, SUR estimates a new \(\Sigma\) after each iteration.

If ZUDEP, you can use the SW option to set the full SW array. This is an \(nr \times nr\) SYMMETRIC array. Otherwise, SUR determines a new SW matrix after each iteration by taking the inverse of

\[
(1) \quad \frac{1}{T} \sum (u_i \otimes Z_i)(u_i \otimes Z_i)'
\]

(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.

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.

center/[nocenter]

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. For more information, see “ZUMEAN and CENTER options” on page RM–541.

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=ONCE.
**lags=** correlated lags [0]
**lwindow=newey/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 2.2 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 4.9 in the *User’s Guide* for more information.

**Variables Defined**
Because **SUR** estimates a whole set of equations, most of the single equation fit statistics aren’t defined.

- **%BETA** VECTOR of coefficients (across equations)
- **%XX** covariance matrix of coefficients (**SYMMETRIC**)
- **%TSTATS** VECTOR containing the t-stats for the coefficients
- **%STDERRS** VECTOR of coefficient standard errors
- **%NOBS** number of observations (**INTEGER**)
- **%NREG** number of regressors (**INTEGER**)
- **%NFREE** number of free parameters, including covariance matrix (**INTEGER**)
- **%LOGDET** log determinant of the estimate of sigma (**REAL**)
- **%LOGL** log likelihood (if not **INSTRUMENTS** (**REAL**)
- **%SIGMA** final estimate of the \( \Sigma \) matrix (**SYMMETRIC**)
- **%NVAR** number of equations (**INTEGER**)
- **%LOGDET** log determinant of the estimate of \( \Sigma \) (**REAL**)

**Output**
The next page shows the output from a two equation **SUR**. Note that **SUR** prints the regression output (controlled by **[PRINT]/NPRINT** 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

<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>-27.71931712</td>
<td>27.03282800</td>
<td>-1.02539</td>
<td>0.30517701</td>
</tr>
<tr>
<td>2. FGE</td>
<td>0.03831021</td>
<td>0.01329011</td>
<td>2.88261</td>
<td>0.00394396</td>
</tr>
<tr>
<td>3. CGE</td>
<td>0.13903627</td>
<td>0.02303559</td>
<td>6.03572</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>

Dependent Variable IGE

Mean of Dependent Variable | 102.29000000
Std Error of Dependent Variable | 48.58449937
Standard Error of Estimate | 26.25678563
Sum of Squared Residuals | 13788.375833
Durbin-Watson Statistic | 0.9856

<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>4. Constant</td>
<td>-1.251988228</td>
<td>6.956346688</td>
<td>-0.17998</td>
<td>0.85716997</td>
</tr>
<tr>
<td>5. FWEST</td>
<td>0.057629796</td>
<td>0.013411012</td>
<td>4.29720</td>
<td>0.00001730</td>
</tr>
<tr>
<td>6. CWEST</td>
<td>0.063978067</td>
<td>0.048900998</td>
<td>1.30832</td>
<td>0.19076540</td>
</tr>
</tbody>
</table>

Dependent Variable IWEST

Mean of Dependent Variable | 42.891500000
Std Error of Dependent Variable | 19.110188596
Standard Error of Estimate | 9.490260477
Sum of Squared Residuals | 1801.3008785
Durbin-Watson Statistic | 1.3647

<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>7. Constant</td>
<td>-0.392515</td>
<td>0.000325</td>
<td>-1.226193</td>
<td>0.226373</td>
</tr>
<tr>
<td>8. FWEST</td>
<td>0.392515</td>
<td>0.000325</td>
<td>1.226193</td>
<td>0.226373</td>
</tr>
</tbody>
</table>

Covariance\Correlation Matrix of Coefficients

<table>
<thead>
<tr>
<th>Constant</th>
<th>FGE</th>
<th>CGE</th>
<th>Constant</th>
<th>FWEST</th>
<th>CWEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>730.773790</td>
<td>-0.916488</td>
<td>-0.234845</td>
<td>0.675155</td>
<td>-0.623916</td>
</tr>
<tr>
<td>FGE</td>
<td>-0.329266</td>
<td>0.000177</td>
<td>-0.111216</td>
<td>-0.569908</td>
<td>0.673231</td>
</tr>
<tr>
<td>CGE</td>
<td>-0.146242</td>
<td>0.000034</td>
<td>-0.125531</td>
<td>-0.306727</td>
<td>0.247257</td>
</tr>
<tr>
<td>Constant</td>
<td>126.962622</td>
<td>-0.052688</td>
<td>-0.039621</td>
<td>48.390759</td>
<td>-0.857767</td>
</tr>
<tr>
<td>FWEST</td>
<td>-0.226193</td>
<td>0.000120</td>
<td>-0.000017</td>
<td>-0.080023</td>
<td>0.000180</td>
</tr>
<tr>
<td>CWEST</td>
<td>0.392515</td>
<td>0.000325</td>
<td>0.000595</td>
<td>0.113618</td>
<td>-0.000475</td>
</tr>
</tbody>
</table>

Covariance\Correlation Matrix of Residuals

<table>
<thead>
<tr>
<th>IGE</th>
<th>IWEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGE</td>
<td>689.4187916587</td>
</tr>
<tr>
<td>IWEST</td>
<td>190.6362560894</td>
</tr>
</tbody>
</table>
Restriction Across Equations: SUR with EQUATE

Applicability
You should only use EQUATE 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

```plaintext
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 EQUATE 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:

<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>0.000065342</td>
<td>-0.0160284252</td>
<td>-0.8202103255</td>
<td>-0.8753618770</td>
</tr>
<tr>
<td>CWEST</td>
<td>-0.00002975</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”. Note that **SWEEP** produces no direct output. Instead, use the variables it defines.

```
sweep (options) start  end
# list of target variables
# list of instrument variables
```

**Parameters**

`start, end` estimation range. By default, maximum range permitted by all variables involved in the regression, including instruments if required.

**Options**

`smpl=SMPL series or formula` (“SMPL Option” on page RM–546)

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=standard SPREAD option` (“SPREAD Option” on page RM–547)

Use **SPREAD** for weighted least squares. The residual variances are assumed to be proportional to the indicated series.

`weight=series of entry weights` (“WEIGHT option” on page RM–549)

Use this option if you want to give unequal weights to the observations.

`equation=EQUATION for target variables`

Omit target supplementary card if you use this option.

`model=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.

`depvar/[nodepvar]`

Controls whether or not the dependent variable(s) from **EQUATION** or **MODEL** are included as target variables.

`instruments/[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**.
coeffs=(output)  $m \times n$ matrix of coefficients
ibeta=(output) VECT[RECT] of coefficients by group
   For $m$ instruments and $n$ targets, COEFFS saves the estimated coefficients into an $m \times n$ array. With $k$ groups, IBETA creates a $k$-VECTOR of $m \times n$ arrays with the coefficients for each group in a separate element of IBETA.

series=VECT[SERIES] for residuals/fitted values
prj=[residuals]/fitted
   For $n$ targets, SERIES creates an $n$–element VECT 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), SWEEP 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 precision of the estimates.

cvout=Output $\Sigma$ matrix (SYMMETRIC)
   CVOUT allows you to save the final estimate of the covariance matrix.

Variables Defined

%BETA  Averaged VECT of regression coefficients
%XX    SYMMETRIC covariance matrix of averaged regression coefficients
%NOBS  Number of observations (INTEGER)
%NREG  Number of instruments (INTEGER)
%NFREE Number number of free coefficients in the full system, including variances/covariance matrices (INTEGER)
%NREGSYSTEM Number of regressors across groups and targets (INTEGER)
%NVAR  Number of target variables (INTEGER)
%NGROUP Number of groups (INTEGER)
%LOGL  Log likelihood (REAL)
%SIGMA Covariance matrix of residuals (SYMMETRIC)
Examples

\[
\text{sweep(group=\%indiv(t), var=hetero)}
\]

# 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.

\[
\text{set group } = 1+(z\{1\}>=g1t)+(z\{1\}>=g2t)
\]

\[
\text{sweep(grouping=group)}
\]

# 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).

This does a panel causality test (for m causing y) allowing heterogeneity in the coefficients and variances. The first SWEEP is the unrestricted model (including lags of m) and the second excludes the lags of m. This uses an INQUIRE instruction to generate a SMPL dummy which will ensure that both regressions use the same entries.

\[
\text{cal(panelobs=40)}
\]

open data panelmoney.xls
data(org=obs, format=xls) 1//1 19//40 realm realy

* Number of lags
* compute p=3
* set dy = realy-realy\{1\}
set dm = realm-realm\{1\}
* inquire(valid=fullsmpl, reglist)
# dy{0 to p} dm{0 to p}
*

\[
\text{sweep(group=\%indiv(t), smpl=fullsmpl, var=hetero)}
\]

# dy
# constant dy{1 to p} dm{1 to p}
compute loglunr=%logl, nregunr=%nregsystem

\[
\text{sweep(group=\%indiv(t), smpl=fullsmpl, var=hetero)}
\]

# dy
# constant dy{1 to p}
compute loglres=%logl, nregres=%nregsystem

cdf(title=“Heterogeneous Panel Causality Test”) chisqr $ 2.0*(loglunr-loglres) nregunr-nregres
compute jointtest=%cdstat, jointsignif=%signif
**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 7 of the *User’s Guide* for more information about Vector Autoregressions.

```
system(options) list of equations in the system
```

**Wizard**

The *Time Series—VAR (Setup/Estimate)* wizard 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, which is the recommended way to handle a VAR. If you are grouping some dissimilar 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 (though that is no longer an issue in practice).

**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.
**System**

**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(model=ratemod)
variables shortrat longrate m1 twdollar
lags 1 2 3 4 6 9 12
det constant
end(system)
```

creates a four variable VAR. Using the older “numbered equations” method, this would be:

```plaintext
system 1 to 4
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

\[(1) \quad \log P_x = \log P_y + \log X\]

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 View—Series Window operation, you can get a statistics table by selecting the desired series and then using the Statistics toolbar icon: X

**Parameters**

- **start**  
  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**  
  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 (which would only make sense if you are using **TITLE** to supply your own title to label the resulting output.)
You can use **PICTURE** to control the formatting of the numbers in the output table. A picture code 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 "Picture Codes" on page RM–545 for more information.

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** saves 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.

Use **TABLE** if you want to give unequal weights to the observations. This option allows you to supply your own title to label the resulting output.

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 program (except by using the **MATRIX** option). 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.

### Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>%MAXIMUM</strong></td>
<td>maximum value found in the list of series (REAL).</td>
</tr>
<tr>
<td><strong>%MINIMUM</strong></td>
<td>minimum value found in the list of series (REAL).</td>
</tr>
</tbody>
</table>
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 below 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=%l(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.

```
open data cps88.asc
data (format=prn, org=columns) 1 1000 age exp2 grade ind1 married $
   lnwage occ1 partt potexp union weight high
*
table (smpl=union, title=“Statistics for Union Members”) /$
   potexp exp2 grade married high
```

```
table (smpl=not.union, title=“Statistics for Non-Union Members”) /$
   potexp exp2 grade married high
```

This uses TABLE for descriptive statistics on two subsamples of a data set.

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>FCAID</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. EXTREMEUM locates the maximum and minimum values of a single series.
Taper

**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 `WINDOW`) using a window with a sharp cutoff such as the FLAT window. It has much less of an effect with TRIANGULAR or QUADRATIC.

```
taper( options )  cseries  start  end  newcseries  newstart
```

**Parameters**

- **cseries**: Complex series to taper.
- **start end**: Range to taper. By default, the range of `cseries`. However, you will generally need these because the taper is applied to the *unpadded* portion of the data.
- **newcseries**: Series for the result of the tapering. By default, same as `cseries`.
- **newstart**: Starting entry of the tapered series. By default, same as `start`.

**Options**

- **type=[trapezoidal]/cosine**: This gives the type of taper. See the "The TYPE option" for technical details.

  Note that `TAPER` has no option analogous to the `FORM` option of `WINDOW`. Tapering is simply the multiplication of two series, so you can implement other tapering functions fairly easily using `CSET` or `CMULTIPLY`.

- **fraction=**fraction of entries affected at each end [0.25]
- **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 $m$ in the formulas below. `FRACTION` is the most convenient, specifying a fraction of the overall length. The default is .25 of the length.
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):

\[
\text{TRAPEZOID:} \quad b(t) = \begin{cases} \frac{t}{m} & 1 \leq t \leq m \\ 1 & m+1 \leq t \leq N-m \\ \frac{(N-t+1)/m}{(N-t+1)/m} & N-m+1 \leq t \leq N \end{cases}
\]

\[
\text{COSINE:} \quad b(t) = \begin{cases} 0.5[1-\cos(\pi t/m)] & 1 \leq t \leq m \\ 1 & m+1 \leq t \leq N-m \\ 0.5[1-\cos(\pi(N-t+1)/m)] & N-m+1 \leq t \leq N \end{cases}
\]

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.

\%KAPPA the factor to divide into \%EDF and \%EBW (computed by \textsc{WINDOW}) to correct for the effects of the taper.

The values of \%KAPPA for the tapers are

\[
\%\text{KAPPA} = \begin{cases} \frac{[1-8m/5N]}{[1-4m/3N]^2} & \text{TYPE = TRAP} \\ \frac{[1-2(m/N)(93/128)]}{[1-2(m/N)(5/8)]^2} & \text{TYPE = COSINE} \end{cases}
\]

Example

This uses a cosine taper affecting 20% of the data on either end.

\begin{verbatim}
frequency 3 768
rtoc 1956:1 2017:12
# prices
# 1
taper(type=cosine,fraction=.20) 1 1956:1 2017:12
fft 1
cmult(scale=1./(2*%pi*%scaletap)) 1 1
\end{verbatim}
TEST — Testing Specific Coefficient Values

**TEST** tests restrictions of the form $\beta = \text{constant}$. 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.

```
  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 *Statistics—Regression Tests* wizard, 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]**
  
  **ZEROS** 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** before version 7.

- **form=f/chisquared**
  
  This determines the form of the test statistic used. By default, RATS will select the appropriate form for the Wald test 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 2.11 in the *User’s Guide*. 

---

RM–496   RATS Reference Manual
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 3.10 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 %NDFTEST variable is set to the number of degrees of freedom.

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.

Variables Defined
%CDSTAT the computed test statistic (REAL)
%SIGNIF the marginal significance level (REAL)
%NDFTEST (numerator) degrees of freedom for the test (INTEGER)
Test

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] xx2sls=%xx
instruments(add) z3 z4 z5
linreg(inst) y
# constant x1 x2
test(coeffs=beta2sls,covmat=xx2sls-%xx)

performs a Hausman test, comparing the 2SLS estimators from a small instrument set with that from a larger set which contains the first.

linreg lgnp
# constant lgnp{1} trend dgnp{1 to 4}
*
* This computes the joint F-test. We suppress the printing,
* because it will give a misleading p-value, based upon an F and
* not the non-standard distribution.
*
test(noprint)
# 2 3
# 1.0 0.0
disp "ADF Joint test for p=1, and trend=0" %cdstat

This does a Dickey-Fuller joint test. While the Wald test gets the statistic correct, it has a non-standard distribution, hence the use of NOPRINT (which will show a standard F significance level).
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 5.4 in the User’s Guide for more information on the subject.

```plaintext
theil(setup, other options) equations
# list of equations (omit if you use the MODEL option)

theil(options) start

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 **SETUP** option. This describes the model and is done once for each analysis.

2) **Forecasting**, using **THEIL** without the **SETUP** or **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 **FORECAST** instruction in addition to **THEIL**.

3) **Final Statistics**, using **THEIL** with the option **DUMP**. This takes the forecast statistics and prints a table for each variable in the model.

**Parameter—With the SETUP Option**

| **equations** | Number of equations in the system. List the equations on the supplementary card if you are not using the **MODEL** option. |
Options—With the SETUP Option

**steps=** number of forecast steps  
**to=** last period for sample data  
[default series length]

Use **STEPS** to indicate the number of forecast steps (periods) to compute.

Use **TO** to indicate the last period in the sample. Since **THEIL** needs to compare the forecasts with actual data, it will not compute any forecasts for periods beyond that period. **TO** defaults to the default series length.

**model=** general model to be evaluated  
[unused]

Of the two ways to input the form of the model to be solved (the other is using the list on the supplementary card), this is the more convenient and is the only way to forecast with a set of FRMLs. **MODELs** are usually created by **GROUP** or **SYSTEM**. If you use this, omit the “list of equation” supplementary card.

**iters=** iteration limit for solution algorithm  
[50]

**cvcrit=** convergence criterion  
[.00001]

**damp=** damping factor  
(1=no damping)  
[1]

When evaluating a nonlinear model (created by the **GROUP** instruction and specified by the **MODEL** option), you can use these three options to control various aspects of the model-solving procedure.

**estimate=** re-estimation interval  
[1]

Use **ESTIMATE** if you re-estimate the model at some interval other than every period (or never). When you apply the forecasting form of **THEIL**, RATS uses the current estimates to generate sets of forecasts beginning in the re-estimation interval periods after **start**. *You have to code the re-estimation yourself.*

Parameters—Forecasting Form

**start**  
Start of the forecast period. The same information can be input using the **FROM** option.

Options—Forecasting Form

**from=** starting period of the forecast interval  
[last estimation end + 1]

Starting period of forecast. This provides the same information as the **start** parameter.

**forecasts=** (input)  
VECT[SERIES] of forecasts

This allows you to input a series of forecasts which can't be generated by forecasting calculations using standard **EQUATIONs** or **MODELs**. Note that you still have to provide a **MODEL** or set of equations in the **SETUP** form so **THEIL** knows what the dependent variable(s) are.

**print/[/noprint]**

Use **PRINT** to get **THEIL** to print the forecasts together with the actual values. *This can generate a lot of output.*
Options—With the DUMP Option

`replace/[noreplace]`

With REPLACE, THEIL replaces the raw forecast statistics accumulated in the array %THEIL (defined by this instruction) with the finished statistics.

You can only use REPLACE when you have reached the end of the period that you are evaluating, since it destroys the raw statistics.

title="Title for output"

window="Title of window"

You can use the TITLE option to label the output with the string you supply. This will also be used as the title for the copy of the report stored in memory. If you use the WINDOW option, the output will be displayed in a (read-only) spreadsheet window (rather than in the output window) with the window title you supply. You can export the contents of this window to a file in any of several 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 re-estimation 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 re-estimation 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>
<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>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
<td>----------</td>
<td>-------</td>
<td></td>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-0.021564104</td>
<td>0.024578867</td>
<td>0.029866540</td>
<td>0.5736811</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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

```plaintext
boxjenk(define=bjeq,sdiffs=1,ar=2,sma=1) ldeuip * 2003:12
theil(setup,estimate=6,steps=24,to=2009:12) 1
# bjeq
do time=2004:1,2009:12,6
   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 2004:1 to 2009:12. 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.

```
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=2009:4)
estimate(noprint,noftests) * 1997:4
do time=1998:1,2009:4
  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 2009:4. 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 `I`th 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.
Theil

Technical Information
Each time you execute THEIL in forecasting form, RATS computes forecasts for steps horizons (or fewer, if the dataend 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 \( N_t \)—the number of times that a forecast has been computed 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", \( N_t \) was 24, \( N_2 \) was 23, and so on.

When you use \texttt{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 the \texttt{THEIL} forecast step and \( y_t \) is the actual value of the dependent variable.

- **Mean Error**
  \[ ME_t = \frac{SFE_t}{N_t} \]

- **Sum of Absolute Errors**
  \[
  SAE_t = \sum_{i=1}^{N_t} |e_{it}|
  \]

- **Mean Absolute Error**
  \[ MAE_t = \frac{SAE_t}{N_t} \]

- **Sum of Squared Errors**
  \[
  SSE_t = \sum_{i=1}^{N_t} e_{it}^2
  \]

- **Root Mean Square error**
  \[ RMS_t = \sqrt{\frac{SSE_t}{N_t}} \]

- **SSE of no-change forecasts**
  \[
  SSENCF_t = \sum_{i=1}^{N_t} \left( y_{it} - y_{i0} \right)^2
  \]
  where \( y_{i0} \) is the “naive” or flat forecast—simply the value of the dependent variable at the period \((\text{start}-1)\) for the \( i \)th use of the \texttt{THEIL} forecast step.

- **RMS of no-change forecasts**
  \[ RMSNCF_t = \sqrt{\frac{SSENCF_t}{N_t}} \]

- **Theil \( U \)**
  \[ U_t = \frac{RMS_t}{RMSNCF_t} \]
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. \( y_t = C(L)x_t \) (univariate distributed lag)
4. \( y_t = C(L)x_t + B(L)u_t \) (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.

TRFUNC 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

```
equation(noconstant,coeffs=||.9,.3||) arma y 1 1
tfunc(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 — 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 7.14.2 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 10.1 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  datatype  list of parameter names
```

**Parameters**

`datatype`  
Data type 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 or functions 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 5.2 in the **User’s Guide** for more information.

```
uforecast(equation=equation, other options) series start end
```

**Wizard**

You can use the **Time Series—Single-Equation Forecasts** wizard to generate forecasts.

**Parameters**

- **series**
  The series into which you wish to save the forecasts. RATS will create this series if it doesn’t already exist.

- **start end**
  The range over which forecasts are to be computed. You can also use the use the FROM, TO, or STEPS option instead.

**Option**

- **equation=equation to forecast**
  The name or number of the equation to forecast. If you omit this option, **UFORECAST** 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 page UG–161 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.
  Because this is a simple shuffling of the residuals, it would not be completely ap-
  propriate for a model with moving average terms if you’re bootstrapping an entire
  sample.

Variables Defined

%FSTART  Starting entry of forecasts (INTEGER)
%FEND    Ending entry of forecasts (INTEGER)

Examples

The command below forecasts an equation called GDPEQ for the four quarters of 2004.
The forecasts are saved into GDPFORE.

uforecast(equation=gdpeq,from=2004:1,steps=4) gdpfore

This example, from Tsay (2010), uses the @UForeErrors procedure to get bench-
mark forecasts using the sample mean, then compares the errors with those from an
AR(1) estimation. See TSAY3P203.RPF for the full example.

stats ibmln * 1997:12
set benchfore 1998:1 1999:12 = %mean
@uforeerrors ibmln benchfore

  AR(1) model
linreg(define=ar1eq) ibmln * 1997:12
# constant ibmln{1}

Out-of-sample static (one step) forecasts for the AR(1)

uforecast(equation=ar1eq,static) ar1fore 1998:1 1999:12
@uforeerrors ibmln ar1fore

This example, taken from Diebold (2004), uses UFORECAST to generate forecasts from an ARIMA model. See the file DIEB3P216.RPF for the full example.

boxjenk(regressors,ar=3,noconstant,define=ar3eq) lsales $
1968:1 1993:12 resids
# time time2 seasons{0 to -11}

Compute the forecasts, and their standard errors over the year 1994. Generate upper and lower 95% confidence bands

ufore(stderrs=stderrs,equation=ar3eq) fcst 1994:1 1994:12
set upper 1994:1 1994:12 = fcst+1.96*stderrs
set lower 1994:1 1994:12 = fcst-1.96*stderrs

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.

set forezone * 1998:12 = t>=1994:1
graph(header="History and 12-Month-Ahead Forecast"),$
   shading=forezone) 4
# lsales 1992:1 1993:12
# fcst 1994:1 1994:12
# upper 1994:1 1994:12
# lower 1994:1 1994:12
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.

```
until  condition
    instruction or block of instructions to be executed until condition is true.
```

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 enclose the loop in an extra set of { and } to signal the limits 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

```
set smpl = 1.0
compute lastnobs=-1
{
    until  lastnobs==%nobs { 
        compute lastnobs=%nobs
        linreg(smpl=smpl) depvar
        # constant testv1 testv2 testv3 testv4
        set normalize = abs(%resids/sqrt(%seesq))
        set smpl = smpl.and.(normalize<3.00)
    }
}
```

This keeps running a regression, each time dropping all observations with residuals larger than three standard errors until no more observations are dropped.
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 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.

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" for details.
**UserMenu**

**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.

```plaintext
* 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

VADD — Adding a Variable to An Equation

VADD adds a variable or its lags to the EQUATION being modified. MODIFY is needed to start the modification. The coefficients on the new variables are set to zero.

\[ \text{vadd( option ) newseries lags list of lags} \]

Parameters

newseries  The series whose lag(s) you want to add to the equation. Use the variable name \( %\text{MVGAVGE} \) to add a moving average lag.

list of lags  The lags of newseries to add to the equation. Use the second form (without the keyword LAGS or the list of lags) if you just want to add the zero lag.

Option

print/[noprint]  PRINT prints the modified equation.

Examples

linreg(define=gdpeq) gdp
# constant  gdp{1} m1{0 to 3}
modify  gdpeq
vadd  %mvgavge lags 1 2
iterate  gdpeq

This estimates an equation by OLS, then adds two moving average lags and re-estimates it using \textit{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 Time Series—VAR (Setup/Estimate) wizard 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 mlcan 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 save 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 Statistics—Covariance Matrix wizard to compute a covariance matrix. You can also compute a covariance matrix for a set of series by using View—Series Window to open a Series Window, 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. If you need something more general, use CMOM with the proper set of options.

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 ("SMPL Option" on page RM–546)
spread=standard SPREAD option ("SPREAD Option" on page RM–547)
weight=series of entry weights ("WEIGHT option" on page RM–549)
shuffle=SERIES[INTEGER] for remapping ("SHUFFLE Option" on page RM–548)
```

These are standard options. Note that SPREAD and WEIGHT apply to all listed series.
matrix=SYMMETRIC array [%SIGMA]

This saves 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....

Examples

```
vcv(matrix=v) 1921:1 1941:1
# rcons rinv rwage
```

computes and prints the covariance matrix of three series, creates $3 \times 3$ 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.

<table>
<thead>
<tr>
<th>Covariance\Correlation Matrix</th>
<th>RCONS</th>
<th>RINV</th>
<th>RWAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCONS 0.89175982593</td>
<td>0.3010684027</td>
<td>-0.5780087046</td>
<td></td>
</tr>
<tr>
<td>RINV 0.41131881809</td>
<td>2.09304660284</td>
<td>0.3863244971</td>
<td></td>
</tr>
<tr>
<td>RWAGE -0.39361453883</td>
<td>0.40304589112</td>
<td>0.52002665162</td>
<td></td>
</tr>
</tbody>
</table>

```
vcv(center)
# xjpn xfra xsui
compu`te qbar=%cvtocorr(%sigma)
```

Computes the covariance matrix of the three series XJPN, XFRA and XSUI and computes QBAR as the correlation matrix using %CVTOCORR. (%SIGMA and the MATRIX option both give covariance matrices). Since NOCENTER is the default, this uses CENTER to take the means out of the data.

```
vcv(noprint,matrix=v) startl endl
# ulist
```

This computes the covariance matrix of the VECT[SERIES] called ULIST and saves it into the matrix V.
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 \( \text{CMOM} \) 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

\( \%SIGMA \) computed covariance matrix (SYMMETRIC)
\( \%LOGDET \) 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 \( \text{CMOMENT} \) instruction compute similar statistics (but with means subtracted for \( \%COV \) and \( \%CORR \), and \( \text{CMOM} \) if used with the CENTER or CORR options). The instruction \( \text{RATIO} \) takes two sets of compatible residuals and conducts a likelihood ratio test based upon them.

\( \%CVTOCORR(a) \) produces a correlation matrix from a covariance matrix.
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.

```
vreplace( option ) 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. These are described below. You can omit transform if you simply want to replace oldseries by newseries.
- **number**: Depends upon the transformation as indicated above.

Choices for transform

The left column is the choice. These can be abbreviated to three or more characters. The right column is the description.

- **difference**: With number as the number of differences
- **sdiff**: With number as the number of seasonal differences. You can combine DIFF and SDIFF on a single VREPLACE.
- **filter**: With the supplementary cards from FILTER repeated.
- **equation**: Filter with number as the identifier for the EQUATION form used.
- **prewhitened**: Residuals from ARMA with number as the identifier for the estimated EQUATION.
- **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.
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
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.

```
while condition
  instruction or block of instructions to be executed as long as condition is true.
```

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 enclose the loop in an extra set of { and } to signal the limits of the compiled section.

Example

```
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
  }
}
```

computes the sum of the first 20 positive values of x.
Window

**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` Complex series to smooth.
- `start` Range of entries to smooth. This defaults to the defined range of `cseries`. It should always be the range used for the Fourier transform.
- `end` 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 Window Options" for details.
- `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=[flat]/tent/quadratic/triangular`

`form=VECTOR with the window form`

The window weights $w_k$ are symmetric about $k=0$ and add up to one. Using the notation $m=window width, n = (m + 1)/2$, the windows provided through the type option have the (unscaled) form:

$$w_k = \begin{cases} 
1 & \text{for } -(n-1) \leq k \leq (n-1) \\
|n-k| & \text{for } -(n-1) \leq k \leq (n-1) \\
n^2-k^2 & \text{for } -(n-1) \leq k \leq (n-1)
\end{cases}$$

(WINDOW = FLAT)

(WINDOW = TENT, TRIANGULAR)

(WINDOW = 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
Window

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:

\[
\text{declare vector mytent(6)} \\
\text{ewise mytent(k)=} 6^4 - (k-1)^4 \\
\text{window(form=mytent,width=11) 1 / 2}
\]

Comments

The choice of window type is largely a matter of personal preference. With relatively smooth 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

%EDF 
Equivalent Degrees of Freedom (REAL)

%EBW 
Equivalent Band Width (REAL)

The formulas for %EDF and %EBW are

\[
\text{EDF = } \begin{cases} 
3n^3 / (2n^2 + 1) & \text{for WINDOW = TENT, TRIANGULAR} \\
1 & \text{in general}
\end{cases}
\]

\[
\text{EBW = } \frac{\pi \text{EDF}}{T} \text{ where } T \text{ is the number of ordinates}
\]

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

\[
\text{fft 1} \\
\text{cmult(scale=} 1.0 / (2*\pi*%nobs)) 1 1 \\
\text{window(width=} 9) 1 / 2 \\
\text{window(width=} 15, type=} \text{triangular} 1 / 3
\]

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

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=binary/cdf/[free]/html/prn/rtf/tex/tsd/wks/xls/xlsx/"(FORTRAN format)"
  The format in which the data are to be written. A FORTRAN format should describe the format of 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
```
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)
```

Finally, you can also use MEDIT to display an array in a report window, and then export the contents of the window to a file using the File—Export... operation.

See Also...

DISPLAY, MEDIT, REPORT, READ
X11 — X11 Seasonal Adjustment

X11 performs a seasonal adjustment on a series using the Census X11 method as revised for the Census Bureau’s X12-ARIMA program. Most of the changes to this have been to the precision of the calculations. Many of the filters used in the original Census-X11 came from lookup tables, which in many cases rounded coefficients to just three or four significant digits. The new X11 engine uses filters that are generated as needed to the full precision available. This also adds two new adjustment modes (log additive and pseudo-additive) and more flexible allowance for holiday effects.

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

Wizard
Use the Data/Graphics—X11 wizard.

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.

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 from year to year.
- It assumes that the seasonality is primarily a function of the calendar (rather than weather-based).

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.
Options

mode=[multiplicative]/additive/pseudoadditive/logadditive

multiplicative/ [nomult] (for compatibility with older versions of RATS)

The MODE option chooses which of the four adjustment modes to use. These are described later in the supplement. The older MULTIPLICATIVE option still works, but we would recommend correcting any of your programs to use the new option.

print=[none]/short/standard/long/full

decimals=number of decimals to show in output [depends upon data]

PRINT controls the amount of output displayed by the 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.

DECIMALS controls the number of digits right of the decimal to display in any of the output. This has no effect on the level of precision of the calculations or the output series, just the printed output. By default, this depends upon the data.

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 X11.

[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 are 1.5 and 2.5.
prefactors=preliminary adjustment factors [not used]

X12-ARIMA emphasizes the use of preliminary adjustment factors to take care of various type of outliers, rather than using the internal outlier detection engine (which is still present). These are usually estimated using the BOXJENK instruction, and created using the ADJUST option on it.

For multiplicative and log-additive adjustments, these should be in the form of factors, that is, 1.0 means no adjustment. If you are doing a log-additive adjustment starting with a BOXJENK model applied to the log of the series, you will have to transform the factors generated by BOXJENK from their additive form to the multiplicative form.

extension=series with out-of-sample forecasts of the input series
leads=number of periods of extension

The EXTENSION series is used in computing some of centered moving averages within the X11 engine to reduce the bias in the end effects on the filters.

Internal Regression Effects

These options control an internal regression of the irregular component on various dummy variables.

tradeday=[none]/apply

This allows 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.
APPLY Computes trading day factors, applies them to final adjustment.

Before RATS 7.1, the following holidays were switch options. These are still supported, though it’s recommended that any of these be done as preliminary factors instead. All of the holiday shifts are adjusted for long-run mean values, which prevents them from picking up a spurious trend effect for particular ranges of data.

easter=number of days before Easter at which effect is felt

It’s assumed that the level of activity is different for this number of days before Easter. This generates a dummy which splits this among February, March and April based upon the number of days falling in each month. The analogue to the old EASTER switch option is 21, though the calculation is now done differently.

laborday=number of days before Labor Day for the effect

It’s assumed that the level of activity is different for this number of days prior to Labor Day. This generates a dummy which splits this between August and September based upon the number of days falling in each month. The value for this that would be equivalent to the old LABORDAY switch is LABORDAY=8.
thanksgiving=number of days before Thanksgiving for the effect
The value which gives the old adjustment is −1 (the day after Thanksgiving). The level of activity is assumed to be different from this point on to December 24.

critical=critical value (t-stat) for outlier detection [see below]
The internal regression includes automatic detection and removal of additive outliers. This is to prevent contamination of the estimates of the calendar effects by very large outliers. Observations for which an additive dummy has a t-statistic bigger than the indicated limit are removed from the regression. These are added sequentially using a forward pass and possibly deleted using a backwards one. The default value depends on number of observations, but is typically around 3.8.

Examples
This seasonally adjusts DEUIP (German Industrial Production) by applying the multiplicative decomposition. 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(mode=mult,adjusted=deuadjip,factors=deufact) deuip

This uses the @GMAUTOFIT procedure to automatically select a model for the log of the data, and then estimates the selected model using BOXJENK, with automatic detection of outliers. The adjustment series needs to be anti-logged before it can be used as a preliminary set of factors.

The value of FINAL at the end of the data is subtracted before the exp is taken so the adjustment will leave the end of data value at its observed level. This is done because the level shift and temporary change dummies are defined from t0 on, and so will give non-zero shift values to the end of the data, rather than the beginning. (The adjusted data will be the same either way; any printed output looks more natural with this correction). The series is adjusted using the log-additive model, with a full set of printed output.

@gmautofit(report,diffs=1) ldata
boxjenk(ar=%%autop,diffs=1,ma=%%autoq,sar=%%autops,$
  sdiffs=1,sma=%%autoqs,method=bfgs,outliers=standard,$
  adjustments=final) ldata
set prior = exp(final-final(2008:7))
x11(mode=logadd,prefactors=prior,print=full) u36cvs

Technical Information
For technical details, please see the X11 Manual Supplement PDF file (included with the Professional version of RATS).
Section 2: Expression Syntax and Operators

This section describes the operators supported in RATS and the basic elements of expression syntax. See Section 1.4.9 of the Introduction 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 highest 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>
Operator Precedence

“Precedence” is the order in which RATS evaluates the operators in an expression. For example, consider the following instruction:

\[ \text{compute } c = 100 - 5 \times 5 \]

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:

\[
\begin{align*}
1 + 2 \times 3 + 4 \times 5 &= 27 \\
(1 + 2) \times 3 + 4 \times 5 &= 29 \\
1 + ((2 \times 3) + 4) \times 5 &= 51
\end{align*}
\]

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, \(A \times B = A \bar{B}\).
- The ^ (or **) 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:

```r
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:

```r
compute [string] name = "Bob " + "Smith"
```
Section 3: Algorithms

This section documents algorithms which are used in several different instructions. Algorithms which are specific to just one instruction will generally be included in the documentation for that instruction.

Autocorrelations

Long-Run Variance/Robust Covariance Calculations
Autocorrelations

Yule–Walker vs Burg Methods

RATS offers two algorithms for computing autocorrelations and partial autocorrelations. The “textbook” calculation is known as METHOD=YULE (for Yule-Walker). This starts by estimating the autocorrelation at lag $k$ using

$$
\hat{\rho}(k) = \frac{\sum_{t=k+1}^{T} (x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{T} (x_t - \bar{x})^2}
$$

(Optionally without subtracting the mean). 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=BURG, on the other hand, computes the partial autocorrelations directly from the data, combining the forwards and backwards representations of an autoregression. See Burg (1967). 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. However, METHOD=YULE is more common, so in textbook replications, we typically use it to reproduce the reported results.
**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

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

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.
A common feature in modern statistics and econometrics is the need to calculate the covariance matrix in what can be written (informally, we make no attempt to restate this as a theorem) as:

\[
\sum_j z_j \approx_d N(0, \text{var}\left(\sum_j z_j\right))
\]

under various assumptions about the behavior of \(z\). This is used in linear regressions with \(z= Xu\) to correct the covariance matrix for serial correlation or heteroscedasticity, in GMM with \(z= Zu\) or \(z= u \otimes Z\) for weighting moment conditions and in maximum likelihood with \(z=\) partial derivatives to correct for misspecification. The instruction \texttt{MCOV} does direct calculation of this covariance matrix, while the same calculation is included within robust error or weight matrix calculation by instructions such as \texttt{LINREG} or \texttt{MAXIMIZE}.

\textbf{Eicker-White/Heteroscedasticity/Misspecification Consistent}

This is the simplest case. The terms in (1) are independent (or close to it), but not identically distributed. The covariance matrix is approximated as

\[
\text{var}\left(\sum_j z_j\right) \approx \sum_j z_j'z_j
\]

This is done using \texttt{MCOV} with no other options, for correcting covariance matrices with the \texttt{ROBUSTERRORS} option with none of the other options described here, and for GMM weight matrices with \texttt{ZUDEP} with none of the other options.

\textbf{Serial Correlation (LAGS and LWINDOWS options)}

To allow for general (moving average) correlation with \(L\) lags for terms in \(z\), we can use the general calculation

\[
\sum_{l=-L}^L \sum_i w_i \left(z_i'z_{i-l}\right)
\]

where \(w_i\) is a set of window weights. This is chosen using the \texttt{LAGS} option on any of the instructions. If the \(w\)'s are all one (\texttt{LWINDOW=FLAT}), the matrix in (3) may fail to be positive semi-definite, which can produce invalid standard errors. Various window types are provided using the \texttt{LWINDOW} option to avoid this. The formulas for windows are most easily described by defining

\[
z = \frac{|l|}{L + 1}
\]

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.

(5) LWINDOW=FLAT \[ w(z) = 1 \]

(6) LWINDOW=NEWEYWEST \[ w(z) = 1 - z \]

(7) LWINDOW=DAMPED \[ w(z) = (1 - z)^\gamma, \gamma \text{ is the DAMP option.} \]

(8) LWINDOW=PARZEN \[ w(z) = \begin{cases} 1 - 6z^2 + 6z^3 & 0 \leq z \leq \frac{1}{2} \\ 2(1 - z)^2 & \frac{1}{2} \leq z \leq 1 \end{cases} \]

(9) LWINDOW=QUADRATIC \[ w(z) = \frac{3}{(6\pi z / 5)^2} \left[ \frac{\sin(6\pi z / 5)}{6\pi z / 5} - \cos(6\pi z / 5) \right] \]

LWINDOW=PANEL, CLUSTER options

When \( L \) is equal to the number of time series observations (thus allowing for arbitrary patterns of serial correlation), (3) with unit weights is the same as

(10) \[ \left( \sum_t z_t \right) \left( \sum_t z_t \right)' \]

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, Greene(2012), section 11.3.2. If this is used in computing a covariance matrix, you get 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 many more if you want consistency.

ZUMEAN and CENTER options

Use the ZUMEAN=VECTOR of assumed means option when the model assumes that the moment conditions have non-zero mean. It replaces \( z \) with \( z - \mu_z \) in any of the previous calculations where \( \mu_z \) is the VECTOR input by the ZUMEAN option. The CENTER option makes a similar adjustment, but subtracts off 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.
Multivariate Residuals

If you have more than one series of residuals, the calculation in (3) is done as

\[ (11) \sum_{i=-L}^{L} \sum_{t} w_t \left( (u_i \otimes Z_t)' (u_{t-1} \otimes Z_{t-1}) \right) \]

with the appropriate changes for other options. This arrangement (blocking the matrix by equation) matches with that expected for a weight matrix by the instructions \texttt{NLSYSTEM} and \texttt{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 \texttt{NOZUDEP} option can only be used with \texttt{LAGS=0}, and only with \texttt{NOCENTER}. It computes the special case of

\[ (12) \Sigma \otimes Z'Z \]
Section 4: Common Elements

This section documents elements which are common to many instructions.

- Date Notation
- Picture Codes
- SMPL Option
- SPREAD Option
- SHUFFLE Option
- WEIGHT Option
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.
**Picture Codes**

Picture codes are used to choose the formatting for numerical values. They take the form `#####` for integers, or `#####`, `#####.#`, or `*.#` for reals. A complex number will display as (real part, complex part) where each part is formatted according to the picture.

A 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.

A * by itself requests the default formatting, which, for a `DISPLAY` instruction, is generally fourteen total places, typically with five digits right of the decimal. (By default, graphics instructions format the axis labels to align decimals while using as few digits as possible).

On all instructions except `DISPLAY`, the picture code will be in quotes, as it will be part of an option:

```plaintext
graph(picture="*.##")
```

For `DISPLAY` the picture is just another field and isn’t quoted (in quotes, it would be a literal string):

```plaintext
display ##.## \%beta
```

As examples, if the value is 123.4567,

<table>
<thead>
<tr>
<th>Picture code</th>
<th>Will format as</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#####</code></td>
<td>123</td>
</tr>
<tr>
<td><code>*.#</code></td>
<td>123.46 (rounding the fractional part)</td>
</tr>
<tr>
<td><code>#####.#####</code></td>
<td>123.457</td>
</tr>
<tr>
<td><code>*</code></td>
<td>123.45670</td>
</tr>
</tbody>
</table>
SMPL Option

Many instructions that deal with series, including most estimations instructions, have an SMPL option, which allows you to selectively include and omit observations within the entry range used by the instruction. To use it, either:

- create a series with non-zero values (usually ones) in the entries you wish to be part of the sample, and zeros in the entries you wish to exclude, or,
- supply an expression as a function of \( t \) (this can be a FRML) that evaluates to true for the entries you want to include, and false for the entries you want to exclude.

Examples

```plaintext
set small = pop<2000 ;* Will be 1 only when POP<2000
linreg(smpl=small) ...
```

The LINREG skips entries where `SMALL=0` (i.e. where `POP>=2000`).

```plaintext
sstat(smpl=yesvm==0) / 1>>nos testvm<.5>>nnnos $
   lpmfitted<.5>>lpmnos prfitted<.5>>prbnos
sstat(smpl=yesvm==1) / 1>>yes testvm>.5>>nnyes $
   lpmfitted>.5>>lpymes prfitted>.5>>prbyes
```

The first SSTATS operates only on the entries where `YESVM` is 0 and the second operates only on the entries where `YESVM` is 1. NOS will be the count (sum of 1’s) for the `YESVM==0` branch, NNNOS will be the number of entries where `YESVM` is 0 and `TESTVM<.5`, etc.

```plaintext
linreg(smpl=age>=30.and.age<=64,robust) logahe
# yrseduc constant
```

The LINREG runs only over the entries where `AGE` is at least 30 and no more than 64.
SPREAD Option

\texttt{spread=SERIES} with assumed residual variances (at least to a scale factor)

Use the \texttt{SPREAD} option to perform weighted least squares to correct for heteroscedasticity. With this option you provide a series to which the residual variances are assumed to be proportional. ("\texttt{WEIGHT}" options on many packages require you to indicate the reciprocal square root of this series). Note that there is a separate \texttt{WEIGHT} option which is for probability-weighted (or stratified) estimates, which is not the same as a heteroscedasticity correction.

Examples

If variances are assumed to be proportional to the series \texttt{INCOME}:

\begin{verbatim}
linreg(spread=income) food
# constant income
\end{verbatim}

This does “feasible” GLS for correcting heteroscedasticity assumed to be proportional to some (unknown) power of \texttt{INCOME}, estimating that by a linear regression of the log of the squared residuals on log income. That’s exp(\ldots)’ed to produce the estimate of the spread series.

\begin{verbatim}
set esq = log(%resids^2)
set z    = log(income)
  *
linreg esq
# constant z
  *
prj vhat
linreg(spread=exp(vhat)) food
# constant income
\end{verbatim}
SHUFFLE Option

\[\text{shuffle}=\text{SERIES[INTEGER]} \text{ with entry remapping}\]

The SHUFFLE option is available on some instructions for simplifying the task of bootstrapping when the data set allows for direct rearrangement of the observations (if, for instance, the observations are assumed to be independent). It won’t help if a parametric bootstrap is required in, for instance, a vector autoregression.

Before using SHUFFLE, you need to do a BOOT instruction to generate a SERIES[INTEGER] with the random remapping entries. If you have a five entry data set, and BOOT generates the sequence 5, 1, 4, 4, 2, then the STATISTICS instruction with that sequence input using SHUFFLE will compute the sample statistics of the input series at those five entries (with 4 repeated and 3 missing).

As with any bootstrapping, you will typically need to embed that within a loop, with a BOOT generating the remapping, the instruction with SHUFFLE generating statistics, and then some form of bookkeeping to process that.

Example

This is part of the BOOTFGLS.RPF program, which bootstraps a feasible estimator for gls for heteroscedasticity correction. Note that SHUFFLE needs to be used on all three LINREG instructions that are part of the calculation.

```
boot shuffle
*  
* Run the original regression using the bootstrapped sample
*  
linreg(shuffle=shuffle,noprint) food  
# constant income
*  
* Generate ESQ using the original sample data (so it’s compatible with the original data)
*  
set esq = log(%resids^2)
set z    = log(income)
*  
linreg(shuffle=shuffle,noprint) esq  
# constant z
*  
* Get the fitted values (again, this uses the original sample) and run the FGLS estimates.
*  
prj vhat
linreg(spread=exp(vhat),shuffle=shuffle,noprint) food  
# constant income
```
**WEIGHT option**

weight=SERIES of (relative) weights for the data points

The WEIGHT option serves a different purpose from the SPREAD option. WEIGHT is for weighting observations (typically with probability weights), while SPREAD is for “weighting” data due to assumed variance differences. The difference is subtle, as the two options will produce the same regression coefficients when given compatible options (if the two are reciprocals). If you are doing “weighted least squares” for correcting for heteroscedasticity, use SPREAD, not WEIGHT.

If \( w_i \) is the series of weights (they don’t have to sum to 1—they are normalized as part of the calculation), then the mean for any function of the data \( \{x_i\} \) is computed as

\[
\bar{f}(x) = \frac{\sum w_i f(x_i)}{\sum w_i} = \sum \left[ \frac{w_i}{\sum w_i} \right] f(x_i)
\]

Note that the weight applies to the function, not to the data. In particular for least squares of \( y \) on \( X \):

\[
\hat{\beta} = \left( \sum w_i X_i' X_i \right)^{-1} \left( \sum w_i X_i' y_i \right)
\]

where the sum of weights will cancel out since one is in the inverse and so becomes a reciprocal. By contrast, weighted least squares for heteroscedasticity (in effect) divides through the data by \( \sigma_i \) (both \( y \) on \( X \) in the case of a regression) right at the start and carries that through all calculations.

WEIGHT is usually used as part of a much more complicated calculation, as the probability weights are generated by another calculation. The following is part of the ROBUSTSTAR.RPF program—it does 10 passes through a loop that adjusts the weights on observations based upon how extreme the residuals are.

```
set weight = 1.0
do iters=1,10
   linreg(weight=weight,noprint) xc
   # constant xc{1 to 11}
   compute sigma=sqrt(%seesq)
   set weight %regstart() %regend() = $
   stdu=abs(%resids)/sigma,%if(stdu<2.576,1.0,$
   %if(stdu>3.291,0.0,%polyvalue(px,stdu)/stdu))
end do iters
```
Bibliography


