Preface

Welcome to Version 7 of RATS. In the three years since version 6 was released, we’ve continued our two-pronged approach of making the power and sophistication of RATS available to a wider range of users through the addition and improvement of “wizards”, while simultaneously increasing that power and sophistication for our existing user base.

Five years ago, we began a project of implementing (all) the worked examples from a wide range of textbooks. These have included everything from introductory econometrics and time series books to graduate level books covering almost the full range of modern econometrics. More recently, we’ve added replication files for a number of important published papers. Our experience with this (over 1000 running examples) has led to great improvements in the program. We’re grateful to Kit Baum, Tim Bollerslev, Chris Brooks, Kai Carstensen, Richard Davis, Steve De Lurgio, Frank Diebold, Jordi Gali, Bill Greene, Bruce Hansen, Jim Hamilton, Fumio Hayashi, Timo Teräsvirta, Ruey Tsay, Harald Uhlig, Marno Verbeek, Mark Watson and Jeff Wooldridge for their help in providing data and programs, and checking results.

The new documentation includes more and better examples. The chapters of the User’s Guide which have seen the most work are 6 (Hypothesis Tests), 10 (Vector Autoregressions), 12 (Special Models) and 13 (Simulations and Bootstrapping). Chapter 13, in particular, has been almost entirely rewritten to bring it up-to-date with the fast-moving branch of computationally-intensive statistics.

Thanks to all those in the RATS community who have contributed ideas, suggestions and procedures for others to use. Special thanks go to Walter Enders for writing the popular e-book, The RATS Programming Language, Rob Trevor for setting up and maintaining the RATS discussion list, and to Paco Goerlich and Norman Morin for the numerous procedures they’ve written. Jonathan Dennis from the University of Copenhagen (the principal programmer of the CATS program) had a major role in designing and refining the REPORT and DBOX instructions. Chris Sims, as always, had a special role in influencing the direction of our coverage of time series. John Geweke has provided very helpful discussions on Bayesian methods.

Tom Maycock has kept the reference manual up-to-date through six interim versions between 6.0 and 7.0, while handling the bulk of the technical support questions and maintaining our extensive web site. To say that he’s invaluable would be an understatement.

I would also like to thank my wife Robin and daughters Jessica and Sarah for their patience, and Jessica, in particular, for her work on the layout of Macintosh dialog boxes.

Thomas A. Doan
August 2007
# Table of Contents

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

## 1. Basics—An Introduction to RATS

1.0 Basic Operating Instructions .......................................................................................... 2

1.1 Getting Started: An Example ...................................................................................... 3

1.1.1 Describing the Data to RATS .............................................................................. 7

1.1.2 Reading In the Data ........................................................................................... 10

1.1.3 Computing Statistics ......................................................................................... 13

1.1.4 Displaying the Data .......................................................................................... 16

1.1.5 Data Transformations and Creating New Series .............................................. 17

1.1.6 Graphing the Data ............................................................................................ 20

1.1.7 Estimating Regressions .................................................................................... 25

1.1.8 Hypothesis Testing .......................................................................................... 31

1.1.9 Fitted Values and Forecasting .......................................................................... 34

1.1.10 Scalar and Matrix Computations ...................................................................... 36

1.2 Learning More About RATS .................................................................................... 38

1.3 More About Reading Data ....................................................................................... 39

1.4 More About Regressions and Estimations .............................................................. 42

1.5 Non-Linear Estimation ............................................................................................ 45

1.6 More About Forecasting .......................................................................................... 47

1.7 More on Data Transformations ............................................................................... 48

1.8 Arithmetic Expressions ............................................................................................ 52

1.9 Conditional Statements and Loops .......................................................................... 56

1.10 Using PROCEDURE and FUNCTION .................................................................... 58

1.11 RATS Instructions: Syntax ..................................................................................... 60

1.12 More on Entry Ranges ............................................................................................ 64

1.13 Error Messages ....................................................................................................... 66

## 2. Dealing with Data

2.1 The Tools .................................................................................................................. 70

2.2 Where Are Your Data Now? .................................................................................... 72

2.3 Changing Data Frequencies ..................................................................................... 76

2.4 Missing Data ............................................................................................................. 78

2.5 RATS Format ........................................................................................................... 81

2.6 Spreadsheet and Delimited Text Formats ................................................................ 85

2.7 Database Formats, ODBC, and SQL ....................................................................... 90

2.8 Text Files: Free Format .......................................................................................... 92

2.9 Text Files: FORTRAN Format ................................................................................ 95

2.10 Haver Analytics and Other Commercial Databases .............................................. 99

2.11 FAME Format Databases ...................................................................................... 100
5.6 Instrumental Variables and Two-Stage Least Squares ....................................... 192

Example 5.3 Instrumental Variables ....................................................................... 194

5.7 Recursive Least Squares ...................................................................................... 195

5.8 Systems Estimation .............................................................................................. 197

Example 5.4 Grunfeld’s Investment Equations: Use of SUR .................................. 200

5.9 Distributed Lags .................................................................................................. 201

5.10 Information Criteria ............................................................................................ 203

Example 5.5 Choosing Lag Length: Information Criteria ........................................ 204

5.11 Restricted Regressions ....................................................................................... 205

Example 5.6 Polynomial Distributed Lags ................................................................. 207

5.12 Ridge and Mixed Estimators .............................................................................. 208

Example 5.7 Shiller’s Smoothness Prior ................................................................... 210

5.13 Recomputing a Covariance Matrix ...................................................................... 211

5.14 Robust Estimation .............................................................................................. 214

Example 5.8 Robust Estimation ............................................................................... 215

5.15 Miscellaneous .................................................................................................... 217

6. Hypothesis and Specification Testing .................................................................. 219

6.1 Introduction ......................................................................................................... 220

6.2 Technical Information (Wald Tests) ................................................................... 223

6.3 Technical Information (Lagrange Multiplier Tests) ............................................ 224

6.4 Testing for Heteroscedasticity ............................................................................ 226

Example 6.1 Testing for Heteroscedasticity ............................................................... 229

6.5 Serial Correlation ................................................................................................ 230

6.6 Granger-Sims Causality/Exogeneity Tests ........................................................... 232

Example 6.2 Granger-Sims Causality Tests ............................................................... 233

6.7 Structural Stability Tests or Constancy Tests ..................................................... 234

Example 6.3 Chow Test Using Subsample Regressions ........................................... 236

Example 6.4 Chow Test Using Dummy Variables .................................................... 237

Example 6.5 General Constancy Tests ..................................................................... 238

6.8 Functional Form Tests ........................................................................................ 240

6.9 Specification Tests .............................................................................................. 242

Example 6.6 Hansen’s Specification Test .................................................................. 246

Example 6.7 Hausman’s Specification Test ................................................................. 247

6.10 Unit Root Tests .................................................................................................. 249

6.11 Cointegration Tests ............................................................................................ 252

Example 6.8 Cointegration Testing Procedures ......................................................... 254

7. Non-Linear Estimation ......................................................................................... 255

7.1 General Principles and Problems ........................................................................ 256

7.2 Newton–Raphson and Related Methods ............................................................. 258

7.3 Derivative-Free Methods ..................................................................................... 262

7.4 Constrained Optimization ................................................................................... 264
10.6 Using IMPULSE and ERRORS ................................................................. 364
  Example 10.3 Graphing an Impulse Response Function .................. 368
  Example 10.4 Structural Decomposition .............................................. 370
10.7 Historical Decomposition ................................................................. 372
  Example 10.5 Historical Decomposition: Use of HISTORY ............ 373
10.8 Cointegration and Error Correction Models ........................................... 374
  Example 10.6 Error Correction Model .................................................... 376
10.9 VAR’s for Forecasting: The Bayesian Approach ........................................... 377
10.10 Differences When Using a Prior ............................................................. 382
10.11 Choosing a Prior .................................................................................... 383
  Example 10.7 Using RUNTHEIL to pick a model ............................. 384
10.12 Selecting a Model .................................................................................. 385
10.13 Using the Kalman Filter ....................................................................... 388
  10.13.1 Sequential Estimation ................................................................. 389
  10.13.2 Time-Varying Coefficients ......................................................... 391
  10.13.3 More General Time-Variation ..................................................... 394
  10.13.4 Dummy Observation Priors ......................................................... 396
10.14 Conditional Forecasts .......................................................................... 397
  Example 10.8 Conditional Forecasting ................................................. 400

11. Simultaneous Equations 403
  11.1 Estimation ............................................................................................ 404
    Example 11.1 Simultaneous Equations Estimation ............................. 406
  11.2 Setting Up a Model for Forecasting ..................................................... 409
  11.3 Solving Models: Various Topics ............................................................ 411

12. Special Models and Techniques 415
  12.1 ARCH and Related Models ................................................................. 416
    12.1.1 Standard Models ........................................................................... 417
    12.1.2 Fat-Tailed Distributions ............................................................... 419
    12.1.3 Asymmetry .................................................................................... 420
    12.1.4 The Mean Model ........................................................................... 421
    12.1.5 ARCH-X, GARCH-X Models ......................................................... 422
    12.1.6 Estimation Options ....................................................................... 422
    12.1.7 Output ........................................................................................... 423
    12.1.8 Using MAXIMIZE ......................................................................... 424
    12.1.9 Out-of-Sample Variance Forecasts .............................................. 426
    Example 12.1 ARCH and GARCH Models ........................................... 427
    12.1.10 Multivariate GARCH Models ...................................................... 428
    Example 12.2 Multivariate GARCH ....................................................... 434
  12.2 Dynamic Linear (State Space) Models ................................................... 436
    Example 12.3 Hodrick-Prescott Filter .................................................... 446
    Example 12.4 Estimating a State-Space Model ...................................... 448
## Introduction

12.3 Markov Switching Models ................................................................. 449  
**Example 12.5** Hamilton Switching Model ........................................ 455  
**Example 12.6** SWARCH (Switching ARCH) ...................................... 458  
12.4 Non-Parametric Regression and Density Estimation .................... 460  
**Example 12.7** Non-Parametric Regression ...................................... 461  
**Example 12.8** Adaptive Kernel Estimation ................................. 462  
12.5 Linear and Quadratic Programming ............................................. 464  
**Example 12.9** Portfolio Optimization ......................................... 467  
12.6 Neural Networks ............................................................................ 469  
**Example 12.10** Neural Network .................................................. 476  

### 13. Simulations, Bootstrapping, and Monte Carlo Techniques 477

13.1 Bookkeeping ................................................................................ 478  
13.2 Bootstrapping and Simulation Tools ........................................... 482  
**Example 13.1** Monte Carlo Test Simulation .................................... 486  
13.3 Rejection Method ........................................................................ 487  
13.4 Standard Posteriors .................................................................... 491  
13.5 Monte Carlo Integration ............................................................. 495  
**Example 13.2** Monte Carlo Integration ........................................ 497  
13.6 Importance Sampling ................................................................. 500  
**Example 13.3** Importance Sampling ........................................... 502  
13.7 Gibbs Sampler ............................................................................. 505  
**Example 13.4** Gibbs Sampling .................................................... 507  
13.8 Metropolis-Hastings ................................................................. 510  
**Example 13.5** Metropolis-Hastings .............................................. 512  
13.9 Griddy Gibbs .............................................................................. 515  
13.10 Bootstrapping and Resampling Methods .................................... 516  
**Example 13.6** Bootstrapping a GARCH Model ......................... 518  
**Example 13.7** Approximate Randomization .................................. 520  

### 14. Cross Section and Panel Data 523

14.1 Overview ...................................................................................... 524  
14.2 Probit and Logit Models ............................................................. 525  
**Example 14.1** Logit and Probit Models ....................................... 531  
**Example 14.2** Probit and Linear Probability Models .................... 532  
14.3 Censored and Truncated Samples ............................................. 533  
**Example 14.3** Tobit Models ......................................................... 539  
14.4 Hazard Models ............................................................................ 540  
14.5 Panel and Related Data Sets .................................................... 542  
14.6 Statistical Methods for Panel Data ............................................ 548  
14.7 Fixed and Random Effects Estimators ...................................... 552  
**Example 14.4** Fixed and Random Effects Estimators .................. 553  
**Example 14.5** Random Coefficients Model .............................. 554  
**Example 14.6** Arellano-Bond GMM Estimator .......................... 555  

---

* RATS User’s Guide
New Features and Improvements

Following is an overview of the improvements and changes in RATS Version 7. Note that many of these were introduced in interim versions. If you're interested in keeping your version of RATS current between major releases, you should look into getting an update subscription.

First, we describe some of the interface improvements. Next, we list the new instructions added since 6.0. This is followed by a description of some general improvements and possible compatibility issues. Finally, we provide a detailed list of changes and improvements to individual instructions, and a note on new functions.

Interface Improvements

RATS 7 greatly expands the number of instructions covered by “wizards.” (There are so many now that there are three menus covering them: Wizards, Data (which covers data reading and transformation) and Statistics. The wizards generate (and execute) RATS instructions, complete with options, parameters, and supplementary cards, which makes this feature particularly useful for new users who are learning the RATS language. For more experienced users, the most helpful wizards are probably those for reading data (Data (RATS Format) and Data (Other Formats) on the Data menu) and those for functions and variables (Function and RATS Variables on the Wizards menu).

The method of obtaining “regressor lists” in the new and existing wizards has been simplified so you can either type or edit the list directly, or you can use a specialized popup dialog box which allows variables to be selected out of a scrolling list.

Syntax Cleanup

For version 7, we have made major changes to standardize the format of instructions and the names of options. With few exceptions, instruction parameters are now used only for input or output series and entry ranges. Other types of information (whether input or output) are provided using options. Options have names and can be input in any order, making them easier to remember, use, and read than parameters, which need to be in a very specific order. While we still support the older syntax (RATS is largely backwards compatible to Version 4), in most cases we haven’t documented in the main manual the clumsier ways of writing instructions. There’s a separate PDF file that includes the older instructions and options.

Among the changes are new FROM, TO and STEPS options for the forecasting instructions, replacing their use of parameters; and CV and CVOUT options for inputting and retrieving a covariance matrix in a wide range of instructions, replacing options (ISIGMA on SUR and NLSYSTEM) and parameters (the 3rd parameter on IMPULSE, ERRORS and SIMULATE).
New Instructions

**DSGE**
Takes a model with expectational terms (indicated by leads) and solves out for a linear or linearized state space model. Note that this is *not* included in the main documentation. See the supplementary PDF file.

**Sweep**
Handles a wide range of analyses based upon a regression of a set of targets on a common set of variables, whether it’s extraction of residuals from components or heterogeneous regressions.

**QZ**
Does a generalized Schur decomposition of a pair of matrices. This is a step which is now included within the **DSGE** instruction.

Other New Features

There have been major changes to the way that graphs are produced and displayed. There are now 30 “representations” for line types, fill types and symbols, describing the pattern or shape, color or gray level and thickness. Each representation comes in a color form and a black and white form. While the color form is almost always the easiest to use on the screen, most hard-copy output still needs to be in black and white. You can now easily switch between the two views using a toggle button on the toolbar when you have a graph displayed. You can also define your own combination of attributes for a representation.

The graphing instructions also have new **FOOTER** and **FRAME** options, and almost any text can be shown in multiple line fields by using `\` in a string to signal a line break.

The four “least squares” instructions (**LINREG** and **NLLS** for single equations, **SUR** and **NLSYSTEM** for systems) now have a common set of options for handling GMM estimation in a flexible way. The **UPDATE** option, for instance, allows for several methods of updating weight matrices which were available in the past for some of the instructions but not others. The new **JROBUST** option allows for the J specification statistic to be “robustified” by either recomputing the statistic or using a non-standard distribution. The biggest change here is to **SUR**, which did not have any options for GMM.

The **DLM** instruction was almost completely reworked from version 6. Although older code using this will still work, the input into this can now be much more natural, and there are many more options for extracting information.

The **CLUSTER** option allows for clustered standard error calculations in any instruction which allowed for robust standard errors before.

Estimation instructions which use **PARAMSETS** now have **REJECT** options which allow the function to return an **NA** (missing value) immediately when parameters go out of range.
New Functions

See Section 2 in the Reference Manual for a complete list of the functions. Almost ninety new functions have been introduced since 6.0. The principal changes are the addition of a number of new random draw functions, particularly “convenience functions” for drawing from standard posteriors; density, distribution and inverse distribution functions not previously available, and new functions which access the internal “perpetual calendar.”

New Operators

There are several new operators which can be applied to matrices. \( A + r \) and \( A - r \) for array \( A \) and real number \( r \) are interpreted to add or subtract \( r \) from all elements of \( A \). Matrix exponentiation (repeated multiplies) can be done using \( A^n \) (or \( A**n \)) where \( n \) is an integer. Elementwise exponentiation can be done with \( A. ^r \), which will take each element of \( A \) to the real power. \( A \sim B \), \( A \sim\sim B \) and \( A \sim \backslash B \) do horizontal, vertical and diagonal concatenation of matrices \( A \) and \( B \) (largely replacing the \%BLOCKGLUE and \%BLOCKDIAG functions).

Incompatibilities

If you used END 1 (or something like it) to split up programs within a single text file, change it to END(RESET). We made this change because END instructions were too easily misinterpreted.

To keep the size of the manual more manageable, and to prevent confusion, we’ve eliminated descriptions of older option names and parameters (and some entire instructions) from the Reference Manual. However, anything which worked with version 6 should still work with version 7.

Changes to Existing Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR1</td>
<td>The new PRHOS option allows retrieval of the individual AR(1) coefficients when doing heterogeneous panels.</td>
</tr>
<tr>
<td>ASSOCIATE</td>
<td>New %EQNSET... functions make this largely obsolete.</td>
</tr>
<tr>
<td>BOOT</td>
<td>The options BLOCK, METHOD and GEOMETRIC allow several types of block bootstrapping.</td>
</tr>
<tr>
<td>BOXJENK</td>
<td>Offers the choice of least squares (as before) and maximum likelihood with the MAXL option. This now has PMETHOD and SMPL options.</td>
</tr>
<tr>
<td>CALENDAR</td>
<td>This supports a new and easier to use syntax.</td>
</tr>
<tr>
<td>CATALOG</td>
<td>Adds the LIKE option for restricting the listing.</td>
</tr>
<tr>
<td>CMOM</td>
<td>Adds a MODEL option to pull the variables in the cross moment matrix out of an existing equation.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>CORRELATE</td>
<td>Has an <strong>ORG</strong> option to allow choice between two layouts for the output.</td>
</tr>
<tr>
<td>CROSS</td>
<td>Similar change to <strong>CORRELATE</strong>. <strong>FROM</strong> and <strong>TO</strong> options were added to replace parameters.</td>
</tr>
<tr>
<td>CVMODEL</td>
<td>The <strong>V</strong> option allows direct modeling of the covariance matrix rather than just its factors. Adds the <strong>REJECT</strong> option.</td>
</tr>
<tr>
<td>DATA</td>
<td>Adds the <strong>SHEET</strong> option for reading from spreadsheets. <strong>SQL</strong> option and <strong>FORMAT=ODBC</strong> were added to the Professional version to read from SQL databases.</td>
</tr>
<tr>
<td>DBOX</td>
<td>Adds options for editing matrices and selecting elements within a fixed matrix. Added <strong>SPINBUTTON</strong> object type.</td>
</tr>
<tr>
<td>DDV</td>
<td>The <strong>COEFFS</strong> option allows you to retrieve the full coefficient matrix for multinomial logit. The <strong>EQUATION</strong> option can be used to feed in the equation to be estimated.</td>
</tr>
<tr>
<td>DECLARE</td>
<td>Can be included at any location within a procedure (not just at the start).</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>Adds options <strong>D</strong> and <strong>PADDING</strong> for fractional differencing.</td>
</tr>
<tr>
<td>DLM</td>
<td>This had a major reworking. <strong>TYPE=SIMULATE</strong> and <strong>TYPE=CSIMULATE</strong> were added for random simulation and conditional simulation. There are options to retrieve many more intermediate calculations. Support was added for partial observables in multivariate models. The input to it through the options has been made more flexible and natural. There are new options for an inverse chi-squared prior on observation equation variance. The new <strong>PRESAMPLE</strong> option allows <strong>DLM</strong> to deal automatically with models which are partially stationary and partially non-stationary.</td>
</tr>
<tr>
<td>EIGEN</td>
<td>The <strong>GENERALIZED</strong> option allows it to solve the generalized eigenvalue problem for symmetric matrices.</td>
</tr>
<tr>
<td>END</td>
<td>This now needs a <strong>RESET</strong> option to reset memory. This prevents accidental reset.</td>
</tr>
<tr>
<td>ERRORS</td>
<td>This adds the <strong>STDERRS</strong> option for retrieving the calculated standard errors of forecast.</td>
</tr>
<tr>
<td>ESMOOTH</td>
<td>The option <strong>FITTED</strong> allows you to obtain (easily) the one-step forecasts. The <strong>CONSTRAIN</strong> option constrains estimated parameters to the unit interval.</td>
</tr>
</tbody>
</table>
**FILTER**  
This adds `TYPE=SPENCER` for Spencer moving averages, `WEIGHTS` for general filters, and the `EXTEND` option for methods of extending the filter at the ends of the data sets (other than truncation of the output). The `REMOVE` option allows a convenient way to detrend or deseasonalize data by regression.

**FIND**  
Adds the `REJECT` option.

**FORECAST**  
This adds the `STDERRS` option to get the standard errors of forecast without a separate `ERRORS` instruction. The options `STATIC` and `ERRORS` allow static (in-sample) forecasting without using a separate `STEPS` instruction.

**GARCH**  
This adds the `I` option for IGARCH. The new `VARIANCES` option controls the variance model for individual series in a CC or DCC. `MV=VECH` and `MV=EWMA` options for multivariate GARCH have been added. The `MVHSERIES` allows the variance series to be retrieved in a form usable for multivariate GARCH-M models. The `DERIVES` option returns the derivatives of the likelihood with respect to the parameters.

**GCONTOUR**  
This adds `FRAME` and `FOOTER` options.

**GRAPH**  
The adds the `FOOTER` and `FRAME` options. There are new `FAN`, `DOT` and `SPIKE` options for `STYLE` and `OVERLAY`. The `OVRANGE` options allows the left and right scales in an overlay graph to have separate vertical ranges. It adds the standard `SMPL` option.

**GRPARM**  
This adds the `PATTERNS` option for globally choosing to use black and white patterns rather than colors. The `IMPORT` option pulls in user-definable choices for representations of graphs. The `RECALL` option, when combined with the `%GRPARM()` function allows you to save and restore settings.

**GRTEXT**  
This has new choices for the `POSITION` option: `BOTTOMMARGIN`, `LEFTMARGIN` and `TOPMARGIN`.

**HISTORY**  
This adds `FROM`, `TO` and `STEPS` options.

**IMPULSE**  
This adds the `STEPS` option. The `CV` and `COLUMN` options are designed to replace parameters.

**INCLUDE**  
The new `COMMENTS` option allows comments to be calculated strings, not just input text.

**INQUIRE**  
The `EQUATION` option allows analysis of the range determined by the variables in an equation. The `VALID` option returns a dummy variable with the valid data points in an equation or regression.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KALMAN</td>
<td>The options $X$, $Y$ and $V$ have been added to do dummy observation priors.</td>
</tr>
<tr>
<td>LDV</td>
<td>Adds the EQUATION option to specify the equation being estimated. Adds the CLUSTER option.</td>
</tr>
<tr>
<td>LINREG</td>
<td>This adds CENTER, JROBUST, UPDATE, ZUDEP and ZUMEAN options for more general GMM.</td>
</tr>
<tr>
<td>LOCAL</td>
<td>Can accept dimension fields and can be placed anywhere in a procedure or function.</td>
</tr>
<tr>
<td>LQPROG</td>
<td>This has options for all inputs. The $\geq$ option allows for constraints that are naturally stated as $\geq$. NNEG (for quadratic programs) allows parameters to be not constrained to be non-negative.</td>
</tr>
<tr>
<td>MAKE</td>
<td>The PANEL option allows creation of a $T \times N$ matrix from a panel data series.</td>
</tr>
<tr>
<td>MAXIMIZE</td>
<td>Adds the REJECT and CLUSTER options.</td>
</tr>
<tr>
<td>MCOV</td>
<td>Adds the MODEL option to take variables out of an existing model. It also adds the CLUSTER and ZUMEAN options.</td>
</tr>
<tr>
<td>MVFRACTILE</td>
<td>Has been combined into MVSTATS.</td>
</tr>
<tr>
<td>MVSTATS</td>
<td>Includes options formerly in MVFRACTILE. It adds an option for handling end-of-sample data windows.</td>
</tr>
<tr>
<td>NLLS</td>
<td>This adds SV and SPREAD options for weighted least squares either including estimated parameters (SV) or not (SPREAD). JACOBIAN allows for Jacobian terms for maximum likelihood. DERIVES returns the derivatives of the residual with respect to the parameters.</td>
</tr>
<tr>
<td>NLSYSTEM</td>
<td>This adds the FRMLVECT option for a model which generates the residuals in a VECTOR form, the MODEL option for more convenient input. JACOBIAN allows for Jacobian terms for full-information maximum likelihood. DERIVES returns the derivatives of the residuals with respect to the parameters. It adds REJECT and CLUSTER and the full set of GMM options: CENTER, JROBUST, UPDATE, ZUDEP and ZUMEAN.</td>
</tr>
<tr>
<td>NONLIN</td>
<td>PARMS can now include VECT[VECT] and other more complicated array structures.</td>
</tr>
<tr>
<td>PREGRESS</td>
<td>This adds METHOD=BETWEEN. It performs an automatic Hausman test with METHOD=RANDOM. INDIV and TIME options are available to retrieve components.</td>
</tr>
<tr>
<td>PRINT</td>
<td>This adds the standard SMPL option.</td>
</tr>
</tbody>
</table>
**Introduction**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRJ</td>
<td>This has <code>LOWER</code> and <code>UPPER</code> options for general truncations.</td>
</tr>
<tr>
<td>PRTDATA</td>
<td>Adds the <code>LIKE</code> option for restricting the series printed.</td>
</tr>
<tr>
<td>REPORT</td>
<td>This adds the options <code>UNIT</code> and <code>FORMAT</code> for direct export of the table. New option choices are <code>ACTION=SORT</code> with <code>BYCOLUMN</code> and <code>SPECIAL=BRACKETS</code>.</td>
</tr>
<tr>
<td>RLS</td>
<td>Adds the <code>CONDITION</code> option to condition on more than the minimal number of observations.</td>
</tr>
<tr>
<td>SCATTER</td>
<td>Adds <code>STYLE=SPIKE</code> and <code>STYLE=STEP</code>. Adds <code>FOOTER</code> and <code>FRAME</code>. And now has the standard <code>SMPL</code> option.</td>
</tr>
<tr>
<td>SET</td>
<td>Now has a <code>SMPL</code> option to leave observations out of the calculation.</td>
</tr>
<tr>
<td>SIMULATE</td>
<td>This adds <code>FROM</code>, <code>TO</code>, <code>STEPS</code> and <code>CV</code> options.</td>
</tr>
<tr>
<td>SPGRAPH</td>
<td><code>SPGRAPH</code>s can be nested so you can add special text (with <code>GRTEXT</code>) to graphs within a matrix. It also adds the <code>FOOTER</code> option.</td>
</tr>
<tr>
<td>STEPS</td>
<td>This has <code>FROM</code>, <code>TO</code> and <code>STEPS</code> options. Note though that it is largely obsolete due to new options on <code>FORECAST</code>.</td>
</tr>
<tr>
<td>STORE</td>
<td>Adds the <code>SHEET</code> option for reading from spreadsheets. <code>SQL</code> option and <code>FORMAT=ODBC</code> were added to the Professional version to read from SQL databases. This has the <code>LIKE</code> option for restricting the set of variable names imported. <code>REFRESH</code> imports only the data for series already on the file.</td>
</tr>
<tr>
<td>STWISE</td>
<td>Adds options for HAC standard error calculations (<code>ROBUSTERRORS</code>, <code>LAGS</code>, etc.)</td>
</tr>
<tr>
<td>SUMMARIZE</td>
<td>Accepts non-linear expressions (on the command line). It does computes the (approximate) variance using the delta method.</td>
</tr>
<tr>
<td>SUR</td>
<td>This adds the <code>MODEL</code> option, and a large number of options for full support for GMM.</td>
</tr>
<tr>
<td>TABLE</td>
<td>Includes <code>TITLE</code> and <code>PICTURE</code> options</td>
</tr>
<tr>
<td>THEIL</td>
<td>Adds the <code>FROM</code>, <code>TO</code> and <code>STEPS</code> options.</td>
</tr>
<tr>
<td>UFORECAST</td>
<td>Now includes <code>BOOTSTRAP</code> and <code>SIMULATE</code> options for simple forms of bootstrapping and random simulation.</td>
</tr>
<tr>
<td>VCV</td>
<td>Adds the <code>WINDOW</code> option.</td>
</tr>
</tbody>
</table>
Welcome to RATS!! RATS is a powerful, flexible tool for the statistical analysis of data. While its primary focus is on time series data, it may also be used very successfully for cross sectional and panel data sets.

RATS can be used for a variety of tasks, ranging from simple regressions, graphics and forecasting applications to very complex tasks that use the capabilities of RATS as a true programming language.

The object of this chapter is to get you up and running as quickly as possible. You should first read your Getting Started booklet to get your program installed, and to learn the basics of starting the program, working with the interface, and executing RATS instructions. Once you’ve done that, the best way to learn how to use RATS is to follow along with the tutorial in this chapter.

We can only give you a taste of what RATS can do, but as you will see, you can do quite a bit with what you learn in these first thirty-odd pages. After you finish, check out some of the other chapters and see how RATS can make your life easier.
Chapter 1: Basics

1.0 Basic Operating Instructions

Overview
To introduce you to RATS, we will take you step-by-step through a sample RATS session. We’ll describe the key RATS instructions, cover the important details of the syntax, and give you a feel for using the program.

First, you’ll want to read the *Getting Started* booklet included with your RATS software. This provides platform–specific instructions for installing and using the software. Our step-by-step tutorial in this chapter assumes you will be running RATS in *interactive* mode, so you’ll need to know how to do that. You can also use RATS in *batch* mode. Details on both methods are presented in the *Getting Started* booklet.

Start the Program
If you are ready to start the tutorial session, go ahead and start RATS in interactive mode, as described in the *Getting Started* book.

As we work through the tutorial, we will ask you to execute a variety of RATS instructions. *Getting Started* provides full details on executing instructions in interactive mode, but here’s a quick overview.

In general, you can execute an instruction by just typing it into the input window and hitting the <Enter> key.

You can also execute a *block* of instructions. First, select the block of text using the mouse (click and drag to select), or using the keyboard (put the cursor at the beginning of the block you want to execute, press and hold the <Shift> key, and then hit the down arrow key until you get to the end of the block you want to execute). The lines you select are displayed in inverse video. When you hit the <Enter> key or click on the “Run” icon, RATS will execute *all* of the selected instructions.

On Windows and UNIX/Linux systems, you can edit lines *without* executing them by putting RATS into “local” mode, either by clicking on the Ready/Local button or by hitting <Ctrl> plus the L key. In local mode, RATS functions simply as an editor. Clicking on the Ready/Local button again (or hitting <Ctrl>+L again) will put RATS back into “ready” mode, where the <Enter> key tells RATS to execute instructions.

On a Macintosh, just use the <Return> key, rather than the <Enter> key, when you want to insert a line without executing it.

You can access the menus using the mouse, or (on the PC) by pressing and holding the <Alt> key, and then pressing the first letter of the desired menu ( “F” for the File menu, “E” for Edit, etc.).
1.1 Getting Started: An Example

In the sections that follow, we will take you through many of the basic operations of RATS. We will be working with a sample dataset referenced in Pindyck and Rubinfeld’s *Econometric Models & Economic Forecasts* (1998) to demonstrate a variety of tasks, including multiple regression models and forecasting.

All of the sample commands that we will describe here are also provided for you in a RATS program file called BASICS.PRG. We recommend that you begin by entering the commands yourself as we discuss them. However, if you have a problem with a command, or want to learn more, you can refer to the BASICS.PRG file, which should be located in your RATS directory or folder. In any case, you should take a look at the BASICS.PRG file at some point, because it provides several examples in addition to those covered here.

The sample dataset is provided in several different file formats including an XLS (Excel) spreadsheet, text formats, and in RATS format. The following data series are provided on these files:

- **Rate**: Three-month treasury bill rate.
- **IP**: Federal Reserve index of industrial production, seasonally adjusted, index 1987=100.
- **M1**: Money Stock M1, billions of US dollars, seasonally adjusted.
- **M2**: Money Stock M2, billions of US dollars, seasonally adjusted.
- **PPI**: Producer Price Index, all commodities, index 1982=100, not seasonally adjusted.

Our primary goal will be to fit the following regression models to these data:

1. \[ Rate_t = \alpha + \beta_1 IP_t + \beta_2 (M1_t - M1_{t-3}) + \beta_3 PSUM_t + u_t \]

   where \( PSUM_t = \frac{\Delta PPI_t}{PPI_t} + \frac{\Delta PPI_{t-1}}{PPI_{t-1}} + \frac{\Delta PPI_{t-2}}{PPI_{t-2}} \)

2. \[ Rate_t = \alpha + \beta_1 IP_t + \beta_2 GRM2_t + \beta_3 GRPPI_{t-1} + u_t \]

   where \( GRM2_t = \frac{(M2_t - M2_{t-1})}{M2_{t-1}}, \quad GRPPI_t = 100\frac{(PPI_t - PPI_{t-1})}{PPI_{t-1}} \)

Our data were taken from Haver Analytics’ USECON database (available separately through Estima). The Haver series names for these are FTB3, IP, FM1, FM2, and PA, respectively, although we’ve converted them to the names shown above for our example. The series begin in January, 1959, and run through September, 1999. Some of the values are slightly different than the older data used in Pindyck and Rubinfeld. Here’s a portion of the data on the file:
Chapter 1: Basics

<table>
<thead>
<tr>
<th>Date</th>
<th>IP</th>
<th>M1</th>
<th>M2</th>
<th>PPI</th>
<th>RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1959:01</td>
<td>36.000</td>
<td>138.900</td>
<td>286.700</td>
<td>31.700</td>
<td>2.84</td>
</tr>
<tr>
<td>1959:02</td>
<td>36.700</td>
<td>139.400</td>
<td>287.700</td>
<td>31.700</td>
<td>2.71</td>
</tr>
<tr>
<td>1959:03</td>
<td>37.200</td>
<td>139.700</td>
<td>289.200</td>
<td>31.700</td>
<td>2.85</td>
</tr>
<tr>
<td>1959:04</td>
<td>38.000</td>
<td>139.700</td>
<td>290.100</td>
<td>31.800</td>
<td>2.96</td>
</tr>
<tr>
<td>1959:05</td>
<td>38.600</td>
<td>140.700</td>
<td>292.200</td>
<td>31.800</td>
<td>2.85</td>
</tr>
<tr>
<td>1959:06</td>
<td>38.600</td>
<td>141.200</td>
<td>294.100</td>
<td>31.700</td>
<td>3.25</td>
</tr>
<tr>
<td>1959:07</td>
<td>37.700</td>
<td>141.700</td>
<td>295.200</td>
<td>31.700</td>
<td>3.24</td>
</tr>
<tr>
<td>1959:08</td>
<td>36.400</td>
<td>141.900</td>
<td>296.400</td>
<td>31.600</td>
<td>3.36</td>
</tr>
<tr>
<td>1959:09</td>
<td>36.400</td>
<td>141.000</td>
<td>296.700</td>
<td>31.700</td>
<td>4.00</td>
</tr>
<tr>
<td>1959:10</td>
<td>36.100</td>
<td>140.500</td>
<td>296.500</td>
<td>31.600</td>
<td>4.12</td>
</tr>
<tr>
<td>1959:11</td>
<td>36.300</td>
<td>140.400</td>
<td>297.100</td>
<td>31.500</td>
<td>4.21</td>
</tr>
<tr>
<td>1959:12</td>
<td>38.600</td>
<td>140.000</td>
<td>297.800</td>
<td>31.500</td>
<td>4.57</td>
</tr>
</tbody>
</table>

Series Names

Series names in RATS can be from one to sixteen characters long. They must begin with a letter, and can consist of letters, numbers, and the _ and $ symbols. Generally, you'll want to choose variable names that are just long enough so you will know what each name represents. In most cases, your data files will include names for the series stored on the file, and RATS will use those names when it reads in the data. Thus, the series names on the file should conform to the restrictions described above.

For this example, we will use the variable names shown in the table above: RATE (for the 3-month T-bill rate), M1, M2, IP (industrial production data), PPI (Producer Price Index data).

What’s Ahead?

In the following pages, we will walk you through the basic steps required to estimate our models:

1. Read the data into RATS.
2. Examine the data to ensure that the data were read properly, and compute some basic statistics.
3. Do some simple data transformations.
4. Generate some time-series and X–Y scatter plots to examine the behavior of the series.
5. Run the regressions.

We will introduce the RATS instructions required for each step in order. These are among the most important instructions in RATS. We will also introduce several other important instructions, and show you how to do generate graphs, do some basic hypothesis testing, and compute fitted series and forecasts.
**RATS Wizards**

RATS provides many menu-driven “Wizards” that provide a convenient alternative to typing in commands yourself. Wizards are available for many common tasks, including reading in data, displaying graphs, performing regressions, and doing hypothesis tests. As we work through this example, we’ll point out situations where the Wizards provide an alternative method for handling various tasks.

However, to truly utilize the power of the RATS software, you will undoubtedly need and want to be able to type in and execute commands directly. With that in mind, we would strongly recommend that you take the time to work through this example by typing in the commands as they are described. By doing so, you will quickly become familiar with the basic syntax rules and the most important RATS commands. With this grasp of the fundamentals, you will be well-prepared to handle just about any econometrics task.

**An Advance Look at the Program**

Below is the sequence of instructions we’ll need to estimate these two regression models. We’ve put this here for a couple of reasons. First, we want to give you a sense of how RATS instructions look when you put them together. Second, those of you who want to jump right in can try typing in the entire program and running it to see what happens. If you do, we still recommend that you work through the next few pages, because they tell you why these instructions look and function the way they do, and teach you the fundamental “rules” of the RATS language. We will also discuss many other instructions and variations not included below. Also, remember that all of these instructions (with more examples not included in the text) are also provided in the BASICS.PRG example file.

```
* Define the date range, read in the data:
  calendar(m) 1959:1
  allocate 1996:2
  open data basics.xls
  data(format=xls,org=columns) / rate m1 m2 ip ppi

* Examine the data:
  table
  print

* Do some data transformations:
  set ppidiff = ppi - ppi{1}
  set mldiff = m1 - m1{3}
  set grm2 = (m2 - m2{1})/m2{1}
  set grppi = (ppi - ppi{1})/ppi{1}
  set pratio = ppidiff/ppi
  set ppisum = pratio + pratio{1} + pratio{2}
```
Chapter 1: Basics

* Estimate OLS regressions:
  linreg rate 1960:2 1980:12
  # constant ip m1diff ppisum

  linreg rate 1960:1 1995:8 resids_rate
  # constant ip grm2 grppi{1}

What Should You Type In?

Beginning on the next page, we put large pointers (➡️) in the margin next to all the instructions (or groups of instructions) that we want you to enter and execute. You'll notice that we also show you several other sample uses of these instructions. Some of these will work with the sample dataset, but others will not. For now, we recommend that you only type in the instructions marked with a pointer.

You may want to save your work periodically using the Save operation on the File menu. If you need to stop and come back to the tutorial later, just re-start RATS, open your saved file, and re-execute the commands you've already typed in. See Getting Started for details on saving and re-using program files.
1.1.1 Describing the Data to RATS

The CALENDAR Instruction

Before we can read in the data, we need to tell RATS a little about it. Since we are working with time series data, we tell RATS when our data begin and their frequency. We do this with the CALENDAR instruction. Type in the following instruction and hit <Enter> to execute the command.

```r
calendar(m) 1959:1
```

If you make a mistake, just retype the instruction exactly as shown above, and hit <Enter> again. Note that entering this instruction doesn’t produce any output or other results. This is true for many RATS instructions. Whenever we introduce an instruction that does produce output, we will tell you what to expect.

The M in parentheses stands for “monthly”. This is called an option, and this option tells RATS that we will be working with monthly data. The date 1959:1 tells RATS that our data begin in the first month (January) of 1959. This value is called a parameter. Most RATS instructions have one or more parameters.

The CALENDAR Wizard

If you select the Calendar operation from the Data menu, RATS displays the Calendar Wizard, which provides a convenient way to set the calendar. Just select the frequency and structure, and enter the starting date from the dialog box, and click on “OK”. RATS will generate the appropriate CALENDAR instruction and insert it into your input window (and execute the command if the Paste AND Execute switch on the Wizards menu is turned on). Note that you do not need to use the Calendar Wizard if you intend to use the Data Wizard (see Section 1.1.2). Here is a sample dialog box:
Aside: Alternative CALENDAR Settings

Many of the Pindyck and Rubinfeld examples apply only to data from November, 1959 on. In our example, we’re reading in all the data on our file (starting in January, 1959), so we’ll use “date parameters” to skip earlier observations when necessary. An alternative would have been to set our CALENDAR to start in November:

\[ \text{calendar}(m) \ 1959:11 \]

This specifies a monthly CALENDAR starting in the 11th month (November) of 1959. RATS can use the date information on the file to skip the data for January through October, and start reading in data beginning with the November 1959 observation.

While our example uses monthly data, RATS can support almost any frequency of data. Here are some sample CALENDAR instructions for other frequencies and starting dates:

- \[ \text{calendar}(q) \ 1960:2 \quad \text{Quarterly, starting 2nd quarter of 1960} \]
- \[ \text{calendar}(7) \ 1999:10:4 \quad \text{Daily (7 per week) starting Oct 4, 1999} \]
- \[ \text{calendar}(a) \ 1980:1 \quad \text{Annual, starting in 1980} \]

See the description of CALENDAR in the User’s Guide for details on the CALENDAR instruction and the many frequencies supported by RATS.

Important Note: If you have cross-sectional data, do not use a CALENDAR instruction. RATS will work fine without it.

The ALLOCATE Instruction

We’ve told RATS when our data starts, and the frequency. Now we need to say when our data end. We do this with the ALLOCATE instruction, which sets the default length of data series. The ALLOCATE command can actually be omitted, in which case RATS will try to guess the default ending date from the first data set you read. However, we generally recommend that you use ALLOCATE to set a default range except in cases where you don’t know how much data you have in your data file.

Our data set runs through February, 1996, so type in the following instruction:

\[ \text{allocate} \ 1996:2 \]

As with CALENDAR, the date “1996:2” is a single parameter, written in “date format”. To refer to a date in RATS, use

- \[ \text{year:period} \] for annual, monthly, and quarterly data (or any other frequency specified in terms of periods per year). In this example, we have set a monthly CALENDAR, so “1996:2” translates to the 2nd month (February) of 1996.
- \[ \text{year:month:day} \] for weekly, daily, etc.
With annual data, period is always one, so any reference to a date in annual data must end with :1. The :1 after the year is very important, because without it, RATS will assume you are specifying an integer number. See the “Aside” below for details.

**Use Four Digit Year Numbers!!**

RATS does allow you to use two-digit year references, but these are always interpreted as being in the 1900’s. For example, it will treat “41:1” as 1941:1. This allows programs written years ago to work, but we strongly recommend that you use four-digit year references in any new programs.

**Aside: Using Entry Numbers Instead of Dates**

You can also just use a number for the series length on **ALLOCATE**. For example,

```
calendar(a) 1922:1
allocate 20
```

is equivalent to:

```
calendar(a) 1922:1
allocate 1941:1
```

because 1941:1 is the twentieth entry given the **CALENDAR** setting. With time series data, you will usually use the dates. With cross-section data, you don’t have a **CALENDAR**, so you just start with

```
allocate (number of observations)
```

This flexibility is also why you need to remember to use the proper format when you specify the length as a date. Suppose you left the :1 off in our example:

```
calendar 1922:1
allocate 1941
```

RATS would assume you wanted series with one thousand nine hundred and forty one entries, equivalent to the year 3862! If you do this, RATS will issue a warning suggesting that you may have meant to use a date (like 1941:1) instead. If you really did mean to use an entry number, just ignore the message and go on, otherwise make the correction and re-execute the **ALLOCATE** instruction.

**Exceeding the ALLOCATE Length**

The length specified by **ALLOCATE** is not a binding constraint—you can define series beyond this default limit as needed by using explicit date/entry ranges.
1.1.2 Reading In the Data

The OPEN Instruction

At this point, you should have entered the following instructions:

```
calendar(m) 1959:1
allocate 1996:2
```

Now you need to tell RATS where your data are. You do this with an OPEN instruction. The data for our example file is supplied in four different types of files. For now, we'll work with the file called BASICS.XLS, which is an Excel-format spreadsheet file. We discuss the other files in Section 1.3. Type in the instruction:

```
open data basics.xls
```

This selects the file BASICS.XLS file for data input. If a data file isn't in the current directory, you can specify a directory (or folder) path as part of the filename:

```
open data c:\winrats\basics.xls
```

If your path includes names with spaces, enclose the entire path and filename in quotes. For example:

```
open data "C:\My Documents\RATS Projects\basics.xls"
```

The DATA Instruction

Now that you've set the data file, you can use the instruction DATA to read the data series from the file into memory. Type in the following instruction. We'll explain each part below. Note that there must not be a space between the word DATA and the left parenthesis.

```
data(format=xls,org=columns) / rate m1 m2 ip ppi
```

This reads the data into five series variables named RATE, M1, M2, IP, and PPI.

As noted earlier, the items in parentheses are called options. Most RATS instructions have options that allow you to supply additional information on how you want RATS to execute the instruction. Our DATA instruction above uses two options: FORMAT and ORGANIZATION (or ORG for short). You can abbreviate option names to three or more characters:

```
data(for=xls,org=col) / rate m1 m2 ip ppi
```

For now, we'll go over these two options quickly. We discuss them in more detail in Section 1.3.
The FORMAT Option

FORMAT gives the format of the file you are reading. As noted above, BASICS.XLS is an Excel spreadsheet file, so we use the option FORMAT=XLS on the DATA command.

Like parameters, most options have a default setting. The default option for FORMAT is FORMAT=FREE, which is used with plain text files containing only numbers and no labeling or dates. RATS supports a variety of different formats, including our own RATS data format. See Section 1.3 and Chapter 2 for details on the formats supported by RATS.

Note: RATS does not try to determine the format of the file based on the file name or extension. You must give it explicitly using the FORMAT option.

The ORGANIZATION Option

The ORG option describes how the data are arranged on the file. You have two choices: ORG=COLUMNS and ORG=ROWS.

You use ORG=COL if the series are arranged in columns running down the file, with one series per column. ORG=ROW indicates that the series are arranged in rows, generally with one series per row (although with some formats series can wrap to multiple rows).

The data in our spreadsheet file is stored as shown on page 4, with one series per column. Thus, we use ORG=COL to read this file.

The Entry Range

With any DATA instruction, RATS needs to know which entries of the series you want to read in. Recall our sample DATA instruction:

```
data(for=xls,org=col) / rate m1 m2 ip ppi
```

In this example, we use a slash symbol (/) as the first parameter. Used in this context, the slash tells RATS to use the current default range (1959:1 to 1996:2 as specified by the CALENDAR and ALLOCATE statements). You can also give an explicit range using either dates or entry numbers. The following are equivalent to the DATA instruction above:

```
data(for=xls,org=col) 1959:1 1996:2 rate m1 m2 ip ppi
```
```
data(for=xls,org=col) 1 446 rate m1 m2 ip ppi
```

All RATS instructions that deal with series allow you to set the range explicitly or use the “/” for the default range.

The Series Names

For files that include series names (which RATS requires for most formats), the series list parameter is optional—if you omit it, RATS will read in all the series on the file. If you do provide a list of series, RATS will read only those series. For
FREE format files and other formats lacking series names, you must supply a list of series names for the series on your DATA instruction. Without them, RATS would have no way to identify the data. The series list is also required when reading some database formats, to avoid accidentally reading a huge number of series.

The Data Wizards

One of the biggest tasks you will face in beginning any project is getting your data read in properly. The Data Wizards can greatly simplify this task, particularly if your data files contain date information that RATS can process. The Data Wizards combine all of the tasks described so far (setting the CALENDAR if necessary, defining the default series length, and reading in data).

RATS actually offers two “Data” Wizards—one for use with RATS format data files and one for all other file types. In either case, the Data Wizards walk you through the process of selecting a data file, specifying the desired frequency and starting date, and reading in the data.

To work with a spreadsheet file, text file, or DBF database file, select the operation Data (Other Formats) from the Data menu to get started. Choose the desired file format from the “Files of Type” field in the dialog box, and then select the file you want open.

To work with data from a RATS format file, first do File—Open RATSData and open the desired file from the dialog box. This will display a window listing the series on that file. Select (highlight) the series you want to read in and then do Data—Data (RATS Format).

Depending on the file format used, RATS may display a dialog box asking for information on the file (for example, whether the data runs in columns or in rows for a spreadsheet file).

Then it will display the Calendar Wizard dialog box (see page 7), allowing you to specify the frequency and starting date you wish to use for this session. When possible, the fields of this dialog box will be set automatically to match the information found on the data file. You can accept these settings, or change them if you want to use a different frequency or starting date.

Finally, for files that do not include series names, RATS will prompt you for names to assign to the series being read in (these can also be read in from a separate text file).

Once this is complete, RATS will generate the appropriate CALENDAR, ALLOCATE, OPEN DATA, and DATA commands to read in the selected data. These commands will be executed automatically if the Paste AND Execute switch is turned on.
1.1.3 Computing Statistics

The TABLE Instruction

Whenever you read data into RATS for the first time, you should do some checking to make sure you have the correct data and have read it in properly. One quick way is to use the instruction **TABLE**, which prints out some basic statistics. Enter the following:

```
> table
```

The **TABLE** instruction, without any options or parameters, tells RATS to produce statistics for all series in memory. Your output should look like this:

<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>RATE</td>
<td>446</td>
<td>6.05858744</td>
<td>2.77510466</td>
<td>2.27000000</td>
<td>16.29000000</td>
</tr>
<tr>
<td>M1</td>
<td>446</td>
<td>448.06255605</td>
<td>312.00002739</td>
<td>138.90000000</td>
<td>1152.30000000</td>
</tr>
<tr>
<td>M2</td>
<td>446</td>
<td>1577.37847534</td>
<td>1141.42636764</td>
<td>286.60000000</td>
<td>3673.10000000</td>
</tr>
<tr>
<td>IP</td>
<td>446</td>
<td>57.46704036</td>
<td>16.87177721</td>
<td>27.40000000</td>
<td>91.30000000</td>
</tr>
<tr>
<td>PPI</td>
<td>446</td>
<td>71.52242152</td>
<td>34.84734464</td>
<td>31.30000000</td>
<td>126.30000000</td>
</tr>
</tbody>
</table>

In verifying your data, most important to check are the number of observations (Obs) and the minimum and maximum values. Be sure they are reasonable given what you know about the data.

To produce a table for a selected set of series, you can follow the **TABLE** instruction with an entry range and a series list. For example:

```
> table / ppi rate
```

will compute statistics only for series **PPi** and **RATE**. Note that we used a slash to tell RATS to use the default range, just like we did on the **DATA** instruction. As with **DATA**, you can also specify a range explicitly:

```
> table 1970:1 1980:12 ppi rate
```

Using the Series Window

The **Series Window** offers another easy way to generate these same summary statistics. First, select the **Show Series Window** from the **Data** menu. You will see a window like this, displaying all the series currently in memory:
Note that the icons on the toolbar have changed to present operations applicable to this list of series. To generate the table of statistics, select (highlight) all of the series in the window and then click on the “Basics Statistics” icon on the toolbar, which looks like this:

This will generate a spreadsheet-style window showing the same results generated by the first `TABLE` instruction:

![Spreadsheet-style window showing statistics](image)

You can print the contents of this window, copy and paste the information into a spreadsheet program, or use the `File–Export` operation to export the contents of the window to any of several file formats.

The `Series Window` has many other uses, as you will see later in this chapter.

**The STATISTICS Instruction**

If you want a more detailed statistical report on a single series, you can use the instruction `STATISTICS`. Here’s a sample command and the resulting output:

```
statistics rate
```

Statistics on Series RATE  
Monthly Data From 1959:01 To 1996:02  
Observations 446  
Sample Mean  6.058587  
Standard Error  2.775105  
t-Statistic (Mean=0)  46.106212  
Skewness  1.186328  
Kurtosis (excess)  1.587381  
Jarque-Bera  151.440700  
Variance  7.701206  
Signif Level of Sample Mean  0.131405  
Signif Level  0.000000  
Signif Level (Sk=0)  0.000000  
Signif Level (Ku=0)  0.000000  
Signif Level (JB=0)  0.000000
The Univariate Statistics Wizard

You can use the *Univariate Statistics* wizard to get summary statistics and other information for a data series. When you select *Univariate Statistics* from the *Statistics* menu, RATS will display the following dialog box:

The first step is to select the series you want to use from the “Series” drop-down list. Then, turn on one or more of the “Basic Statistics”, “Extreme Values”, or “Autocorrelations” check boxes.

The “Basic Statistics” checkbox generates a *STATISTICS* command. The “Extreme Values” check box generates an *EXTREMUM* instruction, which reports the extreme values of the series. The “Autocorrelations” check box generates a *CORRELATE* instruction, which computes autocorrelations (and partial autocorrelations if you provide a series for the “Partial Corrs” field). You can turn on any combination of these three boxes.

The other fields allow you to select the range used for the computations and to select various options for the autocorrelation computations.
1.1.4 Displaying the Data

Displaying Series With PRINT

To display the contents of data series, you use the instruction `PRINT`. To see this in action, enter the instruction:

```
print / m1 m2
```

The output should look something like this:

```
ENTRY          M1              M2
1959:01        138.9           286.6
1959:02        139.4           287.7
1959:03        139.7           289.2
1959:04        139.7           290.1
1959:05        140.7           292.2
1959:06        141.2           294.1
1959:07        141.7           295.2
etc.
```

To see just the 1960 data for `M1`, you can enter:

```
print 1960:1 1960:12 m1
```

If you want RATS to print all the series in memory, just type:

```
print
```

Printing out your series is another good way to verify your data. It is also very useful when you are troubleshooting a program. When you get unexpected results, such as an error message complaining about missing values, try printing out the series involved. You may notice an error that had slipped through earlier checks.

You can use the `PICTURE` option on `PRINT` to control the number of decimal points used in the output. For example, the option `PICTURE="*.#"` tells RATS to display only one digit after the decimal:

```
print(picture="*.#") / rate
```

produces the output shown below. Note that the displayed values are rounded to the nearest tenth:

```
ENTRY         RATE
1959:01          2.8
1959:02          2.7
1959:03          2.9
1959:04          3.0
etc.
```
1.1.5 Data Transformations and Creating New Series

The SET Instruction

In most of your work with RATS, you will need to do at least a few data transformations, and you will often need to create new series from scratch. For our example, we need to define quite a few new series using the instruction SET. We’ll start by generating a couple of differenced series. Enter the following instructions:

\[ \text{set ppidiff} = \text{ppi} - \text{ppi}\{1\} \]
\[ \text{set m1diff} = \text{m1} - \text{m1}\{3\} \]

Let’s examine the first transformation. The \{1\} notation tells RATS to use the first lag of PPI in the transformation. This creates a new series called PPIDIFF, and sets it equal to the first difference of PPI:

\[ \text{PPIDIFF}_t = (\text{PPI}_t - \text{PPI}_{t-1}) \] for each entry \( t \) in the default entry range.

Note that PPIDIFF cannot be defined for 1959:1, because we do not have data for 1958:12, which would be the one period lag from 1959:1. RATS recognizes this, and defines entry 1959:1 of PPIDIFF to be a missing value.

Similarly, M1DIFF is defined as the three-lag difference of the series M1, so that

\[ \text{M1DIFF}_t = \text{M1}_t - \text{M1}_{t-3} \]. M1DIFF will be defined starting in 1959:4.

Compare PPI with PPIDIFF and M1 with M1DIFF:

\[ \text{print / ppi ppidiff m1 m1diff} \]

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>PPI</th>
<th>PPIDIFF</th>
<th>M1</th>
<th>M1DIFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1959:01</td>
<td>31.7</td>
<td></td>
<td>138.9</td>
<td></td>
</tr>
<tr>
<td>1959:02</td>
<td>31.7</td>
<td>0.0</td>
<td>139.4</td>
<td>NA</td>
</tr>
<tr>
<td>1959:03</td>
<td>31.7</td>
<td>0.0</td>
<td>139.7</td>
<td>NA</td>
</tr>
<tr>
<td>1959:04</td>
<td>31.7</td>
<td>0.1</td>
<td>139.7</td>
<td>0.8</td>
</tr>
<tr>
<td>1959:05</td>
<td>31.7</td>
<td>0.0</td>
<td>140.7</td>
<td>1.3</td>
</tr>
<tr>
<td>1959:06</td>
<td>31.7</td>
<td>-0.1</td>
<td>141.2</td>
<td>1.5</td>
</tr>
<tr>
<td>1959:07</td>
<td>31.7</td>
<td>0.0</td>
<td>141.7</td>
<td>2.0</td>
</tr>
<tr>
<td>1959:08</td>
<td>31.7</td>
<td>-0.1</td>
<td>141.9</td>
<td>1.2</td>
</tr>
<tr>
<td>1959:09</td>
<td>31.7</td>
<td>0.1</td>
<td>141.0</td>
<td>-0.2</td>
</tr>
<tr>
<td>1959:10</td>
<td>31.7</td>
<td>-0.1</td>
<td>140.5</td>
<td>-1.2</td>
</tr>
<tr>
<td>1959:11</td>
<td>31.5</td>
<td>-0.1</td>
<td>140.4</td>
<td>-1.5</td>
</tr>
<tr>
<td>1959:12</td>
<td>31.5</td>
<td>0.0</td>
<td>140.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>1960:01</td>
<td>31.6</td>
<td>0.1</td>
<td>140.0</td>
<td>-0.5</td>
</tr>
<tr>
<td>1960:02</td>
<td>31.6</td>
<td>0.0</td>
<td>139.9</td>
<td>-0.5</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice that RATS shows the missing observation as NA, for “Not Available”.

\[ /orightshadlft \]

\[ /orightshadlft \]
Chapter 1: Basics

Next, we’ll create some quarter to quarter growth rates, again using the “(L)” lag notation, and the “/ ” division operator:

$$\text{set grm2} = (m2 - m2(1))/m2(1)$$
$$\text{set grppi} = (ppi - ppi(1))/ppi(1)$$
$$\text{set grip} = (ip - ip(1))/ip(1)$$

Now, we’ll create some annualized growth rates. The “**” operator is the exponentiation (power) operator in RATS:

$$\text{set anngrppi} = 100*( (ppi/ppi(1))^{12} -1.0)$$
$$\text{set anngrip} = 100*( (ip/ip(1))^{12} -1.0)$$

We also need to create a three-period moving average term. First, we define \text{PRATIO} as the ratio of \text{PPIDIFF} (the first difference of \text{PPI} that we created above) to \text{PPI}. Then, we define \text{PPISUM} as the sum of the current and two lags of \text{PRATIO}. This could be done with a single \text{SET}, but is easier to read or modify this way:

$$\text{set pratio} = \text{ppidiff/ppi}$$
$$\text{set ppisum} = \text{pratio} + \text{pratio}(1) + \text{pratio}(2)$$

Note that there are specialized instructions that can be used for some of these operations, such as \text{DIFFERENCE} and \text{FILTER}, but \text{SET} is the most important because it is the most flexible.

Finally, try the following

$$\text{set trend} = t$$
$$\text{set trendsq} = t^{**2}$$
$$\text{print} / \text{trend} \text{trendsq}$$

As you can see, this defines a simple time trend series and a squared trend series. In any \text{SET} instruction, the variable \text{T} represents, in turn, each entry number of the series you are setting. For entry one (the starting date on your \text{CALENDAR}), \text{T}=1. For the tenth entry, \text{T}=10.

You can also use the \text{T} as a subscript for series. For example, you can also compute the \text{GRM2} series with:

$$\text{set grm2} = (m2 - m2(t-1))/m2(t-1)$$

We show you how to do many other types of transformations in Section 1.7. For now, we’ll carry on with our example.
Data Transformation and Related Wizards

RATS offers several Wizards for doing transformations, creating dummy variables, and other data-related operations. The *Transformations Wizard* is probably the most versatile. Selecting *Transformations* from the *Data* menu displays the following:

![Basic Transformations Wizard](image)

Use the “Create” field to type in or select the name of the series you want to create or redefine. The “By/As” field controls the type of transformation. Select “General–Input Formula” if you want to enter your own formula for the transformation, or use one of the pre-defined transformations, including difference, log and square root.

For example, to create `PPIDIFF` as the first difference of `PPI`, you would type in `PPIDIFF` in the “Create” field, select “Difference” from the “By/As” field, and select `PPI` in the “From” drop-down list:

![Basic Transformations Wizard](image)

The *Trend/Seasonals/Dummies Wizard*, *Differencing Wizard*, and *Filter/Smooth Wizard* offer similar series creation and transformation capabilities.
1.1.6 Graphing the Data

Time Series Graphs: The GRAPH Instruction

Now we’ll do some graphs. Type in the following lines:

```
graph(key=upleft) 3
# rate
# ip
# ppi
```

This graphs the series RATE, IP, and PPI, including a key in the upper–left corner of the graph box. Your graph should look like this:

As you can see, GRAPH generates time series graphs. The parameter “3” tells RATS that you want to graph three series. The next three lines are called *supplementary cards*. Many RATS instructions use supplementary cards to get additional information from the user—usually lists of series or equations. Supplementary cards always begin with the # character.

When you are done viewing the graph, close the graph window or click on the input window to bring it back to the front.
With `GRAPH`, you use one supplementary card for each series in the graph. Here’s another example:

```
graph(key=below) 2
  # m1
  # m2
```

`GRAPH` is an extremely flexible instruction with a wide variety of options for fine-tuning the appearance of a graph. If you want to get a bit fancier, try the following:

```
graph(key=upleft,header="Real Money Supply (M1 and M2)") 2
  # m1
  # m2
```

This adds a header to the graph. See Chapter 3 for more detail on producing graphs with RATS.

**X-Y Scatter Plots: The SCATTER Instruction**

We’ve looked at creating time series (data vs. time) graphs with `GRAPH`. Now let’s try graphing one series against another. For this, we use the instruction `SCATTER`. The `SCATTER` instruction is very similar to `GRAPH` except that each supplementary card gives a *pair* of series, the x-series and y-series respectively. Type in the instruction:

```
scatter 1
  # grppi rate
```

Or, if you prefer to use some fancier labelling:

```
scatter(vlabel="Interest Rate",hlabel="PPI Growth Rate", $
  header="Interest Rates vs. PPI Growth") 1
  # grppi rate
```

(The “$” symbol here is used to tell RATS that a command continues on the next line.) This will generate a high-resolution X-Y plot, with X=GRPPi and Y=RATE. We can see that interest rates below about 10 percent are generally associated with a mix of small-magnitude positive and negative movements in the PPI, while larger interest rates generally coincide almost exclusively with positive movements in PPI.

The resulting graph is shown on the next page.
To give you an idea of what is possible, here’s a more complex example that creates a vector of “labels”, and then graphs the series using different colored symbols for different ranges of data. The || ... || notation allows you to supply a vector or array in an expression—see Section 4.6 for details. Also, see Section 1.1.10 for details on the COMPUTE instruction:

```plaintext
scatter(klabel=keys,key=below,vlabel="Interest Rate", $hlabel="Monthly Growth Rate of PPI") 3
  # grppi rate  * 1978:12  1
  # grppi rate 1979:1 1983:12  3
  # grppi rate 1984:1  *  4
```
The Graphics Wizard

You can also use the *Graph* operation on the *Data* menu to generate time series plots. Selecting *Graph* brings up the following dialog box:

To create a standard graph, select one or more series in the “Series” column and click on the “<< Add <<“ button. This adds the series to the “Base Series” list. Use the “Style” field under the “Base Series” list to choose the style used to display these series. If you decide you want to remove a series from the list, just select the series and click “>> Remove >>”.

The tabbed box at the bottom allows you to control many aspects of the graph. In the example above, we’ve added “Industrial Product” as a header. You can use other fields to control various aspects of the graph key and the axes. When you are ready, click “OK” to generate the **GRAPH** command and display the graph.
The **GRAPH** instruction also allows you to create “overlay” or “two-scale” graphs, where some of the series are graphed using one scale (displayed on the vertical axis on the left side of the graph), while other series are graphed using a second scale (displayed on the right side of the graph). To create an overlay graph using the Wizard, just select the series you want to graph using the second scale from the “Series” list, and click on the “>> Add >>” button to the right of the list. This will add the series to the “Overlay Series” list. Use the Style button under this list to choose the style used for the overlaying series.

See Chapter 3 and the **GRAPH** section in the *Reference Manual* for details.
1.1.7 Estimating Regressions

Our Equations

Recall that we want to estimate two regression equations:

1. \( \text{Rate}_t = \alpha + \beta_1 \text{IP}_t + \beta_2 (M1_t - M1_{t-3}) + \beta_3 \text{PSUM}_t + u_t \)

where \( \text{PSUM}_t = \frac{\Delta \text{PPI}_t}{\text{PPI}_t} + \frac{\Delta \text{PPI}_{t-1}}{\text{PPI}_{t-1}} + \frac{\Delta \text{PPI}_{t-2}}{\text{PPI}_{t-2}} \)

2. \( \text{Rate}_t = \alpha + \beta_1 \text{IP}_t + \beta_2 \text{GRM2}_t + \beta_3 \text{GRPPI}_t + u_t \)

where \( \text{GRM2}_t = (M2_t - M2_{t-1})/M2_{t-1} \), \( \text{GRPPI}_t = 100(\text{PPI}_t - \text{PPI}_{t-1})/\text{PPI}_{t-1} \)

We’ve done the data transformations, so we are ready to estimate the models.

Estimating a Linear Regression: LINREG

The basic instruction for estimating linear regressions is **LINREG**. The following command will estimate equation (1), which is taken from Example 4.2 from the 3rd (1991) edition of Pindyck and Rubinfeld:

```r
linreg rate 1960:2 1980:12
# constant ip mldiff ppisum
```

This regresses the dependent variable \text{RATE} on the independent variables \text{IP}, \text{M1DIFF}, and \text{PPISUM}, and includes a constant term (intercept) in the regression (\text{CONSTANT} is a special series name, used for a column of ones in a regression).

We are also telling RATS to limit the regression to the range from February, 1960 through December, 1980, to match the Pindyck and Rubinfeld examples. Note that the results will be somewhat different from those shown in the book, because the data are not identical due to revisions made since Pindyck and Rubinfeld extracted their data. **LINREG** estimates using ordinary least squares (OLS).

The output produced by this command is shown on the following page.
Chapter 1: Basics

**Linear Regression - Estimation by Least Squares**

**Dependent Variable RATE**

Monthly Data From 1960:02 To 1980:12

<table>
<thead>
<tr>
<th>Usable Observations</th>
<th>251</th>
<th>Degrees of Freedom</th>
<th>247</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centered $R^2$</td>
<td>0.674259</td>
<td>$R$ Bar **2</td>
<td>0.670303</td>
</tr>
<tr>
<td>Uncentered $R^2$</td>
<td>0.945263</td>
<td>$T \times R^2$</td>
<td>237.261</td>
</tr>
<tr>
<td>Mean of Dependent Variable</td>
<td>5.4658167331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std Error of Dependent Variable</td>
<td>2.4613624744</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum of Squared Residuals</td>
<td>493.35923057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regression F(3,247)</td>
<td>170.4239</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Significance Level of F</td>
<td>0.00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>-440.96454</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Durbin-Watson Statistic</td>
<td>0.189315</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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>-2.59825936</td>
<td>0.50111595</td>
<td>-5.18495</td>
<td>0.00000045</td>
</tr>
<tr>
<td>2. IP</td>
<td>0.16946855</td>
<td>0.01273584</td>
<td>13.30643</td>
<td>0.00000000</td>
</tr>
<tr>
<td>3. M1DIFF</td>
<td>-0.10954083</td>
<td>0.04099987</td>
<td>-2.67174</td>
<td>0.00804792</td>
</tr>
<tr>
<td>4. PPISUM</td>
<td>31.42821440</td>
<td>7.30011996</td>
<td>4.30516</td>
<td>0.00002408</td>
</tr>
</tbody>
</table>

Examine the Results

Although we won't go into much detail here, we do want to mention a few important points about the regression output. See Section 5.1 for a full description of regression output, including formulas.

Goodness of Fit Measures

You will notice there are four versions of the $R^2$ statistic: the centered $R^2$, the $\overline{R}^2$ ($R^2$ adjusted for degrees of freedom), the uncentered $R^2$, and $TxR^2$ (number of observations x uncentered $R^2$). The centered and adjusted $R^2$ are typically the only ones of interest.

RATS also displays the mean and standard error of the dependent variable. These are simply statistics on the dependent variable, and tell you nothing about the accuracy of the regression model. They are the same values you would get by doing a **STATISTICS** instruction on the **RATE** series over the same range.

Next are the standard error of estimate and sum of squared residuals. These are the usual regression statistics.

Test Results

RATS reports probability results for test statistics by giving the marginal significance level, often called the $P$-value. This is the probability that a random variable with the appropriate distribution will exceed the computed value (in absolute value for Normal and $t$). You reject the null if the marginal significance level is smaller than the
significance level you want. For instance, a marginal significance level of .03 would lead you to reject the null hypothesis if you are testing at the .05 level (.03 < .05), but not if you are testing at the .01 level.

If you want to do a one-tailed test for a Normal or \( t \)-distributed statistic, just divide the reported marginal significance level by two.

**Regression \( F \) and Significance Level**
This is the result of an \( F \)-test for the hypothesis that all coefficients (excluding \texttt{CONSTANT}) are zero. The numbers in parentheses after the \( F \) are the degrees of freedom for the numerator and denominator, respectively. Here, the marginal significance level is basically zero, strongly rejecting the hypothesis of zero coefficients.

**Log Likelihood**
The log-likelihood is another goodness-of-fit measure. One of the advantages of the log likelihood value is that it can be used to compare slightly different model specifications estimated using the same data.

**Tests for Serial Correlation**
The Durbin–Watson statistic tests for first-order serial correlation in the residuals. The ideal result is 2.0, which indicates a lack of serial correlation. Our result of approximately 0.19 strongly suggests that our residuals are serially correlated. We will look at improving the fit using alternative models below, and on page 43.

**Coefficient Table**
The table lists all the variables in the regression, including the constant if you have one in the model. The \texttt{Coeff} column lists the estimated coefficients for each variable (\( \alpha, \beta_1, \beta_2, \) and \( \beta_3 \) in our notation). The \( t \)-statistic is the ratio of a coefficient to its standard error. The significance level is for a two-tailed test for a zero coefficient. Again, our significance levels are very small, suggesting that these variables do have explanatory power.

**Another Example**
Now estimate our second equation (2) with the following (this is Example 4.2 as it appears in the 4th (1998) edition of Pindyck and Rubinfeld):

```
linreg rate 1960:1 1995:8 resids_rate
# constant ip grm2 grppi{1}
```

This regresses \texttt{RATE} on \texttt{IP}, \texttt{GRM2} (growth rate of \texttt{M2}) and the one-period lag of \texttt{GRPPI} (growth rate of \texttt{PPI}). Notice that we’ve added an additional parameter argument after the date range. This parameter allows us to save the residuals in a series. Here, we have supplied the name “\texttt{resids\_rate}”. RATS will create this series and fill it with the regression residuals. Here’s the output from that regression:
Linear Regression - Estimation by Least Squares
Dependent Variable RATE
Monthly Data From 1960:01 To 1995:08
Usable Observations 428 Degrees of Freedom 424
Centered R**2 0.220016 R Bar **2 0.214497
Uncentered R**2 0.866749 T x R**2 370.969
Mean of Dependent Variable 6.1453504673
Std Error of Dependent Variable 2.7927192256
Standard Error of Estimate 2.4751482014
Sum of Squared Residuals 2597.5760545
Regression F(3,424) 39.8669
Significance Level of F 0.00000000
Log Likelihood -993.19280
Durbin-Watson Statistic 0.173981

<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>0.92623530</td>
<td>0.57345098</td>
<td>1.61520</td>
<td>0.10701226</td>
</tr>
<tr>
<td>2. IP</td>
<td>0.07035206</td>
<td>0.00789203</td>
<td>8.91432</td>
<td>0.00000000</td>
</tr>
<tr>
<td>3. GRM2</td>
<td>140.74624080</td>
<td>35.95855432</td>
<td>3.91412</td>
<td>0.00010568</td>
</tr>
<tr>
<td>4. GRPPI{1}</td>
<td>100.89326383</td>
<td>17.41690025</td>
<td>5.79284</td>
<td>0.00000001</td>
</tr>
</tbody>
</table>

Regression Range

RATS reports the range as 1960:1 to 1995:8, as requested by our LINREG command. What if we just used a slash for the date range, as we did on the DATA instruction?

\[ \text{linreg rate / resids_rate} \]
\[# constant ip grm2 grppi{1} \]

This model uses a one period lag of GRPPI, but, since GRPPI and GRM2 aren’t defined until 1959:2, the one-period lag of GRPPI isn’t defined until 1959:3. When you execute the LINREG, RATS will scan the data, determine that 1959:3 is the earliest possible starting date, and begin the regression with that data point. Its ability to handle such entry range issues automatically is a very powerful feature.

Evaluating the Model

The quality of this fit is very similar to our first regression. All the coefficients seem significant, but the centered $R^2$ statistic is fairly small, and the Durbin–Watson is troubling, indicating that our model is not providing a sufficiently acceptable fit. In Section 1.4, we look at some other models that are more promising.

Regression Wizard

You can use the Regression Wizard to perform a variety of regressions. We’ll just provide a brief discussion here—see the Help menu in RATS for full details. Selecting the Wizard (Regressions from the Statistics menu) will a display a dialog box like this:
In the example above, we have selected RATE as the dependent variable from the drop down list. We are using the default setting for the “Method” list, which is “Least Squares”. Additional methods available include two-stage least squares, GMM, AR1, and robust and stepwise regressions.

To add explanatory variables, you can type the variables directly into the “Explanatory Variables” field, just as you would type them on a LINREG supplementary card. Or, click on the “list” button (…) to display the dialog box shown on the next page.
Chapter 1: Basics

As with the series list in the Graphics Wizard, you can add series to the regression by selecting them in the “Available” list and clicking on the “Add” button. To get the one period lag of GRPPI, we selected GRPPI from the list and clicked on “Add Lags”. This brings up a small dialog box, where we entered “1” in the lag list and clicked on OK. At this point, we can just click on OK in the main dialog to generate the appropriate regression instruction.

Note that if you have already done a regression (either using the Wizard or by typing in a regression command yourself) and then select the Regression Wizard, the dialog box will be “pre-loaded” with the settings from that regression. This makes it easy to make minor modifications to a preceding regression. If you instead want to start fresh, just click on “Clear Settings” to reset all the fields to default values.

Again, see the Help for complete details on the Regression Wizard.

Where Do We Go Now?

We’ve finished our simple regressions. In the next few pages, we’ll continue our analysis by doing some simple hypothesis testing and forecasting, looking at some alternative estimation techniques, and introducing some of the other capabilities of the RATS software. All of the commands we discuss are included on BASICS.PRG.
Chapter 1: Basics

1.1.8 Hypothesis Testing

Exclusion Restrictions

Let’s try another regression. Enter the instruction:

```r
linreg rate
# constant rate{1 to 6}
```

This fits a simple autoregressive model, by regressing \( \text{RATe} \) on a constant and lags 1 through 6 of \( \text{RATE} \). (TO is a special word in RATS, used for abbreviating lists.) The results seem reasonable, although the \( t \)-statistics suggest that the longer lag terms may not be significant. Suppose you want to test the null hypothesis that lags 4, 5, and 6 can jointly be excluded from the regression, that is, that the coefficients for all three are zero. You do this with the instruction `EXCLUDE`. Type in the following:

```r
exclude(title="Exclusion test on lags of RATE")
# rate{4 to 6}
```

Note that, like `LINREG`, `EXCLUDE` uses a supplementary card to list the variables. It also supports the same form of lag notation. Here are the results from the \( F \)-test:

**Exclusion test on lags of RATE**

**Null Hypothesis : The Following Coefficients Are Zero**

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Lag(s) 4 to 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{RATE} )</td>
<td>( F(3,433)= 0.87529 ) with Significance Level 0.45382609</td>
</tr>
</tbody>
</table>

The relatively large value for the significance level suggests that we can accept our null hypothesis that the coefficients on lags 4 through 6 are zero.

Other Restrictions Using TEST and RESTRICT

`EXCLUDE` only does exclusion restrictions: tests for coefficients being equal to zero. The `TEST` instruction does more general equality restrictions. In its most common form, you use the `TEST` instruction by supplying one supplementary card listing the coefficient number or numbers that you want to test (the coefficient numbers are the order in which coefficients appeared in the most recent regression), and a second supplementary card listing the restricted values for each coefficient being tested. For example, the equivalent to the `EXCLUDE` test on lags 4 though 6 of \( \text{RATE} \) would be:

```r
linreg rate
# constant rate{1 to 6}
test
# 5 6 7
# 0.0 0.0 0.0
```
This performs a joint test of the hypothesis that the 5th, 6th, and 7th coefficients in
the preceding regression (lags 4 though 6 of RATE) are each equal to zero. If you
wanted to test the hypothesis that the RATE(1) coefficient was equal to 1.0 and
RATE(2) was equal to 0.0, you would do:

```
test
  # 2   3
  # 1.0 0.0
```

which produces the result:

```
F(2,433)= 24.55334 with Significance Level 0.00000000
```

This hypothesis is overwhelmingly rejected.

The **RESTRICT** instruction (and the nearly identical **MRESTRICT**) test whether a
linear combination of coefficients is equal to a given value. The *P&R* example we are
using doesn’t really lend itself to this kind of test, but we will go ahead and use
**RESTRICT** to implement a coefficient equality test so you can see how it works. We
want to test a hypothesis of the form:

\[ H_0: \beta_1 = \beta_2 \]

Because **RESTRICT** tests whether a linear combination of coefficients is equal to some
value, we need to rewrite this as:

\[ H_0: \beta_1 - \beta_2 = 0 \]

We’ll re–run our second regression model and test whether the GRM2 coefficient is
equal to the GRPPI(1) coefficient:

```
linreg rate 1960:1 1995:8
  # constant ip grm2 grppi{1}
restrict 1
  # 3   4
  # 1.0 -1.0 0.0
```
Regression Tests Wizard
You can handle many regression-based hypothesis tests via the *Regression Tests Wizard*. When you select *Regression Tests* from the *Statistics* menu, RATS brings up the dialog box shown below:

Choose “Exclusion Restrictions” to do simple exclusion tests, “Other Constant Restrictions” to test coefficients against specific values, or “General Linear Restrictions” to test restrictions on linear combinations of coefficients.

Each choice will bring up a dialog box specific to that choice. See the RATS Help for details on each of the choices.

More on Hypothesis Testing
Chapter 6 provides full details on hypothesis testing, including detailed information on the instructions described above, as well as the specialized instructions *SUMMARIZE* and *RATIO*. You will also find out how to use RATS’ abilities to compute your own test statistics and marginal significance levels, and how to implement serial correlation and heteroscedasticity tests, Chow tests, Hausman tests, unit root tests, and many more.
1.1.9 Fitted Values and Forecasting

Fitted Values and Static Forecasts

Now we want to compute some forecasts for the interest rate series. The simplest type of forecast is for a regression equation with no lagged dependent variables. In RATS, you can do this with the instruction **PRJ** (short for PRoJect). Rerun the estimation of equation (2) by typing in (or re-executing):

```
linreg rate 1960:1 1995:8 resids_rate
# constant ip grm2 grppi{1}
```

We can compute fitted values for the estimated range by simply doing:

```
prj fitted
```

**PRJ** computes fitted values for the dependent variable of the most recent regression, using the actual values of the regressors and the estimated coefficients. By default, it computes fitted values over the same range as the preceding regression. To compute forecasts for a different range, just specify the desired dates or entry numbers. Here, we compute forecasts for January 1995 through February 1996, as discussed in Example 8.2 of Pindyck and Rubinfeld (1998):

```
prj forecast 1995:1 1996:2
print / rate forecast
```

This computes the forecasts and saves them in a new series called *FORECAST*, which will only be defined over the range 1995:1 through 1996:2 and will be NA elsewhere. We can graph the actual and fitted data with the following command:

```
graph(key=below,header="Actual vs. Fitted") 2
# rate 1994:7 1996:2
# forecast
```

**PRJ** always applies to the most recent regression, and can only be used for single-equation linear models. For more general forecasting purposes, RATS provides the instructions **UFORECAST** and **FORECAST**.

**UFORECAST** is the simplest to use of the three, as it is limited to forecasting single equations. **FORECAST** is more general—it can forecast one equation or many, and can also be used to forecast non-linear models. By default, both **UFORECAST** and **FORECAST** do *dynamic* forecasting. That is, when computing multi-step forecasts, they use the forecasted values from initial steps as the lagged dependent variable values in computing the later forecast steps. Both can also do *static* forecasting if you use the
STATIC option. Static forecasting means that actual values are used for any lagged dependent variable terms in the model.

These instructions are discussed in detail in Chapters 8 and 9 and in the Reference Manual. Here, we’ll look at a quick example to show you how they work.

First, we estimate equation 2 again, but this time we use the DEFINE option to save the estimated model as an EQUATION variable called IRATEEQ. This saves the structure of the equation itself (that is, the names of the dependent and right-hand-side variables) and the estimated coefficients:

```
linreg(define=irateeq) rate 1960:1 1995:8
# constant ip grm2 grppi{1}
```

Once again, we’ll compute forecasts for 14 periods, starting in January 1995 and ending in February 1996. Here is how you can do that using UFORECAST:

```
uforecast(equation=irateeq) olsfore 1995:1 1996:2
```

The EQUATION option tells RATS the name of the equation we want to forecast. The first parameter provides the name of the series into which we want to store the forecasts—here we’ve supplied a new series name, OLSFORE. RATS will create this series and store the forecasts in it. Finally, we provide the starting and ending dates of the range we want to forecast.

If you prefer, you can use the Single-Equation Forecasts wizard on the Statistics menu to generate the appropriate UFORECAST instruction.

And here is how you would do the same operation with FORECAST:

```
forecast(from=1995:1,steps=14) 1
# irateeq olsfore
```

The options and parameters tell RATS to forecast one equation for 14 steps, starting in 1995:1. The supplementary card tells RATS to use the equation IRATEEQ, and to store the computed forecasts in OLSFORE.

Next, we’ll graph our actual and forecasted values. We’ll use the start and end parameters on the first supplementary card to limit the range of data on the graph:

```
graph(header="Interest Rate and Forecast",key=below) 2
# rate 1994:7 1996:2
# olsfore
```
1.1.10 Scalar and Matrix Computations

Most of the instructions we’ve introduced so far work with data series. While these may be sufficient for many statistical tasks, other things that you do may require using integer-valued counter variables, computing scalar real-valued test statistics, doing matrix arithmetic or string manipulation, and so on. Fortunately, RATS is very proficient at such tasks. Chapter 4 discusses these topics in detail, so for now we’ll just give you a brief idea of what the program can do.

The COMPUTE Instruction

You’ve already seen the SET instruction, which does computations involving data series. The key instruction for doing scalar and matrix computations as well as string/label manipulation is the COMPUTE instruction. The basic syntax is simply:

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

For example:

\[ \text{compute \ a} = 1.0 \]
\[ \text{display \ a} \]

The COMPUTE instruction defines a real–valued scalar variable called \( A \), and stores the value 1.0 in it. The DISPLAY instruction displays the contents of the variable.

Variables Defined by RATS

An instruction like LINREG or STATISTICS produces quite a few numbers. And you may need to input some of those numbers into some other calculation. Fortunately, you can retrieve that information (to full machine-precision, not just the number of decimal places shown) using variables which RATS defines.

For instance, the sum of squared residuals from LINREG is in the variable \%RSS; the variance computed by STATISTICS is in \%VARIANCE; the number of observations from almost any estimation instruction is put into \%NOBS. The RATS variables (other than \( T, I \) and \( J \), which have special uses as counters) all start with the \% character, so they won’t conflict with variable names that you might want to use yourself.

See Section 4.2 for more on the use of these. Also note that a full list of these variables (broken down by categories) is available by selecting the RATS Variables operation on the Wizards menu. You will also find them listed in Appendix B of the Reference Manual.
Functions

In addition to the instructions we’ve been describing so far (such as CALENDAR, LINREG, and COMPUTE), RATS also offers a large selection of built-in functions. Functions are not stand-alone objects—you can’t “execute” a function. Instead, functions are used as part of an instruction, such as on the right hand side of a COMPUTE or SET expression, or in formulas used to define non-linear models (see Chapter 7). The functions available in RATS range from mathematical transformations, such as taking logs or square roots, to matrix operations such as transposition and inversion, to complex programming functions that allow you to manipulate equations or implement conditional statements.

Functions are discussed in more detail in Chapter 4. Also, see Section 2 of the Reference Manual for a complete listing of functions available in RATS. For now, we’ll just look at a couple of simple examples.

Suppose that (for instructional purposes) you want to do a least squares regression using matrix algebra rather than the LINREG instruction. If $X$ is a rectangular matrix of the dependent variables (one series per column) and $Y$ is a vector containing the dependent variable, the formula for computing the estimated coefficient vector is:

$$\hat{\beta} = (X^t X)^{-1} (X^t Y)$$

To implement this in RATS, you can use the MAKE instruction to copy the data from your series into the “X” and “Y” arrays, and then use COMPUTE, along with the INV() (matrix inversion) and TR() (matrix transposition) functions, to compute the coefficient vector: If you want to try this, enter the following:

```
make xmat 1960:1 1995:8  
# constant ip grm2 grppi{1}  
make ymat 1960:1 1995:8  
# rate  
compute invXX = inv(tr(xmat)*(xmat))  
compute XY = tr(xmat)*ymat  
compute beta = invXX*XY  
display beta
```
1.2 Learning More About RATS

Our Example

This concludes our step-by-step introduction to RATS. We could do more with the data, but you probably want to get on with your own projects. We’ve covered the most important RATS instructions, so you may know enough to begin working with your own data. If you skipped sections, it’s a good idea to go back over them at some point to see what you’ve missed.

Also note that the BASICS.PRG example program file includes additional examples demonstrating serial correlation correction and simple non-linear estimation techniques. See Sections 1.4 and 1.5 later in this chapter for details on these topics.

Using the Manual

There is almost no limit to what you can do with RATS. It is only a question of learning how. You may need more information on the instructions we’ve already covered. Or, you may be interested in other types of analysis. With that in mind, we would like to offer some suggestions on how to find out what you need to know.

Yes, there are a lot of instructions, and the manuals are large. Don’t worry! RATS is a very powerful and general program, and you may only need to use a fraction of its capabilities, at least in the beginning. Some suggestions for learning about RATS:

- First, look through the rest of Chapter 1. You’ll find a variety of information, including more details on topics we’ve already introduced, and some general information you will need to use RATS effectively and efficiently.
- Take a look at the Reference Manual to get a feel for how it is organized and how the details on each instruction are presented. Look up some instructions you’ve already seen, like DATA and LINREG. When you see something new in an example, use the Reference to find out exactly what is going on.
- Take a look at Chapter 2, which deals with reading and writing data files.
- Use the Table of Contents and the Index! They are probably the quickest way to find the information you want.
- Use the example program files included with RATS. All of the full examples from the manual are provided. See the “Procedures and Examples” PDF file included with the program for a complete list.
- Appendix D lists all the RATS instructions, grouped by their general function. This is a quick way to find instructions you may need to for a particular task.
- You can use the help facility as a quick reference for the interface and all of the instructions.
- See the procedures included with RATS, as well as the many procedures and example programs available on our Web site (www.estima.com). Many complex tasks have already been implemented as RATS procedures.

Most important, don’t be afraid to try something.
1.3 More About Reading Data

File Formats

As noted earlier, RATS supports several data file formats, and we have supplied versions of the sample data from Section 1.1 in four of these formats. Here’s a list of those files:

- **BASICS.RAT** This is a RATS format data set. RATS format is our own file format, which we describe in Chapter 2.

- **BASICS.XLS** This is the Excel-type spreadsheet file we used in the example. You can open and view this file with virtually any recent spreadsheet program. The top row of the file contains the variable names, the first column contains dates (entered as numbers with Excel date formats), and the remaining five columns contain the data, one column per variable.

- **BASICS.PRN** This is a PRN format or “labeled” text file. It contains the same information as the XLS file, including series names and dates (which are labels enclosed in quote marks), but is a standard ASCII text file that can be opened by RATS or any other text editor or word processor.

- **BASICS.DAT** This is a free-format text file. It contains only the five columns of numbers, one column per variable. It does not contain any other characters (like variable names or headers).

Except for the free-format file, these files all include series names along with the data. This gives you a lot of flexibility when you read in the data:

- You can include the names of some or all the series on your `DATA` instruction (these names must match those on the file). RATS will read only those series you request. You can list the series names in any order.
- You can omit the series names entirely. RATS will read all the series on the file.
- All of the file formats that include series names can also contain dates for the observations on the file. This allows you to select a subset of entries, or even change the frequency of the data while you are reading it in (see Section 2.3).

**The FORMAT Option**

Each of the formats listed above requires a different `FORMAT` option (see Chapter 2 for information on other formats).

Here are the commands you would use to read each of the sample files:
For BASICS.RAT: \[ \text{data(format=rats)} / \text{rate m1 m2 ip ppi} \]

For BASICS.XLS: \[ \text{data(format=xls,org=columns)} / \text{rate m1 m2 ip ppi} \]

For BASICS.PRN: \[ \text{data(format=prn,org=columns)} / \text{rate m1 m2 ip ppi} \]

For BASICS.DAT \[ \text{data(format=free,org=columns)} / \text{rate m1 m2 ip ppi} \]

Remember, you can omit the list of series names for RATS, XLS, PRN, and WKS format files, which always include series names on the file.

The ORGANIZATION Option

For spreadsheet and text files, there are two obvious ways that a data set can be arranged:

- Series run down the page in a column, with each row representing one observation for all series.
- Series run across the page in rows, so that each row represents all observations for one series.

In RATS, we refer to the first arrangement as organization by columns (or as organization by observation, because each row represents a different observation). We refer to the second arrangement as organization by row (or as organization by variable, since each row represents a different variable).

Organized by Column/Observation:

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0</td>
<td>105.3</td>
<td>42.5</td>
</tr>
<tr>
<td>2</td>
<td>101.2</td>
<td>106.2</td>
<td>41.7</td>
</tr>
<tr>
<td>3</td>
<td>103.5</td>
<td>106.1</td>
<td>43.8</td>
</tr>
<tr>
<td>4</td>
<td>104.2</td>
<td>107.5</td>
<td>44.4</td>
</tr>
<tr>
<td>5</td>
<td>104.3</td>
<td>108.5</td>
<td>45.0</td>
</tr>
</tbody>
</table>

Organized by Row/Variable:

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>X5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0</td>
<td>101.2</td>
<td>103.5</td>
<td>104.2</td>
<td>104.3</td>
</tr>
<tr>
<td>2</td>
<td>105.3</td>
<td>106.2</td>
<td>106.1</td>
<td>107.5</td>
<td>108.5</td>
</tr>
<tr>
<td>3</td>
<td>42.5</td>
<td>41.7</td>
<td>43.8</td>
<td>44.4</td>
<td>45.0</td>
</tr>
</tbody>
</table>

Earlier versions of RATS used ORG=OBSERVATION and ORG=VARIABLE as the option choices. For clarity, we now use ORG=COLUMNS for the first case, but ORG=OBS is still accepted. Similarly, the second case is handled using ORG=ROWS or ORG=VARIABLE.

The One Advantage of Free-Format

Free-format is less rigid about the one observation–one row, or one variable–one row organization described above. Instead, it requires merely that each observation (or each variable for ORG=ROWS) begin on a new row. It doesn’t care how many physical rows are used to represent a “logical” row. FORMAT=FREE, for instance, will accept the following:
Chapter 1: Basics

Organized by Row (Free-format only)

100.0  101.2  103.5
104.2  104.3
105.3  106.2  106.1
107.5  108.5
42.5   41.7   43.8
44.4   45.0

If you compare this with the example above, you will see that each variable takes up two rows. If you have this type of arrangement in some other type of file, you must either convert the file to free-format (that is, get rid of all information other than the data), or rearrange the file so it only has one observation or series per row. See Chapter 2 for more information.

Be Careful!

Obviously, a mistake in setting the ORGANIZATION option will cause some serious problems. For example, suppose you tried to read the “Organized by Column” sample with ORG=ROW. Your series X1 would contain the values 100.0, 105.3, 42.5, 101.2, 106.2, and 41.7—clearly not what you intended.

Remember to check your data after you first read it in. See Sections 1.1.3 and 1.1.4 for ways to verify your data.

Mixing and Matching Frequencies

In applied time series work, it is not uncommon to have data from several different sources. And, more than occasionally, the data will be reported at different frequencies. GDP numbers, for instance, are usually collected quarterly, unemployment data are monthly, interest rates and stock and commodity prices can literally be collected at any interval you desire.

RATS allows you to integrate all these types of data into your work. Do you have data which are monthly, but you want it to be quarterly? Just set a quarterly CALENDAR, and the DATA instruction will automatically convert the data (to quarterly averages by default, although you can select from other compaction methods). Quarterly to monthly? DATA will replicate the quarterly values for each month within the quarter. RATS can even convert between weekly and monthly, daily and annual, etc.

If you need to use this capability, see Section 2.3.

Multiple Data Files

You do not have to load your data with a single instruction. You can use several DATA instructions with a single data file or you can use multiple OPEN DATA and DATA instructions to read from multiple data sources. See Chapter 2 for more.
1.4 More About Regressions and Estimations

What’s Available?

RATS has quite a few regression and estimation instructions:

- **LINREG** is a very general linear regression routine which can do ordinary and weighted least squares, instrumental variables and two-stage least squares.
- **STWISE** does stepwise (linear) regressions.
- **AR1** estimates a linear regression with correction for first order autocorrelated errors.
- **NLLS** performs non-linear least squares.
- **SUR** estimates systems of (linear) equations using joint GLS (Seemingly Unrelated Regressions) or three-stage least squares.
- **GARCH** estimates univariate and multivariate ARCH and GARCH models.
- **NLSYSTEM** estimates systems of non-linear equations.
- **LDV** handles limited dependent variable models (censored and truncated data).
- **DDV** estimates discrete dependent variable models (logit, probit, Poisson regression).
- **PREGRESS** performs panel data estimation.
- **RREG** estimates using robust methods.
- **RLS** estimates using recursive least squares.
- **SWEEP** performs a regression of a set of “targets” on a set of “instruments”.
- **DLM** estimates state-space models.
- **MAXIMIZE** performs maximum-likelihood estimation, and can be applied to a huge variety of models.
- **CVMODEL** estimates covariance matrix models for structural VAR’s.

We introduced **LINREG** in Section 1.1.7. Below, we’ll discuss some basic elements of standard regression instructions and look at the **AR1** instruction—one of several estimation instructions that operate similarly to **LINREG**. In Section 1.5, we’ll look briefly at more general non-linear estimations. See Chapter 7 and the *Reference Manual* for more information on these.

Regression Format

“Regression format” is the manner which RATS uses to list regressors. It consists of a list of series and *lag fields*. You create a lag field by appending to the series name the list of desired lags in braces (*{}*). You can use the notation *(\text{n1 TO n2})* to represent all lags between \text{n1} and \text{n2}. Leads are represented as negative lags.

- \text{M1\{0 TO 4 8\}}\text{ M1 lags 0,1,2,3,4, and 8}
- \text{RATES\{1\}}\text{ RATES lagged once}
- \text{SEASON\{-1\}}\text{ One period lead of SEASON}
Supplementary Cards

The instructions **LINREG**, **AR1**, **STWISE**, **LDV**, and several others all have a form similar to:

```
linreg depvar start end resids
#< supp. card >  (explanatory variables in regression format)
```

that is, the dependent variable is listed on the instruction line, but the regressors are listed on a supplementary card. For example:

```
linreg ppi
# constant ip m2
```

regresses PPI on CONSTANT, IP, and M2.

Note that the intercept (CONSTANT) is *never* included in a regression automatically. You must list *all* the explanatory variables.

Ordinary Least Squares

**LINREG** is a very general instruction, used for many different forms of linear regressions. Without any bells and whistles, however, it just does ordinary least squares, as shown in the examples in Section 1.1.7. Various options allow you to do instrumental variables, weighted least squares, heteroscedasticity correction, and more. See Chapter 5 and the description of **LINREG** in the Reference Manual for details.

Regressions with Autocorrelation Correction

The instruction **AR1** estimates a regression including a correction for first-order serially correlated errors. Options allow you to select from Cochrane–Orcutt, Hildreth–Lu, and three other estimation procedures. The default is Hildreth–Lu. In Example 6.6 of Pindyck and Rubinfeld (1998 edition, example 6.5 in 1991 edition is similar), the authors estimate a Cochrane–Orcutt AR1 model for the sample data set used earlier in this chapter. As shown in the **BASICS.PRG** example file, you can estimate the model using the command:

```
ar1(method=corc) rate 1960:1 1995:8
# constant ip grm2 grppi{1}
```

Note that the syntax of **AR1** is virtually identical to that of **LINREG**. The formatting of the output is also very similar, except for the addition of the coefficient and test statistics for \( \rho \) (the serial correlation correction term).

Fitted Values

As we mentioned in Section 1.1.9, you can use the **PRJ** instruction to get fitted values from the most recent **LINREG** regression. This also works for many other regressions. However, some non-linear and multi-equation estimation instructions, such as **SUR**,
and \texttt{NLLS}, won’t allow the use of \texttt{PRJ}. For those, you can get the fitted values by subtracting the residuals from the dependent variable using the \texttt{SET} instruction. For example:

\begin{verbatim}
  nlls(frml=nllsmodel) rate / nllsresid
  set fitted = rate - nllsresid
\end{verbatim}

\textbf{Covariance/Correlation Matrix}

The estimation instructions all have an option \texttt{VCV} which requests display of the covariance/correlation matrix of the coefficients; this is placed below the standard regression output. This is a square table with covariances on and below the diagonal and correlations above the diagonal. For example:

\begin{verbatim}
  linreg(vcv) rate 1960:1 1995:8
  # constant ip grm2 grppi{1}
\end{verbatim}

produces (following the standard regression output)

\begin{verbatim}
Covariance\Correlation Matrix of Coefficients
\begin{tabular}{lcccc}
  & Constant & IP & GRM2 & GRPPI{1} \\
  Constant & 0.32884603 & -0.9071376700 & -0.6222523249 & -0.0656161953 \\
  IP & -0.00410542 & 0.00006228 & 0.3181626937 & -0.0532739768 \\
  GRM2 & -12.83113427 & 0.09029009 & 1293.01762850 & 0.0249884540 \\
  GRPPI{1} & -0.65535740 & -0.00732276 & 15.64993276 & 303.34841448 \\
\end{tabular}
\end{verbatim}

Here, the variance on the \texttt{IP} coefficient (shown on the diagonal) is nearly zero (0.00006228). The covariance between the \texttt{IP} and \texttt{GRM2} coefficients is around 0.09 (below diagonal), while the correlation for these two is approximately 0.318 (above diagonal). Note, by the way, that the header says Covariance\Correlation. We use the \ backslash to help you remember the structure of the matrix: covariance below and correlation above.
1.5  Non-Linear Estimation

Estimating linear models via ordinary least squares and related techniques is a fairly simple process, involving straightforward matrix computations. As you have seen, such models are easily handled in RATS. Non-linear estimation, however, is an inherently more general and complex task—the variety of different models that can be estimated is virtually limitless, and fitting these models requires the application of complex iterative optimization algorithms, rather than simple computations.

The estimation process often demands a fair bit of expertise and judgment on the part of the user, because the algorithms can be sensitive to initial conditions and are prone to converging to local, rather than global, optima. Thus, it should be no surprise that fitting non-linear models in RATS requires more effort and attention than fitting simple OLS models.

Nonetheless, the mechanics of estimating non-linear models in RATS are actually relatively simple. In fact, some non-linear models (Cochrane–Orcutt and related first-order autoregressive models, as well as logit and probit models) are handled with single instructions (\texttt{AR1} and \texttt{DDV}, respectively). Even the more general techniques, including non-linear least squares and maximum likelihood estimation, require just three basic steps:

1. Use the \texttt{NONLIN} instruction to define the set of parameters to be estimated.
2. Use one or more \texttt{FRML} instructions to define the formula (or formulas) to be estimated.
3. Estimate the model using the appropriate command (\texttt{NLLS} for non-linear least squares, \texttt{NLSYSTEM} for non-linear systems estimation, \texttt{MAXIMIZE} for maximum-likelihood estimation, for example).

Arriving at a reliable solution may require several steps, using different initial conditions or different choices of algorithm, and a careful evaluation of the quality of resulting fits. If you are interested in doing non-linear estimation, please be sure to read Chapter 7. For now, we’ll look at a simple example.

Suppose you want to estimate equations 1 and 2 from Section 1.1 using non-linear least squares, rather than OLS. It’s very easy, and here’s the necessary code (see \texttt{BASICS.PRG} for the complete example):

```plaintext
nonlin b0 b1 b2 b3
frml f1 rate = b0 + b1*ip + b2*m1diff + b3*ppisum
nlls(frml=f1) rate 1960:2 1980:12

frml f2 rate = b0 + b1*ip + b2*grm2 + b3*grppi(1)
nlls(frml=f2) rate 1960:1 1995:8
```

The \texttt{NONLIN} instruction defines the four free parameters to be estimated (the “coefficients” for this model). We use the same parameter list for both models.
Chapter 1: Basics

The FRML instruction defines the “formula” that you want to estimate. Formulas describe a calculation which depends upon the time subscript \( T \), basically coding up something like the expression on a SET. The right-hand-side expression is defined as a function of the current entry number, and can consist of any combination of real or integer constants, scalar variables, series, lag notation, the “T” subscript, functions, and even other formulas.

The NLLS command estimates the specified formula. In this example, the first NLLS command will generate a warning that it is using zero as the initial value for the four parameters, because we have not provided our own initial values. For a simple problem like this, almost any starting values will work fine. For more complex situations, providing an appropriate set of initial values becomes something of an art, and an important one at that (see Chapter 7).
1.6 More About Forecasting

What’s Available?

RATS can forecast using a wide range of modelling techniques:

- Simple regression models (Sections 1.1.9 and 8.2)
- Simultaneous equations (Chapter 11)
- Vector autoregressions (VAR’s) (Chapter 10)
- Exponential smoothing (Sections 9.2 and 9.3)
- Box-Jenkins (ARIMA) models (Sections 9.2 and 9.4)
- Spectral methods (Sections 9.2 and 9.5)

Chapter 8 provides a general overview of forecasting, while specific kinds of forecasting models are addressed in the sections noted above.

Equations and Formulas

Section 1.1.9 introduced the idea of defining equations and using \texttt{UFORECAST} or \texttt{FORECAST} to compute forecasts using equations. \texttt{FORECAST} can also produce forecasts for a formula, or a set of formulas (introduced in the previous section). Note the distinction between equations and formulas:

\textbf{Equations} are descriptions of \textit{linear} relationships. You can define them directly using the instruction \texttt{EQUATION}, but usually you create them using the \texttt{DEFINE} options of estimation instructions, as shown in Section 1.1.9. With \texttt{UFORECAST}, you use an \texttt{EQUATION} option to supply the equation you want to forecast. With \texttt{FORECAST}, you either list the equations you want to forecast on supplementary cards (one card per equation), or you use the \texttt{MODEL} option to specify a \texttt{MODEL} (group of equations and/or formulas) that you want to forecast. \texttt{MODELS} are usually constructed using the \texttt{GROUP} instruction.

\textbf{Formulas} (also called \texttt{FRML}s) are descriptions of (possibly) \textit{non-linear} relationships. These store a “\texttt{SET}” style function together with the dependent variable, and are normally defined using the \texttt{FRML} command. Before you can forecast a formula or set of formulas, you \textit{must} group them into a model using the \texttt{GROUP} instruction. You then use the \texttt{MODEL} option on \texttt{FORECAST} to forecast the model. See Chapter 8.

Simulations

You can computed random simulations of a model using \texttt{UFORECAST}, or with the \texttt{SIMULATE} command, whose syntax is virtually identical to that of \texttt{FORECAST}. You can also use other tools to supply your own shocks to a system. See Chapter 13 for details.
Chapter 1: Basics

1.7 More on Data Transformations

Introduction

SET (introduced in Section 1.1.5) is the general data transformation instruction. RATS also has several special-purpose instructions for doing transformations, such as SEASONAL for creating seasonal dummies and DIFFERENCE and FILTER for difference and quasi-difference transformations. However, most of the time you will be using SET.

Below, we describe SET in more detail, and present a kit of standard data transformations. Most of the examples will not based upon the data set we have been using, so you won’t necessarily be able to execute them as part of the tutorial. It should be easy to adapt them to your own needs, however.

We also describe the various Wizards available for doing many of these types of operations.

The Instruction SET

The general form of SET is

\[
\text{set}(\text{options}) \quad \text{series} \quad \text{start} \quad \text{end} \quad = \quad \text{function}(T)
\]

In the function part of the SET instruction, you can use constants, scalar variables, other series, matrices, and any of the arithmetic and logical operators available in RATS. In its most basic form, SET defines one series as a simple transformation of another series. For example:

\[
\text{set loggdp} = \log(gdp)
\]

sets each entry of the series LOGGDP to the log of the corresponding entry of GDP.

The start and end parameters are optional, and you can usually skip them when you are doing your initial transformations after DATA.

To set a series as a function of other variables, you use standard arithmetic notation:

\[
\begin{align*}
\text{set totalcon} &= \text{durables} + \text{nondur} + \text{services} \\
\text{set scaled} &= \text{resids/sigmasq}
\end{align*}
\]

Trend Series

You use the variable \( T \) to create trend series and dummy variables based upon “time”. \( T \) is equal to the number of the entry being set, where the entry you specified on CALENDAR is given number 1. For example:

\[
\begin{align*}
\text{set trend} &= t \\
\text{set trendsq} &= t^{*2} \\
\text{set exptrend} &= \exp(.05*t)
\end{align*}
\]

creates linear, quadratic and exponential (5% growth) trend series, respectively.
Chapter 1: Basics

Seasonal Dummies

RATS provides a special instruction `SEASONAL` for creating seasonal dummies. You use it in one of two forms:

```
seasonal seasons
seasonal(period=1948:2) february 1948:1 1997:12
```

The first creates `SEASONS` as a dummy for the last period of the year (4th quarter or December). By using `SEASONS` and its leads in a regression, you can get a full set of dummies without having to define a separate one for each period. With monthly data, the “lag field” `SEASONS{ -10 TO 0 }` covers dummies for February (lag “–10”, which is a 10-period lead) to December (lag 0).

The second creates `FEBRUARY` as a February dummy defined from 1948 to 1997. The fifth parameter (indicating that 1’s start in 1948:2) makes it a dummy for February.

Other Dummies

Dummy variables are easy to construct using logical and relational operators (which we discuss in Section 1.8), since these operators return the values zero or one.

```
set dummy = t>=1972:3.and.t<=1974:3
set large = pop>5000
set female = .not.male
```

The first example creates a dummy series called `DUMMY` using a logical expression. It stores the value 1 in entries where the logical expression is true, and a 0 in the other entries. In this case, the expression is true between 1972:3 and 1974:3 inclusive, so `DUMMY` is 1 for those entries and 0 elsewhere. The second sets `LARGE` to 1 when the corresponding entries of `POP` are greater than 5000, and 0 elsewhere. In the third example, entries of `FEMALE` are 1 when `MALE` is 0, and are 0 elsewhere.

Trend/Seasonals/Dummies Wizard

You can create many of the trend, seasonal, and dummy variables described above using `Trend/Seasonals/Dummies` on the Data menu.

Lag and Lead Transformations

Transformations involving lags or leads of a series can be written using the `T` subscript, lag notation, or a combination of both.

```
set pavg = ( price(t) + price(t-1) ) / 2.0
set pavg = ( price + price{1} ) / 2.0
set inflation = 400.*log( deflator/deflator{1} )
```

The first two are identical. They create `PAVGE` as the average of the current and first lag values of `PRICE`. The first uses `T` explicitly, the second uses lag notation. Which style you use is a matter of taste. The third example computes annualized growth rates of `DEFLATOR` (in percent) using the log difference approximation.
All of these transformations involve a lagged value, so the first entry will be set to missing: if \texttt{PRICE} is defined over 1922:1 to 1941:1, \texttt{PAVGE} will be defined only over 1923:1 to 1941:1.

**Differencing**

Simple differencing is, of course, easy to handle with \texttt{SET}:

\begin{verbatim}
set diffgd = gd - gd{1}
\end{verbatim}

RATS also offers the instruction \texttt{DIFFERENCE} for regular, seasonal, or fractional differencing:

\begin{verbatim}
difference gd / diffgd
difference(sdiffs=1,diffs=1) gd / ddsgd
\end{verbatim}

As noted earlier, you can also use \textit{Transformations or Differencing} on the \textit{Data} menu to create differenced series.

**Growth Rates**

There are several ways to compute growth rates or approximations to them. The first two \texttt{SET} instructions below compute (for quarterly data) annualized growth rates, the third and fourth compute period over period rates and the last computes the year over year growth.

\begin{verbatim}
set growx = 4.0*log(x/x{1})
set growx = ((x/x{1}) ** 4 - 1.0)
set growx = log(x/x{1})
set growx = x/x{1} - 1.0
set growx = x/x{4} - 1.0
\end{verbatim}

The \textit{Transformations Wizard} on the \textit{Data} menu can create simple growth rate series.

**Benchmarking and Normalizing**

These generally require one of the two options for \texttt{SET}; either \texttt{FIRST} or \texttt{SCRATCH}:

\begin{verbatim}
set(first=100.) capital 1920:1 1941:1 = .90*capital{1}+invest
\end{verbatim}

computes \texttt{CAPITAL} from an investment series beginning with a value of 100.0 for the first period (1920).

\begin{verbatim}
set(scratch) gdp 1950:1 1998:4 = 1000.0*gdp/gdp(1975:3)
\end{verbatim}

renormalizes \texttt{GDP} to take the value 1000 in 1975:3. You need the \texttt{SCRATCH} option because this uses two entries of \texttt{GDP} (current entry and 1975:3) in the function. See \texttt{SET} in the \textit{Reference Manual} for more on these options.
Chapter 1: Basics

The %IF Function—Conditional Expressions

The logical function \%IF(x, y, z) returns the value y if x is non-zero, and returns z if x is zero. This is a very useful function. For example:

```plaintext
set w = %if(test<0, %na, w)
set testseries = %if(t<=1990:12, seriesone, seriestwo)
```

The first makes all entries of W for which the series TEST is negative equal to missing (%NA is how you represent the missing value in expressions). The other entries are not affected. The second stores the values of SERIESONE in TESTSERIES for all entries through 1990:12. Entries from 1991:1 on are taken from SERIESTWO.

Note that the %IF function only evaluates a single–valued expression each time it is called. It is the SET instruction itself that “loops” over the entries in the sample range, calling %IF once for each entry in that range.

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.

```plaintext
set stockprice = %if(%valid(stockprice),stockprice,-999.0)
```

This replaces all the missing values of STOCKPRICE with the value -999.00. You might use this in preparation for exporting this series to a file format (such as ASCII) which doesn't support special codes for missing values.
1.8 Arithmetic Expressions

Introduction
The arithmetic expressions in RATS form the basis of the general transformation instruction SET, as well as calculations involving scalar and array variables, which you do primarily with the instruction COMPUTE (see Section 1.1.10 and Chapter 4).

RATS expressions are similar to those employed in many programming languages and applications. We give enough detail here to allow you to do almost any standard data transformation. However, we only discuss a few of the RATS functions—See Section 2 in the Reference Manual for a full list of available functions, and Appendix A for a quick guide to arithmetic operators.

Constants
You can represent a numeric value using a variety of forms:

- with or without decimal points: 5 13.6 -.393
- in scientific notation with a suffix of one of the forms: E+n, E+n, E-n, where n is a non-negative whole number: 2.3E5 (230000) -.4E-4 (-.00004)

In addition, RATS provides the following two constants:

%PI The constant π
%NA The missing value code

Arithmetic Operators
RATS supports the following arithmetic operators:

+ addition
– subtraction or negation
* multiplication
/ division
** or ^ exponentiation
+= Increment and assign (a += b is equivalent to a = a+b)
-= Decrement and assign (a -= b is equivalent to a = a–b)
*= Multiply and assign (a *= b is equivalent to a = a*b)
/= Divide and assign (a /= b is equivalent to a = a/b)

You can use parentheses () to control the order of execution of the operations, and you can nest sets of parentheses if necessary. You cannot use brackets [ ] or braces { } as substitutes, because these have other meanings in RATS. In the absence of parentheses, operations are done in the following order:
1. Negation (− used to change sign)
2. Exponentiation (exception: −a**b does a**b first)
3. Multiply and divide
4. Add and subtract
5. Logical operators (see below)

If two operations have the same precedence, they are done from left to right, so A−B+C is equivalent to (A−B)+C. The one exception to this is **: A**B**C is the same as A**(B**C).

All of this is based upon natural order of operations in algebraic formulas. Just as a+b/c is interpreted as a+(b/c) in algebra, A+B/C is A+(B/C) in RATS.

More on Functions

RATS provides many useful functions. As indicated in Section 1.1.10, functions do not serve as stand-alone commands by themselves. They are used in an instruction, such as on the right hand side of a SET or COMPUTE instruction, or as part of an instruction parameter. Functions accept one or more arguments and return a value or set of values which are then used by the instruction that called the function. Function arguments must be enclosed within parentheses () and there should not be a space between the function name and the “(“. Some of the more important ones:

- **LOG(x)** natural log (logₐ)
- **EXP(x)** e^x
- **SQRT(x)** square root
- **ABS(x)** absolute value
- **SIN(x)** sine (of x in radians)
- **COS(x)** cosine (of x in radians)
- **%IF(x, y, z)** is y if x is non-zero and z if x is zero.
- **%VALID(x)** is 0 if x is “missing” and 1 otherwise

See Section 2 in the Reference Manual for a complete list of available functions.

Examples

Here are some mathematical expressions and their RATS equivalents.

<table>
<thead>
<tr>
<th>Expression</th>
<th>RATS Code</th>
<th>Expression</th>
<th>RATS Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>b^2 − 4ac</td>
<td>b**2 − 4<em>a</em>c</td>
<td>1/(1 + y^2)</td>
<td>1/(1 + y**2)</td>
</tr>
<tr>
<td>2^−c</td>
<td>2**−c</td>
<td>a − bc^d</td>
<td>a − b*c**d</td>
</tr>
<tr>
<td>log(1+w^2)</td>
<td>log(1 + w**2)</td>
<td>e^−</td>
<td>u</td>
</tr>
</tbody>
</table>

Logical and Relational Operators

Logical operators have many uses in RATS, one of which is creation of dummy variables. They code true/false statements, returning a value of one for true and zero for false. These operators are shown below:

“∧” and “∨” can be any expression or number.
Chapter 1: Basics

Expression Translates to the Statement
A==B (or A.EQ.B) A is Equal to B
A<>B (or A.NE.B) A is Not Equal to B
A>B (or A.GT.B) A is Greater Than B
A>=B (or A.GE.B) A is Greater than or Equal to B
A<B (or A.LT.B) A is Less than B
A<=B (or A.LE.B) A is Less than or Equal to B
.NOT.A A is NOT non-zero (that is, A is false)
A.AND.B Both A and B are non-zero (true)
A.OR.B Either A or B or both are non-zero (true)

Some important points about these operators:

• The logical operators are lowest in order of precedence, so A+B==C*D is done as (A+B)==(C*D).

• The first six (the relational operators) take precedence over the last three, which are listed above in order of precedence:
  A.OR.B.AND.C.OR.D is done as A.OR.(B.AND.C).OR.D

• When testing for equality, be sure to use “==” (two equals), not “=” (just one)!!!

Examples

\[ Y > 3.0 = \begin{cases} 1 & \text{if } Y > 3 \\ 0 & \text{if } Y \leq 3 \end{cases} \quad T >= 10.\text{and}.T <= 20 = \begin{cases} 1 & \text{if } T \geq 10 \text{ and } T \leq 20 \\ 0 & \text{Otherwise} \end{cases} \]

Integer vs Real Numbers

RATS distinguishes between integer and real (floating point) numbers. Integers are whole numbers which serve primarily as entry numbers, loop indices and subscripts. For instance, the date 1970:1 is an integer, and subscripts such as \( T-1 \) are integer expressions. A constant typed without a decimal point is considered to be an integer:

\( 100.0 \) is real, \( 100, 1, 2 \) and \( 3 \) are integer.

When an operation mixes an integer with a real, the integer is converted to its equivalent as a real number before the operation is completed. For instance, if any one of \( A, B \) or \( C \) is real, \( (A+B+C)/3 \) is the same as \( (A+B+C)/3.0 \).

• Division of two integers results in the integer quotient with no fraction, so \( 10/3 \) is 3 and \( 1/2 \) is 0. This may or may not be what you intend, so you need to be careful in such situations.

• If you need to force RATS to convert an integer to a real in a situation in which that would not be done automatically, use the function \texttt{FLOAT}, for instance \texttt{FLOAT(T)}. Similarly, you can convert reals to integers with the function \texttt{FIX}. \texttt{FIX} truncates any remainder. If you want to round first and then convert to integer, do something like \texttt{FIX(ROUND(X,-1))} which rounds to the nearest 10 and converts to an integer.
Continuation lines

If you need to break up an expression and continue it on the next line, the best place to do so is before or after +, -, *, or /. Use a $ to indicate that the instruction is continued on the next line (see page 63). You may not split an expression at the following places:

- in the middle of a variable name or a number
- between a function or series name and a ( or { symbol following it
- in the middle of a logical operator such as .AND.

Missing Values

Operations involving a missing value produce a missing value as a result, except for equality and inequality (== and <>). Because missing values propagate automatically, you don’t have to worry about making special tests to insure that an expression only operates on valid numbers.

An invalid operation of any form will also produce the missing value code as a value. Examples are SQRT(-1.0) and 1/0. RATS does not distinguish between illegal operations (such as SQRT(-1.0)) and operations which could be assigned a “value” of infinity, such as 1./0.

Within an expression, you can represent a missing value only as %NA; not as NA or . or any of the alternatives that can be used on data files.

Referencing Series or Array Elements

You can access a particular entry of a series or an array by listing the entry or element in parentheses immediately after the variable name. For series, you would use an expression of the form “seriesname(entry)”, such as: FOODPROD(5) or DISPINC(1939:1).

An array reference would be of the form “arrayname(element)” for one-dimensional arrays (called VECTORS in rats), or “arrayname(row,column)” for two-dimensional arrays (RECTANGULAR or SYMMETRIC).

RATS also allows you to define “arrays of arrays”, such as a VECTOR of RECTANGULAR arrays. To reference a particular element of an object like this, just list the two sets of array element references consecutively. For example, for a VECTOR of RECTANGULAR arrays, the reference would be “arrayname(element)(row,column)”.


1.9 Conditional Statements and Loops

RATS provides several instructions that can be helpful in programming complex tasks or implementing repetitive analysis. These include the conditional instruction **IF**, the looping instructions **DO**, **DOFOR**, and **LOOP**, and conditional loops **WHILE** and **UNTIL**.

See Chapter 16 and the *Reference Manual* for more on these instructions, as well as other instructions that even allow you to create menu and dialog-driven interactive procedures.

Conditional Execution

The **IF** instruction allows you to control the flow of program execution based on the condition of a logical expression. For example:

```
if test==1
    display "Condition is true"
else
    display "Condition is false"
```

The “Condition is true” message is displayed only if the variable `TEST` contains the value 1.

Important note: The **IF** instruction only evaluates a single conditional expression—it does not include any sort of implicit looping over series or matrix entries. If you want to set values of an array or series based on a condition, you probably want to use the `%IF()` function, rather than using **IF** inside a loop. See page 51 for details.

Loops

RATS offers five different looping instructions (plus the `%DO` looping function, described in Section 2 and under **EWISE** in the *Reference Manual*):

**DO** is a standard loop instruction which loops over an integer index. It is like a FORTRAN DO or a BASIC FOR.

**DOFOR** is less standard but often very useful. It loops over a list of items: series, matrices, numbers, etc.

**WHILE** loops as long as an expression is true.

**UNTIL** loops until an expression is true. It always executes the enclosed instructions at least once; by contrast, **WHILE** can fail to loop a single time.

**LOOP** loops unconditionally. You need to use a **BREAK** instruction to get out of the loop.

**DO**, **DOFOR** and **LOOP** have a somewhat different setup from **WHILE** and **UNTIL**. They all loop over the instructions down to a matching **END**. Some examples:
do time=1990:1,1992:12
    linreg y time time+47
    # constant y(1)
    set c1 time time = %beta(1)
    set c2 time time = %beta(2)
end do
print / c1 c2

The code above does a “rolling regression”, using 48 data points for each regression, with the first regression starting in 1990:1, the second in 1990:2, and so on. This utilizes the fact that dates in RATS are handled as integer entry numbers, which means that you can use dates in integer computations and in DO loops. On each trip through the loop, the two SET instructions copy the estimated coefficient values from the reserved variable (set by LINREG) into entry TIME of the series C1 and C2. When the loop is complete, these series will contain all 48 pairs of coefficient estimates.

do for i = realgnp to realbfi
    set(scratch) i = log(i{0}/i{1})
end dofor

This does a percent change calculation for series from REALGNP to REALBFI. We use I{0} rather than just I so that I is treated as a series number, not an integer value.

By default, WHILE and UNTIL loop only over the next instruction or block of instructions. When you want the conditional loop to include more than one instruction, you must enclose them within braces { and }.

compute total=0.0, count=0
while total<100.0 {
    compute count=count+1
    compute total=total+series(count)
}
end

The END instruction is needed only if this is not within a larger compiled section. It puts RATS out of compile mode so it can execute the instructions. If the conditional instruction is inside a larger compiled section, omit the matching END:

if docount==1 {
    compute total=0.0, count=0
    while total<100.0 {
        compute count=count+1
        compute total=total+series(count)
    }
} end if

The IF command puts RATS in compile mode, and thus requires a matching END. The WHILE statement occurs inside a compiled section, so it does not require an END.
1.10 Using PROCEDURE and FUNCTION

What are PROCEDURE and FUNCTION?

Procedures and Functions are collections of RATS commands that can be executed with a single call to the Procedure or Function. Procedures are called just like built-in RATS instructions such as \texttt{LINREG} or \texttt{FORECAST}. In effect, they allow you to create your own "instructions". Functions are similar, but they are called in expressions, rather than as stand-alone commands. They allow you to create user-defined functions similar to functions already built into RATS, like \texttt{INV()} or \texttt{DET()}

These are the most advanced form of the RATS “compiler” structures (other compiled structures include \texttt{DO} loops and conditional blocks defined using \texttt{UNTIL} or \texttt{WHILE}—see the previous section). In fact, procedures and functions are so advanced that, once they have been written, other people can use them without having to understand any of the instructions within the procedure or function itself. In many cases, these allow you to implement cutting-edge econometrics tasks without having to program them yourself.

We have included a number of procedures with RATS—see the “RATS Procedures” PDF file included with RATS for the current list. We provide brief instructions on using procedures below. See Chapter 16 for more details on procedures, and Section 7.11 for details on user-defined functions. Also, you may want to look through the procedure files included with RATS (they have \texttt{*.*SRC} extensions) to see what tools are provided, or to get tips on writing your own procedures.

Also, you can find many other procedures and functions, written by us at Estima and by RATS users around the world, available for downloading free of charge from our website (www.estima.com).

Each procedure is usually stored on its own text file. Some of the procedures included with RATS are described in the manual. For the others (including those available on our web site), see the comment lines at the top of the procedure file.

Loading the Procedure

To use a procedure or function stored on a separate file, you need to have RATS execute the instructions that define the procedure. In most cases, RATS can do this automatically, by searching for a file with a \texttt{.SRC} extension whose name matches the name of the procedure (see the next page and Section 16.2.1 for details). Otherwise, you can compile procedures stored on an external using the \texttt{SOURCE} instruction:

\begin{verbatim}
source name of file with PROCEDURE or FUNCTION
\end{verbatim}

By default, \texttt{SOURCE} does not display the commands it is reading in. If you want to see the commands (which can be helpful if using a procedure for the first time, or debugging code you are writing), just use the \texttt{ECHO} option on \texttt{SOURCE}:

\begin{verbatim}
source(echo) bjident.src
\end{verbatim}
Executing a Procedure

A procedure has a name consisting of between 1 and 16 characters. You execute it using

@procedure name( options ) parameters
# < supplementary cards >

that is, everything is the same as a standard RATS instruction except two things:

- The procedure name is preceded by @.
- You can’t abbreviate the procedure name.

The following uses the procedure DFUNIT (Dickey–Fuller Unit root test) to do unit root tests on two different series. We don’t need the SOURCE instruction here, because RATS should be able to find the DFUNIT.SRC file automatically:

```r
cal(q) 1980
open data gdpdat.rat
data(format=rats) 1980:1 1999:4
@dfunit logusagnp
@dfunit loggbrgdp
```
1.11 RATS Instructions: Syntax

Introduction

It is time to define, formally, the syntax of RATS instructions. We have introduced most of the pieces informally along the way, but we need to define all the elements so you can use the descriptions in the Reference Manual.

Instruction Syntax

The general form of any RATS instruction is:

```
instruction (option field) parameters
# supplementary cards
```

The following instruction demonstrates all the parts that can make up an instruction:

```
graph (extend, key=upleft, header="Industrial Product and PPI") 2
  # ip
  # ppi
```

The Instruction Name

The “instruction name” (GRAPH in the example above) is the first object on a line. It does not have to begin in the first column.

Note: only the first three characters are significant, so GRAPH can be shortened to GRA. We use these abbreviations in many of the examples in this manual.

The Option Field

Immediately following the instruction name is the option field:

- It is enclosed in parentheses, with the left parenthesis typed in the first position after the instruction name. A very common error made by people starting with RATS is putting a space before the option field. Don’t!!
- Within the option field (that is, inside the parentheses), you separate options with commas.
- If an instruction has no options or you do not need to change any of its defaults, omit the option field entirely.

Options provide information about how you wish an instruction to be executed. In the example above, the EXTEND option asks GRAPH to extend dotted lines across the graph.

Most options have a default value which is used unless you indicate otherwise. GRAPH, for instance, has over forty separate options which allow you to make the graphs you create more attractive, informative, or readable, but the default values will still produce a reasonable picture.
Option Types

There are three types of options: switch options, choice options, and value options. The “option name” (which can be abbreviated to three or more letters) is used in different forms depending on the option type:

**Switch options**  A switch option is either on or off. It is set “On” if the option name is used by itself (EXTEND) and “Off” if it is prefixed by NO (NOEXTEND).

**Choice option**  With a choice option, you select from a collection of available settings. They take the form

\[ \text{Option=Choice Keyword} \]

as in KEY=UPLEFT; among the other alternatives are KEY=None and KEY=BETWEEN. Like option names, you can abbreviate Choice Keywords to three or more letters.

**Value option**  With value options, you supply the necessary information, such as a series name or an expression. This takes the form

\[ \text{Option=Constant, Variable Name or Expression} \]

HEADER="Retail and Farm Price" is an example.

The Instruction Parameters

For most instructions, one or more parameters follow the option field. These often indicate the series and entries to which the instruction is to be applied.

- Separate parameters from each other and from the option field or instruction name by one or more blank spaces.
- Usually, you can use expressions in the parameter field.

The sample GRAPH instruction on the previous page includes a single parameter—the number “2”. For GRAPH, this indicates that two series are to be plotted.

Most instructions have a fixed number of parameters, usually no more than five or six. However, some important ones, DATA in particular, accept a list of series names in the parameter field, so the instruction could have many parameters.

Supplementary Cards

Certain instructions, including GRAPH, require extra information in addition to that supplied on the instruction line itself. You enter this information on supplementary cards. These are usually very specific to each instruction: for instance, the regressors for instructions like LINREG are listed on supplementary cards.

In the sample GRAPH instruction on the previous page, the two supplementary cards indicate the series to be plotted (IP and PPI are the names of the series).
Chapter 1: Basics

You type a supplementary card with a # as the first non-blank character on the line. If you can’t fit all the information on one line, indicate a continuation by putting a $ at the end of the first line or lines (see page 63). Note that this is for readability—RATS will accept any length line:

```
linreg m1 1970:1 1988:6
# constant ppi{1 to 12} m1{1 to 12} $
ip{1 to 12}
```

Character Set

Except within strings ("..."), you can use upper or lower case letters interchangeably. Other characters used are the numbers and the following:

```
., # $ ( ) ; * / + - = " " _ & : @ > < { } [ ] % | ^
```

Symbolic Names

Symbolic names are the names of series and other variables. RATS imposes the following restrictions on a name:

- it must begin with a letter or the % symbol
- it must be a combination of letters, numbers and the characters %, _ and $
- it may be no more than 16 characters long (or, more accurately, only the first sixteen characters matter)

Examples are M1, GNP82$, IPD_TOTAL, STOCKPRICES. The case of the letters does not matter. On output, they will always be upper case.

Names must be unique, so you must avoid the names that RATS has reserved for functions and values. Most of the reserved names begin with the % symbol to help differentiate them from user-defined variables, but a few very common function names (SIN and COS, for instance) do not. The most common conflicts are with I, J and T, which are reserved integer variables. See Appendix B in the Reference Manual for a complete list of the reserved names.

Numbers

You can type data values and numeric constants in several forms:

- with or without decimal points: 5 13.6 .393
- in scientific notation with a suffix of one of the forms: E+n, E+n, E-n where n is a non-negative whole number; examples 2.3E5 (=230000) -.4E-4 (=–.00004)

Literal Matrix Notation

You can use the || and | characters to supply a matrix as a “literal” expression, rather than using a predefined variable (see Section 4.6 for details). For example:

```
boxjenk(ar=||1,4,12||) y
```
Comment Lines and Comment Blocks

A line with * as the first non-blank character is treated as a comment and is ignored. You can also mark an entire block of lines as a comment by using the symbol /* to mark the beginning of the block and */ to mark the end of the block. (Note: /* and */ must be the first (non-blank) characters on the lines). While this provides a convenient way to enter several lines of comments, it is probably most useful commenting out an existing block of instructions. For example:

```
* This is a comment line

/*
   This is a comment block
   linreg(inst) y
   # constant x1 x2
*/

linreg y
# constant x1 x2
```

This skips the instrumental variables regression, but does do the regular linear regression.

We use comments rather extensively in the examples of this manual to help you understand the programs. However, note that we also often use marginal comments which are set off from the rest of the line by being in italic type rather than bold face. This is not legal in an actual RATS program.

Continuation Lines

RATS will accept instructions of any physical length. Sometimes, though, a line becomes so long that it is difficult for you to handle. To split a line into more manageable segments, put a $ character at the end of the unfinished line.

There are certain restrictions on the places at which a single expression may be broken up, as described in Section 1.8. In most instances, you put the $ at the end of the line, separated from the rest of the line by at least one blank. An example is

```
set gnp = consumption + investment + govtexpend + $ export - import
```

There is no limit on the number of continuation lines allowed for a single instruction.

Multiple Statements on a Line

You can put several instructions on a single typed line if you separate them with semi-colons (;). It is a good idea to use this only for a set of short instructions which you see almost as one, or for attaching short comments to instructions. Examples:

```
set gdp = log(gdp) ; set m1 = log(m1)
smpl 1960:1 1985:3          ;* Set Sample for Remaining Instructions
```
1.12 More on Entry Ranges

Background
Each instruction which acts upon data series has a pair of parameters indicating the range of entries which the instruction is to use. We call these start and end.

In many situations, every instruction in a program will use the same entry range. And only rarely will the range change with almost every instruction. Because of this RATS provides two methods which allow you to avoid retyping the range on every instruction:

1. Default ranges, in which RATS uses the longest stretch of entries it can given the series involved in the instruction.
2. The instruction SMPL, which sets the default range for subsequent instructions.

The / Symbol
The slash symbol (/) takes the place of both start and end. It tells RATS to use the default entry range for that instruction. For example:

```plaintext
data(format=rats) / rate m1
```

If start and end are the last two parameters that you are using on the instruction, you can simply omit them entirely to get the same behavior.

The “default range” will normally be the range specified by the CALENDAR and ALLOCATE instructions. When you use the / symbol in instructions such as LINREG or other estimation instructions, RATS automatically adjusts the range to accommodate any missing values in the range, as well as any leads or lags.

If you don’t supply an ALLOCATE instruction, the default length will be taken from the range used on your DATA instruction.

The * Symbol
You can also set one end of the range explicitly and let RATS determine the other end. Use * for the parameter you want RATS to determine.

```plaintext
linreg rate * 1990:1
# constant rate{1 to 3}
```

This runs the regression from the earliest period possible (given available data and the maximum lag length of 3), through the period 1990:1.
Using SMPL

The **SMPL** instruction lets you control the default range. It is an important instruction in TSP–like programs, but is less so in RATS because:

- You can set explicit entry ranges on individual instructions where necessary.
- You can use default ranges on most transformation and regression instructions.

**SMPL** is useful in situations where you want to run a sequence of regressions, forecasts, or other operations over a common fixed interval (other than the default range).

For instance, suppose you have data from 1922:1 through 1941:1, but you want to run two regressions over the period 1923:1 to 1935:1.

```
smpl 1923:1 1935:1
linreg foodcons
# constant dispinc trend
linreg foodprod
# constant avgprices
```

Once you set a **SMPL**, RATS uses the **SMPL** range as the default. To clear a **SMPL**, just issue a **SMPL** instruction with no parameters. We recommend that you do any preliminary transformations before you set a **SMPL**.

If you need to skip entries in the middle of a data set, use the **SMPL** option (see Section 5.2).
1.13 Error Messages

Overview

In a perfect world, we wouldn’t need to talk about errors or error messages. Unfortunately, you will undoubtedly encounter an occasional error message in your work with RATS. In this section, we discuss how to interpret error messages and correct errors.

RATS Errors and Error Messages

Whenever RATS detects an error, it displays an error message describing the problem. Where possible, these messages provide you with specific information about the error.

Each error message begins with an error code: a one-, two-, or three-letter code indicating the general type of error, plus a numeric code identifying the specific error. For example, general syntax errors begin with the code “SX”, syntax errors involving options have the code “OP” (for OPtion), and errors involving linear regressions begin with the code “REG”.

See Appendix C in the Reference Manual for a complete listing of all the RATS error messages, along with some suggested causes and remedies.

Syntax Errors

Syntax errors probably appear most often. These occur while RATS is reading a line (as opposed to errors which occur as RATS is trying to execute a line it has read successfully). Syntax errors generally mean that RATS couldn’t make sense of the line it was processing. Missing parentheses, misspelled instruction or option names, illegal characters, and incorrectly entered numbers are some common causes of syntax errors.

RATS displays syntax errors somewhat differently when executing instructions from a file than it does in interactive mode.

In interactive mode, RATS displays two lines of information: the error message, and then the portion of the line which caused the error, enclosed in >>>> and <<<<< symbols. RATS only prints the line up to the point at which the error occurred. It will display a maximum of 20 characters, counting back from the point of the error.

For example, suppose you tried the following in interactive mode:

```plaintext
set trendsq 1 50=t**2
```

This should have at least one blank space between the “end” parameter (50) and the =. Because that space is missing, RATS will produce the following error message:

```plaintext
Error SX15. Trying to Store Into Constant or Expression
>>>set trendsq 1 50=t<<<<
```
As you can see, RATS stopped processing the line immediately after it detected the illegal assignment operation.

If this occurs while processing a file (either in batch mode or when using `SOURCE`), the output would look like this:

```
SET TRENDSQ 1 50=T<<<<
Error SX15. Trying to Store Into Constant or Expression >>>>**2
```

It marks the point of the error with `<<<<` on the line before the error message. On the third line, it displays `>>>>` followed by the remainder of the line (if any).

RATS often needs to look one character ahead to identify particular operators. For instance, if it sees an `=`, it must check the next character to see if the operator is `==`. Thus, the true culprit may be one character before the one shown.

If the reason for a syntax error isn’t immediately obvious, **check the entire line**, not just the point at which RATS stopped reading the line. RATS can often interpret the actual mistake in a way that allows it to go on reading a few more characters from the line.

For instance,

```
data (unit=input) / gnpdata
```

produces the error

```
Error SX11. Identifier INPUT Is Not Recognizable >>>>data (unit=input)<<<<
```

RATS reads all the way to the right parenthesis before it generates the error, but the problem is actually the (illegal) blank space between `DATA` and the left parenthesis. This blank causes RATS to interpret `(UNIT=INPUT)` as the `start` parameter, rather than as the option field. It is legal to use an expression like this for `start`. However, `INPUT` is not a recognizable variable name, so RATS produces the error message.

**Other Types of Errors**

Once RATS has successfully interpreted the line, it does not echo back the characters on the line. All you will see is the error message.

**Termination Due to Errors**

When executing a program in batch mode, RATS will stop execution immediately if it encounters an error. In interactive mode, you can usually continue working from the point of the error once you’ve solved the problem. For example, if you forget to dimension an array before using it in an `EWISE` instruction, you can simply `DIMENSION` the array, and then re-execute your `EWISE` instruction.
Chapter 2
Dealing with Data

Almost every time that you use RATS, you will need to bring data into the program. Since collecting and organizing your data set may be the single most time-consuming part of your analysis, you would obviously like RATS to accept your data with as little fuss as possible. We have provided you with a wide range of options for handling this important step.

You should note that RATS is not a “data set-oriented” program. Simply selecting a data file does not make the contents immediately available for analysis: you must use the DATA instruction to bring the data into working memory. This may seem to be an unnecessary extra step, but it offers numerous advantages. For example, you have the option of reading in only certain series and certain entries from the data set. Converting data from one frequency to another can be handled automatically as you read in the data.

RATS supports reading and writing data from a wide range of sources and in many different formats. The most popular are RATS format and Excel spreadsheets.
Chapter 2: Data

2.1 The Tools

Below, we discuss the instructions you can use to read data from a file into a RATS session and to write data from RATS out to a file. If you are new to RATS, please see the tutorial in Chapter 1 for additional information.

Getting Data In: The DATA Instruction and Data Wizard

Most of your projects in RATS will require reading data in from a file or files. The critical instruction for this is \texttt{DATA}, which reads data from a file into one or more series variables stored in memory. You use the \texttt{FORMAT} option on \texttt{DATA} to tell RATS what type of file you will be reading. Depending on the type of file, you may also need to use the \texttt{ORGANIZATION} option to specify whether the data series on the file are arranged in rows or columns. Note that the default format option for reading data is simple free-format text. RATS does not attempt to determine the format of the data itself—you need to set the \texttt{FORMAT} option yourself.

To read data from an external file, you must first open that file. You can do this with the instruction \texttt{OPEN DATA}. If you forget to open the file first, RATS will ask you to select a file when you execute a \texttt{DATA} instruction.

The \textit{Data Wizard} can help you with the instructions needed to read a data file. See Section 1.1.2 for details.

Getting Data Out: The COPY Instruction

The companion instruction to \texttt{DATA} is \texttt{COPY}, which writes data series to a file. You can produce data files in a wide range of formats, including spreadsheets, RATS format files, text files, and binary files.

As with \texttt{DATA}, you need to open a file before writing to it with \texttt{COPY}. If you forget, RATS will prompt you for a file name. You open the file with an \texttt{OPEN COPY} instruction. For RATS format files, \texttt{COPY} can only create a new file. To write data to an existing RATS file, use the instructions \texttt{DEDIT}, \texttt{STORE}, and \texttt{SAVE} (see Section 2.5).

Other Instructions for Reading and Writing Data

\texttt{DATA} and \texttt{COPY} can only read and write \texttt{SERIES} type variables.

For other types of variables (such as scalar reals, integers, arrays, labels, strings), you can use the \texttt{INPUT} or \texttt{READ} instructions (with the \texttt{UNIT=DATA} option) to read data into RATS, and the \texttt{WRITE} or \texttt{DISPLAY} instructions (with the \texttt{UNIT=COPY} option) to write data to a file.

The \texttt{REPORT} instruction (see Section 4.8) can also be useful for outputting regression results and other information. If you use the \texttt{WINDOW} option, \texttt{REPORT} will send output to spreadsheet-style window, which you can then export to a variety of file formats using \textit{File–Export}. Or you can use the \texttt{UNIT} and \texttt{FORMAT} options to write the \texttt{REPORT} output directly to a file.
Double-Check Your Data!

It is very important to be sure that data are being read in properly, particularly when reading from a data file for the first time. The best way to do this is to use the PRINT command to display the series on the output window so you can inspect the data visually. If you are working with very long series, at least check a limited range of entries. You can also use the TABLE or STATISTICS instructions to display basic statistical data on your series, or graph the series using the GRAPH instruction.

If you’re running rats interactively, you can create a table of the processed data, show the table of statistics and a number of basic graphs off the series window. After reading the data, do Show Series on the Wizards menu to get the list if it’s not already on the screen; then select the series that you want to examine and click on the toolbar icon that does the operation you want.

Moving Data Around

RATS allows you to write data in a wide range of forms using COPY. You can also open or create a RATS format file using New RATSData or Open RATSData on the File menu, and move data from one RATS format file to another or from the working data series to the RATS file. In fact, with RATS format used as an intermediate stop, you can easily translate data from, for instance, a poorly formatted text file to a well–organized spreadsheet.

The RATS file format is directly portable across different platforms (Mac, Windows, UNIX) with no translation required. You simply need to be able to transfer binary data. XLS, WKS, PRN, DIF and the RATS PORTABLE format are other machine-independent formats which RATS supports.
Chapter 2: Data

2.2 Where Are Your Data Now?

The suggestions below are designed to help you figure out the best approach to getting your data read into RATS so you can make use of them. Details on the various file formats supported by RATS follow later in this chapter.

On Paper

If you have quite a bit of data, you’re probably best off scanning it to create a text file, and continuing as described in that part of this section. Since character recognition software is still imperfect, you’ll have to be extra careful in checking that the data came in properly.

If you need to key in your data, these are the simplest ways to prepare it:

- If it is a small data set, you can type the data into a text file using any text editor (including RATS itself), and then read the data into your program using `DATA (FORMAT=FREE)` . You can also include the data directly in your program file by using a `DATA` instruction with the `UNIT=INPUT` option followed by the data values.
- You can use the data editor available through the New Series icon to type the data directly into a RATS format data file. (Open or create the file first using Open RATSData or New RATSData on the File menu).
- You can use a spreadsheet program or data base manager to type in the data. RATS supports Excel XLS, Lotus WKS and similar formats, and dBase DBF files. You can use any program which can save data in one of these formats. Read the data into RATS using `DATA` with the appropriate `FORMAT` option.

The RATS data editor (the second option listed above) is designed to handle time-series data sets easily, editing one series at a time. If your data set is more “case-oriented,” a spreadsheet or data base editor is likely to be more convenient, since you can edit across series. If you use a spreadsheet program, see the restrictions outlined in Section 2.6.

No matter which method you use, be sure to check the data after reading it into RATS.

In RATS Format

If your data are already stored in a RATS format file, you can read the data using the instruction `DATA (FORMAT=RATS)`.

The RATS file format is the same for all versions 4 and above and is portable across all platforms supported by RATS. You can also read files created with RATS Versions 2 and 3 running on the same type of computer you are currently using.
In a Spreadsheet

RATS can read data from a variety of Excel and Lotus spreadsheet formats, using the `DATA` options `FORMAT=XLS` (for Excel formats) or `FORMAT=WKS` (for Lotus WK1, WKS, WK3, etc.). Use the `ORGANIZATION` option to tell RATS whether the data are in rows or columns. For multi-page workbook files, you can use the `SHEET` option to tell RATS which worksheet you want to read.

Before you can read the data, you need to make sure the spreadsheet conforms to the restrictions of Section 2.6. In particular, make sure that the variable names are acceptable to RATS: they must start with a letter and consist of letters, digits, _ and $, and only the first sixteen characters are significant (see Section 2.6 for a way to convert series names). Also, the cells outside of the data table should be completely empty. It can sometimes help to erase several columns to the right and several rows below your actual data. RATS makes a pass through the file to try to determine the boundaries of the actual data table, but the less extra information on the file the better.

If you update your data set frequently, you will probably be better off leaving it in the spreadsheet. However, if the data are fairly stable, you should convert it to RATS format, to gain the speed and flexibility that it provides.

If the spreadsheet has a layout far removed from that described in Section 2.6, you may need to export it as a text file and use the techniques described below.

In a Database File

If the data are in a database which you can access using SQL commands, you may be able to read it using the Professional version of RATS using `FORMAT=ODBC`. See Section 2.7.

If your data is in a DBF format (dBase and compatible programs), you can read the data using the `DATA` command with the `FORMAT=DBF` option. Names for any fields you choose to read must conform to the RATS series name restrictions. RATS can make use of date fields if included on the file. You can specify the series you want to read, or leave the series list blank to read in all fields on the file.

In a Haver Analytics Database

RATS can read data directly from a DLX file from Haver Analytics. You use an `OPEN HAVER` instruction to let RATS know the database that you are accessing, then `DATA (FORMAT=HAVER)` to read a selected set of series. See Section 2.10 for details.
In a FAME Database File

FAME is a database management program provide by FAME Information Services, Inc. If you have the “Professional” version of WinRATS, or have the RATS–FAME add-on interface for UNIX RATS, you can read data from, and write data to, FAME format databases.

The process for reading and writing data is very similar to other file formats. On DATA or COPY commands, just use the option FORMAT=FAME (the ORG option is not required). Also, you must list the names of all the series you want to read or write.

See Section 2.11 for more on working with FAME files.

In the Global Insight Basic Economics Database

RATS can read data directly from the “Full” version of the Basic Economics database, available from Global Insight (formerly known as DRI Basic Economics, and before that, as Citibase). You need to use the Preferences operation on the File menu to tell RATS where the data are located on your network or hard disk. Then, in your program, use DATA with the option FORMAT=DRI (or FORMAT=CITIBASE) to read data directly from the database.

In a Text File

What you should do depends a lot on the format that you have inherited and how big the data set is. See Section 2.8 for details, but the following general guidelines apply:

• If you can read it free–format (numbers separated by blanks or commas, without any text labels on the file), use FORMAT=FREE and the appropriate ORGANIZATION option.
• If you can easily edit it into a free-format file (by eliminating strings), do that.
• If the file already contains series labels, or such labels can be added easily, you should be able to edit the file so that it can be read using FORMAT=PRN. You could also consider importing the file into a spreadsheet program and reformatting it. In either case, the required format is described in Section 2.6.
• Failing any of the above, if the file is at least in well-organized columns, it may be readable using a FORTRAN format (Section 2.9).

Regardless of the form, if the data file is fairly stable, we recommend that you convert it to RATS format or one of the spreadsheet formats.
Text Files—An Example

Consider the following. It has three years of quarterly data for two firms.

<table>
<thead>
<tr>
<th>YEAR</th>
<th>FIRM A</th>
<th></th>
<th></th>
<th></th>
<th>FIRM B</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>11.3</td>
<td>11.6</td>
<td>10.9</td>
<td>12.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td>13.0</td>
<td>12.8</td>
<td>11.5</td>
<td>12.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>12.9</td>
<td>13.0</td>
<td>13.2</td>
<td>13.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We cannot easily convert this to PRN because the data for a single series span multiple rows. We can follow the second strategy by eliminating the “FIRM A” and “FIRM B” lines and deleting the “2003”, “2004”, and “2005”. We would then read the resulting file with

```
cal(q) 2003
open data test.dat
data(format=free) 2003:1 2005:4 firma firmb
```

This would be our choice with a data set of this size. An alternative, which we would use if this file were larger, is to use a FORTRAN format. We discuss how that could be done in Section 2.9.

After reading the data, we would recommend writing the series to a RATS format file using the DEDIT, STORE, and SAVE instructions:

```
dedit(new) mydata.rat
store firma firmb
save
```

This will allow you to easily read the data in future sessions using a simple DATA (FORMAT=RATS) command. You will also be able to view or edit the data, or convert them to other frequencies. If, instead, you want to convert to an Excel spreadsheet, do something like

```
open copy mydata.xls
copy(format=xls,org=cols,dates) / firma firmb
```
2.3 Changing Data Frequencies

To convert data to a different frequency, you simply need to set the **CALENDAR** to the desired frequency and read in the data using the **DATA** instruction. Given a mismatch between the frequency of the data on the file and the current **CALENDAR** seasonal, RATS will automatically compact or expand the data to match the **CALENDAR** setting.

You can take advantage of this for any of the file formats for which RATS can process dates: RATS format, spreadsheet files (including PRN), DBF, PORTABLE and any of the commercial database formats.

**Compacting Data**

To compact from a higher frequency to a lower frequency, just follow these steps:

1) Make sure the source data file contains valid date information at the original (higher) frequency. RATS can’t compact data unless it can determine the frequency of the source data from the dates on the file.

2) Set the **CALENDAR** to the target (lower) frequency. For example, if you are compacting monthly data to a quarterly frequency, specify a quarterly **CALENDAR**.

3) Read the data using the **DATA** command. RATS will automatically compact the data to match the **CALENDAR** frequency using the method specified by the **COMPACT** or **SELECT** option. The default is **COMPACT=AVERAGE**.

The **COMPACT** and **SELECT** options allow you to select from several compaction methods. Note that the two options are mutually exclusive. If you try to use both, RATS will honor the **SELECT** choice. The choices are:

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

- **AVERAGE** simple average of the subperiods
- **SUM** sum of the subperiods
- **GEOMETRIC** geometric average of the subperiods
- **FIRST/LAST** first or last entry of each subperiod, respectively
- **MAX/MIN** maximum value or minimum value from each subperiod

```
select=subperiod to select
```

compacts data by selecting a single subperiod from within each period.

Suppose you have a quarterly **CALENDAR** and you want to read a monthly data series. **DATA** with **SELECT=3** will read in the third month from each quarter (that is, the March, June, September, and December observations from each year). With the default option of **COMPACT=AVERAGE**, each quarterly value will be the average of the three months which make up the quarter.

It is more complicated when you move between dissimilar frequencies: weekly to monthly, for instance. If you use **SELECT=2**, it will select the second full week within the month. Suppose you have the weekly observations shown below. Since weekly
data are classified in RATS according to the end of the week, March 13, and not March 6, will end a full week for this data set. If you compact data using the other methods, RATS will give 6/7 of the March 6 value to March, and 1/7 to February. If the input data are:

<table>
<thead>
<tr>
<th>Week ending</th>
<th>March 6</th>
<th>March 13</th>
<th>March 20</th>
<th>March 27</th>
<th>April 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15.0</td>
<td>11.0</td>
<td>13.0</td>
<td>18.0</td>
<td>20.0</td>
</tr>
</tbody>
</table>

the value for March (with COMPACT=AVERAGE) is:

$\frac{6/7 \times 15.0 + 11.0 + 13.0 + 18.0 + 4/7 \times 20.0}{6/7 + 3 + 4/7} = 14.97$

**Expanding Data to Higher Frequencies**

If RATS detects that the data on the file being read are at a lower frequency than the current CALENDAR setting, it will automatically expand the data to the higher frequency by setting each subperiod equal to the value for the full period. For example, if you set a quarterly CALENDAR and read in annual data, RATS will set the value of each quarter in a given year to the annual value for that year.

For dissimilar frequencies, the value of a period which crosses two periods of the lower frequency data is a weighted average of the two values. For instance, in moving from monthly to weekly, the value for the week ending March 6 will be:

$\frac{1}{7} \times \text{February} + \frac{6}{7} \times \text{March}$

For a more complex interpolation, you would first read the data as described above to expand to the higher frequency, and then apply the desired interpolation routine to the expanded data. RATS ships with three interpolation procedures you can use—these are the DISTRIB, INTERPOL, and CHOWLIN procedures (supplied on the files DISTRIB.SRC, INTERPOL.SRC, and CHOWLIN.SRC respectively).

DISTRIB and INTERPOL use only the data in the series being interpolated, while CHOWLIN uses related series. DISTRIB and CHOWLIN maintain the sum of the data across each full period, while INTERPOL keeps fixed the final value within each full period.

The following example reads in quarterly GDP data at a monthly frequency. The DISTRIB procedure is employed to produce a monthly series. As noted in the comments in DISTRIB.SRC, we need to multiply the result by 3 (the ratio of the periodicities) to maintain the original levels.

```
calendar(m) 1947
open data haversample.rat
data(for=rats,verbose) 1947:1 2006:4 gdp
@distrib(factor=3) gdp gdpm
set gdpm = 3*gdpm
```
2.4 Missing Data

Coding

Internally, RATS represents missing data with a special value. On most machines this is the value “+infinity.” On output, a missing value is denoted as “NA” for Not Available. In a RATS expression, this value is available as %NA. Thus,

\[
\text{set fixx} = \%if(x<0.0,\text{na},x)
\]

will set \text{FIXX} to be missing for entries where \text{X}<0 and to \text{X} otherwise. You can test whether a value is missing using the \%VALID function: \%VALID (\text{X}) is 0.0 if \text{X} is missing and 1.0 otherwise. You can also use \text{X}==%NA.

RATS has special ways of handling missing values for each of the formats.

Spreadsheet Files

The “+infinity” coding is the same as is used by the major spreadsheet programs for N/A’s. Thus, if a cell contains the explicit missing value function (\text{NA}() in Excel), RATS will read a missing value at that cell. When you export data to the spreadsheet, RATS passes the code value (as a number, not a function), so it will display as \text{NA} or #N/A on the spreadsheet.

RATS will also interpret blank cells within the spreadsheet as missing values.

Spreadsheet-Style Text (PRN) Files

Missing values are a problem if you have to save the data in a space or tab separated text format (\text{PRN} or \text{TSD}). You can’t simply leave the cell blank in the worksheet, as you can with an actual spreadsheet, because a blank cell is indistinguishable from the blanks used to separate data items. Instead, you’ll have to use one of the codings described in the next paragraph.

Text Files

RATS will accept the following as codes for missing values in text-format data files:

- The characters “NA” (upper or lower case)
- A decimal point, followed by zero, one, or two non-numeric characters (for example, . or .\text{NA})

Note, however, that you cannot use these within a RATS expression: use \%NA for that.

RATS will interpret any block of non-numeric characters as a missing observation. However, if the characters don’t fit the description above, RATS will issue a warning message that it has encountered invalid input.

For text files being read using FORTRAN formatting, you can use the codes (such as \text{NA}) for missing values used by other text files. You can also leave a blank area where the missing values should be and use \text{BLANK=MISSING} on the \text{DATA} instruction. This will interpret the blank area as a missing value (as opposed to a zero).
Numeric Codes

If you have data which use a specific numeric value (–999 or something similar) to indicate a missing value, use the option `MISSING=missing_value_code` on `DATA`. *Whenever possible, use integers for missing value codes.* The realities of numerical precision in the computing world mean that RATS may not be able to match exactly a decimal-valued code such as –999.99.

Suppose your data set uses –9999 for missing values. You could use something like the following:

```
data(format=prn,org=columns,missing=-9999) / sales revenue
```

This would read the series `SALES` and `REVENUE`, and convert any values of –9999.0 to missing values.

If you have several missing value codes (such as –999 = no response, –998 = invalid response), you have two options:

- Edit the file and replace the two codes by a single code.
- Read the data and alter it within RATS.

This is an example of the latter:

```
data(format=free,org=col) / income charitable mortgage
set income = %if(income==-999.or.income==-998, %na, income)
```

The `%IF` function tests whether the current entry of `INCOME` is equal to either of our two missing value numeric codes. If so, the function returns the `%NA` missing value code, which is then stored in `INCOME`. Otherwise, the existing value of `INCOME` is retained.

To apply this procedure to multiple series, we can use a `DOFOR` loop:

```
dofor i = income charitable mortgage
   set i = %if(i{0}==-999.or.i{0}==-998, %na, i{0})
end dofor
```

Note that the index variable `I` will actually be an integer variable containing the series number associated with the current series. As a result, we need to use the lag notation `{0}` (specifying the zero lag) on the right hand side of the `SET` instruction. This tells RATS to treat `I` as a series, not as an integer number.
Skipped Dates

Suppose that you have a (daily) data set in which holidays and other non-trading days are simply omitted. If you have no date information on the file, there is little you can do: you will have to treat it as an irregular time-series (omit the \texttt{CALENDAR} entirely, or use \texttt{CAL(IRREGULAR)}), and just supply the total number of observations for the \texttt{length} parameter on \texttt{ALLOCATE}.

If you \textit{do} have dates on the file, \texttt{DATA} (and \texttt{STORE} with \texttt{CONVERT}) will code entries corresponding to the skipped dates as missing values. For example, consider the following portion of an \texttt{XLS} file, which skips the (U.S.) Thanksgiving holiday on November 23, 2000:

\begin{verbatim}
SALES_DATA
2000:11:20 3590.50
2000:11:21 4256.05
2000:11:22 2987.23
2000:11:24 6799.87
\end{verbatim}

You might read the data with the following instructions:

\begin{verbatim}
calendar(d) 2000:1:2
open data sales.xls
data(format=xls,org=col) 2000:1:2 2000:12:31 sales_data
\end{verbatim}

The data in the \texttt{SALES_DATA} series for the week of Thanksgiving would be:

\begin{verbatim}
2000:11:20 3590.50
2000:11:21 4256.05
2000:11:22 2987.23
2000:11:23 NA
2000:11:24 6799.87
\end{verbatim}

If you really \textit{want} to omit the holidays without having gaps in your data (in the sense that you want to treat the data for November 22 and November 24 as adjoining entries), you cannot treat the data set as “Daily” because RATS insists that daily data be five days a week. If you use \texttt{CALENDAR(IRREGULAR)}, RATS will ignore the dates on the data file and read the data from the file into consecutive entries. \textit{There is little difference between the two ways of handling the holidays, unless you do some type of time-series analysis that uses lags or leads.}

Another option for these situations is to go ahead and read the data using a \texttt{DAILY} or \texttt{SEVENDAY CALENDAR}, as in the Thanksgiving example above, then use the \texttt{SAMPLE} instruction to create “compressed” versions of the data without the missing-values:

\begin{verbatim}
sample(smpl=%valid(sales_data)) sales_data / sales_nomissing
\end{verbatim}

You can use the uncompressed data for printing or generating graphs (where you want to include date information), and use the compressed series for estimating Box–Jenkins, GARCH, and other similar models that don’t accept missing values.
2.5 RATS Format

What is It?

RATS format is a file format designed for the special problems of time series data. It has many advantages over other formats:

- Data access is more flexible: you can set up a single file with many series and select only the ones you need for a particular application. You can also select a specific set of entries—you don't have to read in entire series.
- The data series do not need to have identical structures. For example, you can mix monthly, quarterly and daily series on one file and can have series running over different intervals.
- RATS can automatically compact or expand data to match the current CALENDAR frequency. See Section 2.3.
- There are many ways to add data to a file or edit series already on a file.
- Data retrieval is extremely fast.

RATS also allows a certain amount of flexibility in reading the spreadsheet files (Section 2.6): you can select specific series off a file and restrict yourself to a subset of entries, and, under certain circumstances, you can compact and expand. However, those formats expect well-structured data, where all series are the same frequency.

Stored Information

For each series on the file, RATS retains the following information:

- the series name
- the frequency of the data (annual, quarterly, monthly, panel, etc.)
- the dates (or entries) for which the series is available
- up to two lines of comments
- the data itself

Reading Data into RATS

To work with data from a RATS format file, you can either:

- Open the file using the OPEN DATA instruction, and then read the data using DATA (FORMAT=RATS), or
- Open the file using the Open RATSDta operation from the File menu, select (highlight) the series you want, and then use the Data (RATS Format) wizard from the Data menu to read in the data.

With RATS format, you can read any combination of series from the file. You can also read specific sets of observations from the full data set by using the start and end parameters on the DATA instruction. This is one of the most important advantages of RATS format.
Chapter 2: Data

If you omit the list of series when using DATA, RATS will read all the series on the file. Be cautious when working with large files, or you might read in many unneeded series.

RATS allows you to read data series which have different frequencies. The series will be converted automatically to the frequency you specified via the CALENDAR instruction or via the Data wizard. See Section 2.3 for details on frequency conversion.

There is one special restriction in reading a RATS data file: the name you use for a series in your program must be the same as the name for the series on the data file. That is, if M1 is called FMS on the data file, you will have to use the name FMS in your program also. Of course, you could certainly then copy the data to a series called M1 by doing SET M1 = FMS and use M1 in the remainder of the program. To change the name of the series as stored on the file, you can either open the file using File-Open RATSDATA and use the “Rename” toolbar button, or else open the file with DEDIT and use the RENAME instruction.

Creating and Editing RATS Format Files

RATS provides special instructions for creating and editing RATS format data files. Never try to make changes to a RATS data file using a text editor. The instructions for working with RATS format files are:

COPY When used with the option FORMAT=RATS, this writes a set of series to a new RATS format file. This is the easiest way to write data to a RATS file, but cannot be used to append data to an existing file—use DEDIT, STORE, and SAVE for that.

DEDIT initiates the editing of a new or existing file. Once you have opened a file using DEDIT, you can use the following instructions:

STORE adds specified series to the file, or converts the contents of a data file of another format.

SAVE saves changes to the file.

EDIT performs on-screen editing of an existing or new series.

INCLUDE adds a single series to the file.

CATALOG produces a directory of series on the file.

DELETE deletes a series from the file.

RENAME renames an existing series.

PRTDATA prints one or more of the data series stored on the file.

UPDATE makes changes to a small set of entries of an existing series.

QUIT ends an editing session without saving the changes.
The “Open RATSDATA” Window

As mentioned earlier, when you open a RATS file using File–Open RATSDATA, RATS displays a “RATSDATA” window showing the series on the file. This window allows you to do several things with the data:

- You can read data into memory by selecting the series you want from the Window and then doing Data (RATS Format) from the Data menu.
- You can view or edit the data in a series by just double-clicking on a series.
- You can add new series to the file—just click on the icon.
- You can rename a series by selecting the series and clicking on the Rename icon.
- You can generate time series, histogram, and box plots for one or more series, using the graphing toolbar icons: 
- You can display basic statistics (the same output provided by the TABLE instruction) using the icon.
- You can compute and plot the autocorrelations of a series by clicking on the icon.

Notes

RATS will automatically insert missing values for any entries which are out of the range of data on the file. If you want to find out what RATS actually does with the data as it reads them in, use the option VERBOSE. This will tell you whether it is compacting or expanding a series, and whether it has to pad with missing values.

Examples of DATA

Suppose the data file MYDATA.RAT has REAL_GDP, quarterly from 1947:1 to 2007:1, and TBILLS, monthly from 1946:1 to 2007:5.

cal(q) 1947
open data mydata.rat
data(format=rats) 1947:1 2007:1 real_gdp  tbills

will read REAL_GDP and quarterly averages (the default compaction option) of TBILLS, over the range 1947:1 through 2007:1.

cal(m) 1954
allocate 1999:12
open data mydata.rat
data(format=rats) / tbills

will read the (monthly) data for 1954 through 1999 for TBILLS.
Chapter 2: Data

Examples of Editing

dedit company.rat
    cal(q) 1950
    open data ascii.dat
data(format=free) 1950:1 2003:4 order invent shipment backlog
    store order invent shipment backlog
    save

reads four series from a free–format file and stores them on the RATS file
COMPANY.RAT.

cal(d) 1991:1:2
dedit(new) options.rat
    open data options.wk3
    store(convert=wks,org=col,adddates)
    save

this converts all data from the (undated) WK3 file OPTIONS.WK3, creating
OPTIONS.RAT. The data will be stored as daily data, beginning with entry 1991:1:2.

Using Older RATS Format Files

RATS Version 7 uses the same RATS file format as Versions 4 through 6, so files can
be interchanged seamlessly among these versions. It can also read from files created
with Versions 2 and 3 of RATS.

To read from an older RATS format file, just use OPEN DATA and DATA in the usual
manner. Similarly, to edit an older file, open the file as usual with DEDIT and then
use any of the usual editing commands. When you issue a SAVE instruction, RATS
will ask you if you want to convert the file to Version 7 format, or keep the file in
Version 2/3 format.

Note that older RATS format files do not support data frequencies that were added
with Version 4, including intra-day data and dated panel data. Such series are stored
as undated on Version 2/3 RATS files. The older format also does not support series
names longer than eight characters.
2.6 Spreadsheet and Delimited Text Formats

Supported Formats

Spreadsheets are a popular tool for data storage, and many commercial database vendors allow users to download data in a format readable to such programs. RATS supports several formats associated with such programs:

- **XLS**: Microsoft Excel XLS spreadsheets
- **WKS**: Lotus 123 Version 1.x, 2.x, and 3.x worksheet files (WK3, WKS, WK1, WRK and WR1 files)
- **PRN**: Print Files (text files with a spreadsheet–style layout)
- **CDF**: Comma–delimited text files
- **TSD**: Tab and space delimited text
- **DIF**: Data Interchange Format

XLS and WKS are typically the best choices from among these.

RATS supports these formats in several ways:

- You can read data from a file using the instruction **DATA**.
- You can write data to a file using the instruction **COPY**.
- You can convert data from a spreadsheet file directly to a RATS format file using the **CONVERT** option of **STORE**. Unless you are planning to maintain the data set in the spreadsheet, this is the recommended procedure for using these files.
- You can convert the data to RATS format using the utility program **RATSDATA**.

Required Form

RATS expects a spreadsheet to look something like the screen shot below. We refer to this as organization by **column** (series arranged in columns, one observation per row).
The series can also run across the rows: dates (if present) in the first row and each row of data beginning with the series label in column A. This is organization by row or variable. In either case, the block of data must be in the upper left hand corner of the worksheet.

**Dates**

The column (or row) of dates is optional. Dates permit more flexible use of the data file. If your file includes dates, the date information must appear in the first column or row of the spreadsheet. You can enter the dates as strings or as functions or values. If you do enter them as functions or values, be sure to format the cells with a date format.

If you use strings, the dates must be in the form year/period or year/month/day. You can actually use any of the following characters to separate the parts of the date:

/ . : - Q q M m P p

For instance, 2007-1 and 2007Q1 are also acceptable. Do not include blanks within a part of the date: use 2007:03:10 rather than 2007:3:10.

With annual data, you can use just the year number in the data file (2007 rather than 2007:1), but RATS will only recognize this as a date if it is entered as a string rather than a numeric value. Remember that in your RATS programs themselves, you must include the “:1” for annual date references.

**Labels**

The labels at the beginning of each series are not optional. RATS uses them to determine which series to read from the file. It doesn’t matter how these are formatted. Labels should normally appear in row 1 (if organized by columns) or column A (if organized by rows). If you have additional rows of header information before the row of series labels (on data arranged by column), you can use the SKIP option on data to skip those.

As described under “Symbolic Names” in Section 1.11, variable names must start with a letter, consist of letters, digits, _ and $, and be sixteen characters or less in length (they can actually be longer, but only the first sixteen are significant). If you really don’t have the option of changing the names (for instance, you are downloading the data, and updating it frequently), your best strategy is to set up a short program to convert the data to RATS format and RENAME the series:

```
dedit prices.rat
open data prices.xls
store(convert=xls,org=columns)
rename gg:a0232  gga0232
rename gg:a1231  gga1231
save
```

You can then use DATA(FORMAT=RATS) to read the data from the PRICES.RAT file.
Chapter 2: Data

Reading Data

To read data from a spreadsheet file, use **DATA** with the appropriate **FORMAT** and **ORG** options. The series labels on the file allow RATS to identify where the different series are, so you do not have to read all series off the file. The labels also mean that you can list the series in any order on the **DATA** instruction. *If you skip the list of series parameter on DATA, RATS will read all the series on the file.*

Whether you can select specific observations out of the file depends in part on whether or not the file includes a date column:

- If there is no date column or row on the file, RATS assumes the first observation on the file corresponds to the first observation you request on **DATA**. You can ask only for entries at the beginning of the file, never a block in the middle or end. If you need to be able to read in a subset of the data, you could first convert the file to RATS format, using **STORE(CONVERT=...)**.
- If there are dates on the file, and the frequency on the file matches the current **CALENDAR** seasonal, **DATA** will locate the entries requested and fill in any skipped dates with missing values.
- If the frequency of the current **CALENDAR** does not match the frequency of the data on the file, RATS will compact or expand the data as it reads it in. See Section 2.3.
- If you don’t use a **CALENDAR** instruction, RATS will read observations from the beginning of the file. Any dates on the file are ignored.

You can also use the *Data Wizard* to read data from a spreadsheet. Just select *Data (Other Formats)* from the *Data* menu and specify the appropriate file format (such as “Excel Files (XLS)”), and select the file you want to read. The Wizard will walk you through the rest of the process. See Section 1.1.2 and the Help system in RATS for more information.

Writing Data

There are several ways to write data to a spreadsheet file. The most common is to use the **COPY** command to write data series. Begin by opening a file for output with **OPEN COPY**. Then use **COPY** to write the data to the file. We recommend that you use the **ORG=COL** option. You can include dates on the file with the **DATES** option. Do a **CLOSE COPY** command if you want to be able to open the file in another application right away. You must write the entire file using a single **COPY** instruction—you cannot use multiple **COPY** commands to append data to existing spreadsheet files.

You can also write series to a spreadsheet by doing *Show Series Window* from the *Data* menu, selecting the series you want to export, and doing *File–Export*.

Finally, you can also create a spreadsheet from data in a RATS format file using the instruction **PRTDATA**.
Chapter 2: Data

Examples

Make sure that you use the proper extension (XLS, WKS, or PRN for example) on the file name. RATS will not add it automatically.

```
  cal(q) 1999:1
  all 2000:3
  open data sample.xls
  data(format=xls,org=col) 1999:1 1999:4 cge fge
```

reads series CGE and FGE, for the dates 1999:1 through 1999:4, from the sample worksheet earlier in this section.

```
  cal(q) 1999:1
  open data sample.xls
  data(format=xls,org=col) 1999:1 2000:3
```

This reads all the series on the sample file. Because there are no data for 2000:2 and 2000:3, those entries are filled with NA.

```
  open copy sample.xls
  copy(format=xls,org=col,dates) 1999:1 2000:1 ige fge cge
```

produces the sample worksheet from earlier in the section.

```
  cal(m) 2002:1
  all 2006:12
  open data daysales.wk3
  data(format=wks,org=col,compact=sum)
```

In the above example, even though the file is WK3, you can use FORMAT=WKS, which works for all of the various Lotus formats. This also sums daily data on the file to create monthly data in working memory.

```
  cal(q) 1950:1
  open data sales.xls
  data(format=xls,org=col,sheet="quarterly",verbose) 1950:1 2006:4
```

This reads data from an Excel workbook containing several spreadsheet pages. Here, we are reading the page entitled “Quarterly”. The VERBOSE option produces information about the data file, including the frequency and starting date. It will also indicate if any frequency conversion is being performed to match the CALENDAR frequency.
Adding Dates to a Spreadsheet

Here’s an easy way to add monthly or quarterly date labels to a spreadsheet file if your program has an `@date(year,month,day)` function (Lotus and Quattro have this, Excel uses `=date`). First, format the first column (or row, if your data are organized by rows) of the worksheet as one of the date formats. Next, enter the date number corresponding to your first date in cell A2 (or B1 for `ORG=ROWS`) using the `date` function. In Excel, for example, you could enter the date June 1, 1999 with the formula:

\[ \text{date}(1999, 6, 1) \]

To fill the rest of the date column or row with monthly dates, enter the following formula in cell A3 (or C1):

\[ \begin{align*} 
\text{=date(year(a2+31),month(a2+31),1)} & \quad \text{(Excel)} \\
@\text{date}(@\text{year}(+a2+31),@\text{month}(+a2+31),1) & \quad \text{(Lotus, Quattro)} 
\end{align*} \]

If `ORG=ROWS`, use `+b1+31` in the formulas, rather than `+a2+31`.

Finally, copy this formula to the rest of the cells in the date column or row. For quarterly dates, just use `+92` in place of `+31`.

Troubleshooting

Because spreadsheets are designed to be human-readable more than machine-readable, RATS and RATSDATA can run into some problems interpreting them. If your worksheet seems to conform to the restrictions above, but RATS doesn’t read it correctly or generates error messages (such as “unexpected end of file”), first check the file and your program to make sure that you’re requesting the appropriate number of observations.

Then, check the number of rows and columns the program is processing by including the `VERBOSE` option on `DATA`. Is it different from what you expect? (Be sure you count the label and date rows/columns). If so, try deleting or erasing several (apparently blank) columns to the right and rows below your data table. There probably are some errant cells out there which you can’t see. As an alternative, you can also try extracting the block of data to a new file.

If at all possible, stay away from `PRN`: it is not a well-defined format and access is relatively slow. Use RATS to convert the data set.

If you have a `DIF` file created by another program, there may be some extraneous information on the file, such as a header line or lines. If so, delete this information by editing the file in a spreadsheet program, or else use the `SKIP` option on the `DATA` instruction to skip the lines preceding the row of series names.
2.7 Database Formats, ODBC, and SQL

All versions of RATS can read and write data in the DBF format used by dBase III and dBase IV and compatible programs. The Professional version of RATS can also read data from any database format that supports Open Database Connectivity (ODBC) and SQL.

Reading Data Using ODBC and SQL (RATS Professional Only)

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

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

\[ \text{open odbc } \text{dsn} \]

where “dsn” is the Data Source Name for the database.

You can then read data from the database by executing a DATA command with the SQL option, providing a SQL command string that reads in the desired data. The SQL command creates a table (internally) which is then processed similar to the spreadsheet files in Section 2.6.

For example, the following reads in dated closing price data from a file called PriceData, in a Data Source named StockData, and then displays the data using PRINT.

\[
\begin{align*}
\text{calendar(d) 2006:1:1} \\
\text{allocate 2006:03:31} \\
\text{open odbc "StockData"} \\
\text{data(format=odbc,$sql="select Date,ClosingPrice from PriceData order by MarkDate")}
\end{align*}
\]

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

\[
\begin{align*}
\text{cal(m) 1995:1} \\
\text{open odbc "Sales"} \\
\text{data(format=odbc,compact=sum,$sql="select date,sum(subtotal) as sales from invoice group by date order by date") 1995:1 2006:12}
\end{align*}
\]
dBase DBF Files

For reading and writing dBase DBF files, use the option FORMAT=DFB on your DATA or COPY instruction. No ORGANIZATION option is required.

When reading data, you can specify the names of the series you want to read, or omit the list to read in all the series on the file. You can also use the Data Wizard to read these files. Select Data (Other Formats) from the Data menu, choose “dBase Files (*.DBF)” in the file type field, and open the data file. The wizard will walk you through the rest of the process. See Section 1.1.2 for more.

The field names in the database serve as the series names, and each record is treated as a separate observation. If the file includes a date field as the first field, RATS will be able to make use of the date information. If you have, say, quarterly data, this could be a “date” field which gives a date within each quarter, or a “character” field which has the dates as strings in the typical format for quarterly labels (1975:1 or 1975-1, for example). All series on the file should have the same frequency.

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

```
calendar(d) 1996:1:2
allocate 2005:12:31
open data stockdat.dbf
data(format=dbf)
```

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

FORMAT=DFB can also be used on PRTDATA for copying data out of a RATS format file.
2.8 Text Files: Free Format

What is Free–Format?

Free–format files are simply text files containing a collection of numbers separated by blank spaces, commas, tabs, or carriage-returns. Free format files can be very convenient if you want to type in a short data set, because you can create them with any editor that can save files as plain text. They are also the most natural format for data scanned in from printed material. However, because these files do not contain series labels or date information, they are not good for long-term use, and RATS cannot read them with much flexibility. Usually, you will want to convert free-format files to RATS or one of the spreadsheet formats.

If you have a text file that does have variable labels, date-format strings (such as “yyyy/mm/dd”), or other non-numeric characters you can:

- edit out any non-numeric characters other than labels and (optionally) dates, and read the file with FORMAT=PRN. This allows RATS to read the series labels and dates from the file. Note that PRN files must have labels. If you have a well-formatted text file which doesn’t have variable labels, you may find it convenient to add labels and then read the file as PRN format. See Section 2.6.
- edit out all the non-numeric characters and read the file as FORMAT=FREE.
- if the non-numeric characters appear in some sort of regular pattern, you may be able to read the file using a FORTRAN format. See Section 2.9.

Reading Data

With FORMAT=FREE, RATS reads series line by line, according to the following guidelines:

- If your data are organized by row, each variable must begin on a new line. The data for a given series can, however, extend over more than one line—just make sure that the data for the next series begins on a new line.
  
  If DATA does not find enough entries on a line to fill a series, it automatically moves on to the next line and continues reading numbers.

- If your data are organized by column, the data for each new observation must begin on a new line. As above, data for a particular observation can extend over multiple lines.
  
  If DATA does not find enough entries on a line to fill all the series for an observation, it automatically moves on to the next line and continues reading numbers. When it has read data for all series for that observation, it drops to the next line to start reading in the next observation.

- RATS interprets the characters NA (upper or lower case) or a period followed by zero, one, or two non-numeric characters as a missing value. If it encounters any other non-numeric characters, RATS will interpret them as a missing observation and display a warning message.
You can also use the Data Wizard to read these files. Select Data (Other Formats) from the Wizards menu, specify “Text Files” from the file type field, and open the data file. The wizard will walk you through the rest of the process. Because there are no series names on the file, RATS will display dialog boxes prompting you to enter names for each series. See Section 1.1.2 for more.

**Line Length**

RATS can handle lines of any length.

**Data Output**

To create a free-format file, open the file for output with **OPEN COPY**. Use the **COPY** instruction to write data series to the file. With **FORMAT=FREE**, **COPY** will only write out the data itself—it will not include series labels or dates. Use **FORMAT=PRN** or **CDF** if you want to produce a text file which includes series labels or dates.

**Troubleshooting**

The **FORMAT=FREE** option has one advantage over other formats: it allows data for a single observation (or a single variable) to cover several lines. The disadvantage of this is that it becomes more difficult to pinpoint errors. If you forget one number early in the file, **DATA** will automatically pull in the next line to fill the omitted value and throw the data off.

If you get the error “Unexpected end of file...” when reading with **FORMAT=FREE**, it means that RATS reached the end of the data file before it filled all of the data series. To determine what happened, do the following:

- Check your **ALLOCATE** range or the **start** and **end** parameters on the **DATA** instruction to make sure you are reading the range that you intending. (If you have **ALLOCATE 2003:5:14**, do a **DISPLAY 2003:5:14** to see how many data points are implied by the **ALLOCATE** range).
- Make sure that you have the proper **ORGANIZATION** option.
- Check that the data file has the number of observations and variables that you expect, at least at first inspection.
- If all seems to be correct and it’s a small enough file, you can do a quick check for typographical errors.

If all this fails to locate the problem, you will have to let **DATA** help you find the problem. For illustration, suppose you have a file that looks like this:

```
1,2,3
10.20,30
100,200,300
1000,2000,3000
```
This is supposed to have 4 observations on each of three series, but the second line has a decimal point between the 10 and the 20, where it should have a comma, so the line only contains two values (10.20 and 30) rather than three. If we read this with

\[ \text{data(org=col) 1 4 first second third} \]
\[ \text{print} \]

we will get the error message,

\text{Unexpected end of file while processing line 5. (series FIRST entry 4).} \]

We can tell the following from this:

1. \textbf{DATA} thinks it needs a fifth line to read the requested data. Our data set is supposed to have four lines—this tells us that, in fact, the file has as many lines as we think.

2. \textbf{RATS} was trying to read the fourth observation from a fifth line, so for some reason we are precisely one line off at the end of the file.

We can find the bad observation by examining the values of the series with \textbf{PRINT}. We look for values which have ended up in the wrong series (observation 2 will have 30 for series \texttt{SECOND} and 100 for series \texttt{THIRD}) or try to locate the place where one of the series gets off sequence.
2.9 Text Files: FORTRAN Format

When Do I Use It?

Many text files can be read with the default FORMAT=FREE option or as FORMAT=PRN. On occasion, however, you may encounter a text file which RATS cannot read using these options—perhaps because it contains information other than just the data: it might, for instance, have a header before each series. There are three ways to deal with such a data set:

- If the file has labels for each variable, you may be able to modify the file so that it can be read with FORMAT=PRN.
- You can edit the file, stripping out all information except the numbers, and then use FORMAT=FREE.
- If the file has the proper structure (as described below) you can use a FORTRAN format.

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

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

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

Fw.d For numbers with a fixed decimal place: a total of w positions (total digits to the left and the right of the decimal) with d digits to the right of the decimal. Very large and very small numbers will not fit an F format. For example, the number 1000000. cannot be printed as F15.8.

Ew.d For numbers in scientific notation. When writing files, this format is useful because it can handle a number of any magnitude, but it can be difficult to pick out large and small values at a single glance. For example, it takes a bit of work to see that the first of 1.343E-02 and 8.547E-03 is the larger.

Gw.d This is a mixture of the F and E formats. If a number can reasonably be printed with the F format, that is used. All others (very large and very small) are displayed in the E format.
Chapter 2: Data

\( I_w \) For integer numbers. If there is a fractional part to the number, it is ignored.

\( A_w \) This is used (by \texttt{COPY}) only for the date strings. It prints a character string, left justified, in a field of width \( w \).

\( w \times \) Indicates \( w \) spaces

\(/\) Indicates a skip to the next line

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

The FORMAT Option

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

\[
\text{FORMAT}="( \text{format string } )"
\]

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

\[
data(\text{org}=\text{col},\text{format}="(f8.5,2x,f8.4,2x,f8.5,2x,f8.3,2x,f8.5,2x,f8.5)"
\quad 1947:1 1979:4 \text{ wage interest stocks mortgage charitable misc} )
\]

With ORG=ROWS

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

\[
data(\text{format}="(11x,4f15.7)") \quad 1 \ 100 \ gnp \ m1
\]

\[
\text{read}(n,1000) \quad (\text{gnp}(i),i=1,100)
\quad \text{read}(n,1000) \quad (\text{ml}(i),i=1,100)
\quad \text{format}(11x,4f15.7)
\]

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

With ORG=COLS

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

96 RATS User’s Guide
data(org=col,format="(f9.4,2x,f10.3,2x,f9.4)")  1 100 gnp m1 ipd

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

Mixed Character and Numeric Data

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

Suppose you have the following data file:

<table>
<thead>
<tr>
<th>Value</th>
<th>Label</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0450</td>
<td>ME</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2.3210</td>
<td>MS</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1.8930</td>
<td>MN</td>
<td>2.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>

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

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

The following data set was used in Section 2.2. We show here how to use FORTRAN format to simplify the process of reading the data.

<table>
<thead>
<tr>
<th>FIRM A</th>
<th>1996</th>
<th>11.3</th>
<th>11.6</th>
<th>10.9</th>
<th>12.3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1997</td>
<td>13.0</td>
<td>12.8</td>
<td>11.5</td>
<td>12.5</td>
</tr>
<tr>
<td></td>
<td>1998</td>
<td>12.9</td>
<td>13.0</td>
<td>13.2</td>
<td>13.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIRM B</th>
<th>1996</th>
<th>22.5</th>
<th>21.9</th>
<th>24.3</th>
<th>25.6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1997</td>
<td>21.9</td>
<td>21.8</td>
<td>22.6</td>
<td>23.5</td>
</tr>
<tr>
<td></td>
<td>1998</td>
<td>22.5</td>
<td>25.0</td>
<td>24.5</td>
<td>25.4</td>
</tr>
</tbody>
</table>

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

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

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

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

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

```
data(format="(//(12x,4f9.1))",org=rows) / firma
data(format="((12x,4f9.1))",org=rows) / firmb
```
2.10 Haver Analytics and Other Commercial Databases

RATS supports several formats for commercial databases. The Haver Analytics databases available through Estima are provided in both their native Haver format (DLX) and in RATS format. RATS format is covered in Section 2.5. To access the data in DLX format, you need to do the following:

```
open haver name of database without extension
data(format=haver, other options) start end list of series
```

You can use any of the options for expanding or compacting data from Section 2.3. For instance, if you read monthly data when you have a quarterly calendar, by default, the DATA instruction will compact the series by quarterly averaging.

You can read data from more than one Haver database—do the OPEN HAVER and DATA instructions for one, then a new OPEN HAVER and DATA instruction(s) for the other.

The following reads six series (GDP, GDP deflator, gross private domestic investment, unemployment rate, one year T-bill rate and M2) from the USECON database. USECON is a 12,000 series database with a broad range of key U.S. macroeconomic series.

```
open haver m:\data\haver\usecon
data(format=haver) / gdpq dgdp fq lr fcm1 fm2
```

The OECD Main Economic Indicators are also available from Estima. These are provided in a set of RATS format data files with one for each country. To access data for several countries, execute several OPEN DATA and DATA instructions:

```
open data canoeecd.rat
data(format=rats) / canrgdps canm1s cancd90d cancpinf canusxsr
open data usaoecd.rat
data(format=rats) / usargdps
```

RATS can also read the Basic Economics database (formerly known as Citibase), which is maintained by Global Insight. You must first use the Preferences operation on the File menu to tell RATS where the data are located on your hard disk. With that done, you read data with

```
data(format=dri,other options) start end list of series
```
Chapter 2: Data

2.11 FAME Format Databases

Reading and Writing FAME Data

Support for reading and writing FAME format files is provided in the Professional versions of WinRATS or UNIX RATS.

As with the other data file formats supported by RATS, you use the instructions DATA and COPY, respectively, to read series from and write series to a FAME data base. For FAME format files, you use the option FORMAT=FAME on DATA or COPY (the ORG option is irrelevant).

A simple command to read data from a FAME file would be:

```
open data famefile.db
data(format=fame) / income expenses sales
```

Because FAME data bases often contain a very large number of time series, you must specify the names of the series you want to read in when using the DATA instruction. This is different from some other file formats, such as RATS and XLS, where an empty series list causes RATS to read all the series on the file.

To add series to an existing FAME file, use the APPEND option on the OPEN COPY instruction. For example:

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

We recommend that you always use OPEN(APPEND), rather than OPEN, unless you explicitly want to replace an existing data base.

FAME Objects Supported by RATS

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

Symbolic Name Conversion Between RATS and FAME

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

The lookup table file must be an ASCII file, containing two columns of text separated by at least one space. The first column on each line is a RATS series name, while the second column is the associated FAME name. See the next page for further details on the structure of lookup tables.
Using Lookup Tables

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

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

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

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

If you are interested in upgrading to the Professional version of WinRATS or UNIX RATS, please contact Estima.
Chapter 2: Data

2.12 PORTABLE Format

What Is It?

FORMAT=PORTABLE is a text version of the RATS file format. PORTABLE includes all of the information stored on the RATS file, but in text form. You can create PORTABLE format files using:

• COPY with FORMAT=PORTABLE for data currently in memory.
• PRTDATA with FORMAT=PORTABLE for series stored on a RATS format file.

An example of a COPY instruction and the layout it produces is:

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

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

The key elements are the series name (on a separate line), the line containing the frequency and starting and ending dates, the header lines, and the row of === which ends the header. The data are printed four entries to a line.

DATA with FORMAT=PORTABLE

You can read a data set in PORTABLE format into RATS using DATA with FORMAT=PORTABLE. You can read any series from a PORTABLE file, in any order.

Archiving Data

The most important use of PORTABLE format is to create archival human-readable copies of RATS format files, using the PRTDATA instruction. For example:

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

This writes all of the series stored in the RATS file MYDATA.RAT to a text file called ARCHIVE.TXT.
2.13 Binary Data

When To Use It

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

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

Thus, you should only use `FORMAT=BINARY` under the following conditions:

- The data set is quite large, so the speed advantage is noticeable.
- The data set is fixed—you won’t need to change it during your work.
- You will always be using exactly the same `DATA` instruction to access the data set. This `DATA` statement should be identical in form (ORG options, entry range, and series), to the `COPY` instruction which produced the data file. (Or the `READ` must be equivalent to the `WRITE` if you are working with arrays).

You can only read data in binary files created by RATS if they were originally written using `COPY FORMAT=BINARY` or `WRITE FORMAT=BINARY`.

Interfacing with Other Programs

You may be able to read binary data written by another program, or read into another program binary data created by RATS, but only if it uses a compatible compiler. You must also be very careful to read the data back in exactly the same way as it was written:

- Real-valued data use a 64-bit representation. This is double precision, except on machines with 64-bit single precision, like a DEC Alpha.
- With `ORG=ROWS`, RATS reads (or writes) each `series` with a single call, so you should use a separate FORTRAN `WRITE (READ)` for each as well.
- With `ORG=COLS`, each `observation` is read (written) with a single call; again you should use a separate `WRITE (READ)` for each.
2.14 File Handling Tips and Tricks

Can’t Open a File?

If you have created a data file in RATS and are having trouble opening the file in another application, you need to make sure that RATS doesn’t still have the file open. For example, if you are working in interactive mode and do:

```
open copy test.dat
copy(format=prn,org=columns) / x y z
```

and then immediately try to open TEST.DAT in another application while RATS is still running, you will probably get an error message. And, if you view a directory listing of TEST.DAT using Windows Explorer or a similar utility, it will appear as a zero-byte file. That’s because RATS still has the file open. In order to access the file in another application, just close the file by issuing the command:

```
close copy
```

or else by quitting the RATS application.

File Units

In RATS, you open a file by associating a file name with an input/output “unit.” For example, the statement:

```
open data myfile.txt
```

simply associates the filename “myfile.txt” with the DATA unit, which is one of the reserved unit names in RATS. Every RATS instruction that can read or write data has a UNIT option which allows you to specify the source or destination unit for that operation.

Fortunately, this business of file unit names is generally transparent to the user, because all of the relevant instructions have default settings for the UNIT option. For example, the DATA instruction applies to the “DATA” unit by default. So, you can read data from a file by simply doing something like:

```
open data sample.rat
data(format=rats)
```

Because DATA is the default unit for the DATA instruction, it will automatically read the data from the SAMPLE.RAT file, which has been associated with the DATA unit by the OPEN command. If no file has been associated with the DATA unit, RATS will prompt you for a filename.

In some cases, however, you may find it helpful to make use of the UNIT option, particularly when you want to have several data files or output files open simultaneously. This is made easy by the fact that RATS lets you define your own unit
names. For example, suppose you want to read data from two different files in your program. You can either:

- Use an **OPEN DATA** command to associate the first file with the **DATA** unit, and then read the data in using **DATA**. Then, repeat this procedure for the second file:
  
  ```
  open data first.rat
  data(format=rats) / x
  open data second.rat
  data(format=rats) / y
  ```

- Or, you can define your own custom unit names, and associate one file with each name. You can then read from the files in any order by specifying the appropriate unit name. For example:
  
  ```
  open data1 first.rat
  open data2 second.rat
  data(format=rats,unit=data2) / y
  data(format=rats,unit=data1) / x
  ```

Because you are using different unit names for each file, both files remain open, so you can go back and read from either one without having to repeat the **OPEN** command.
Chapter 3
Graphics

*RATS* offers three main graphing instructions: `GRAPH` for time-series graphs, `SCATTER` for x-y graphs, and `GCONTOUR` for creating contour graphs. All three instructions offer a very large collection of options, allowing you to create publication-quality graphs easily. Aside from a variety of basic graph types (such as line graphs and bar graphs), RATS offers a number of special effects, such as overlay graphs (displaying on a single frame data in two different forms or with two different scales) and arrays of smaller graphs.

Graphs created can be exported in several well-known formats (such as PostScript) for placement in documents. You can adjust the appearance of lines and fills to match the needs of your publication.
3.1 Graphics

The section provides general information on working with graphs in RATS. We will cover the instructions used to generate graphs, as well as the parts of the RATS interface which deal with graphs, including exporting graphs for use in other applications.

Overview of the Graphics Instructions

RATS has three main instructions for creating graphs:

- **GRAPH** creates time series plots, or, more generally, plots of a series which is arranged as a sequence, such as autocorrelations or lag coefficients.
- **SCATTER** creates scatter (x vs y) plots. In addition to series vs. series, it does graphs where the x axis is a grid on the real line, either equally or unequally spaced.
- **GCONTOUR** produces contour plots.

These are very flexible instructions which allow you to create a wide variety of graphs.

In addition, three auxiliary instructions: **SPGRAPH**, **GRTEXT**, and **GRPARM** provide even more flexibility. Among the capabilities:

- You can use either color or dashed lines and patterns to distinguish series. RATS will automatically translate color into patterns on black and white output devices.
- **GRAPH** can produce line graphs, filled line graphs, several types of bar graphs, and high-low-close graphs.
- **SCATTER** can use symbols or dots to show scatter points, and allows you to connect consecutive points. It will also do bar graphs and filled line graphs if the (x,y) series are organized correctly.
- You can generate a graph with series having different scales (which we refer to as “overlay graphs”) using just a single **GRAPH** or **SCATTER** instruction. The overlay options also allow you to mix different graph styles on a single graph. For example, you could plot one series as a line and another series using a bar-graph style.
- You can create pages of equally sized “postage-stamp” graphs.
- You can use graph style sheets to set the color and thickness of lines, choose from a range of fill patterns for bar graphs, and more.

The **GRAPH** and **SCATTER** instructions were introduced in the tutorial in Chapter 1. For complete details on these and the other instructions mentioned in this Chapter, see the remainder of this Chapter and the relevant sections in the *Reference Manual*. 
3.2 Displaying Graphs on the Screen

Interactive Mode

When you are working in interactive mode and either create a new graph using `GRAPH`, `SCATTER`, or `GCONTOUR`, or open an existing graph using `File–Open Graph`, RATS displays the graph in a graph window.

By default, it will use a separate graph window for each graph, although you can use the `Preferences...` operation on the `File` menu to tell RATS to re-use the same graph window for each new graph.

Graph windows are much like any other type of window: you can move or size them, switch back and forth among other open windows, and so on. They also make it easy to save, print, export, and copy and paste graphs, as described on the following pages.

When you have a graph as the active window, you can use the toolbar icon: ![Graph Color Switch Icon] to switch between viewing the graph in color or in black and white. Color is usually easiest to read on the screen, but if you need to see (roughly) how the graph will look if done in black and white, you can see it that way.

It’s quite possible for a RATS program to generate a large number of graphs. You can quickly eliminate the graph windows without losing other information by using the `Close All Graphs` operation on the `Window` menu.

Batch Mode

Graphs are generally not displayed to the screen when you run in batch mode (the Windows version does display all graphs in a single graph window in batch mode). Use `OPEN PLOT` or `ENVIRONMENT GSAVE` instructions in your program if you want to save your graphs to disk (see the following pages for additional information on saving graphs).
Chapter 3: Graphics

3.3 Saving and Opening Graphs

Overview
RATS offers three ways to save graphs:

The File–Save As... Operation
In interactive mode, you can save a graph using the Save As... operation from the File menu. By default, RATS will save the graph in Estima’s own RGF (RATS Graph File) format. Use the “Save File as Type” field in the dialog box to select another format, such as PostScript or Windows Metafile.

You can use this feature to convert existing RGF files to other formats as well. Use File–Open Graph... to open an RGF file, and then use Save As... to save it in another format.

“OPEN PLOT filename”
If you issue an OPEN PLOT instruction, any subsequent graphs you generate will be saved to the selected file in RGF format. This works in both interactive and batch modes, and offers the option of saving multiple graphs to a single file. See the next page for more information.

“ENVIRONMENT GSAVE=template GFORMAT=format”
The ENVIRONMENT instruction with the GSAVE=template parameter is your best choice for automatically saving multiple graphs into separate files (one graph per file), and for automatically saving files in formats other than RGF, such as PostScript. You supply a filename “template,” and RATS saves any subsequent graphs to disk using that template plus a sequence number for the filename. By default, this saves graphs in RGF format—add the GFORMAT parameter to save the graph in other formats. See the next page for more information.

Note that on most platforms, you can also use the Copy operation on the Edit menu to copy graphs to the clipboard for pasting into other applications. See “Preparing for Publication” on page 113 for more information.

Available Formats
All versions of RATS can save graphs in Estima’s own RGF (RATS Graph Format) file format, or in PostScript format. The Windows versions can also save a graph as a Windows Metafile or HPGL (Hewlett Packard plotter format) file; on the Macintosh you can save as PICT. Use RGF format if you want to be able to re-open the graph in RATS for viewing or printing. Note, though, that RGF is not designed for permanently saving graphs. RGF is likely to change with each release of RATS, so you will only be able to read files created with the same version that wrote them.

Use PostScript, Windows Metafile, HPGL, or PICT if you are going to be importing into other applications. PostScript provides the highest quality image and is the format that we have used in this manual.
Section: Using “ENVIRONMENT GSAVE=template”

The *template* will usually be a filename string that includes an asterisk (*) symbol. For each graph you generate, RATS will construct a filename by replacing the asterisk with the sequence number of that the graph, and will save the graph to disk using the constructed filename. You can omit the asterisk if you want to save a single graph under a specific name.

The GFORMAT=paramater controls the format in which the graph is saved. Choices for format parameter: RGF, PORTRAIT, LANDSCAPE, WMF, HPGL, and PICT (PORTRAIT and LANDSCAPE are PostScript format, saved in portrait and landscape orientations, respectively—see page 114). The availability of some formats is platform-dependent, as described on the previous page. The default is RGF format.

The following code plots a series of impulse response graphs, and saves them in a set of PostScript format files named IMPRSP1.EPS, IMPRSP2.EPS, and so on.

```r
environment gsave="imprsp*.eps" gformat=portrait
list ieqn = 1 to neqn
do i=1,neqn
   graph(header=header(i),key=loleft,number=0) neqn
   cards impblk(i,ieqn)
end do i
```

Section: Using “OPEN PLOT filename”

The OPEN PLOT instruction causes RATS to save all subsequent graphs generated by your program to the selected file, until you either do a CLOSE PLOT instruction, or use another OPEN PLOT instruction to open a different graph file.

There is no problem with putting several pages of graphics on a single file if you just intend to print them. However, if you are going to translate a graph to another format, you should put only one graph on a file. If your program generates several graphs, you can either use a separate OPEN PLOT instruction for each graph, as shown below, or use the ENVIRONMENT GSAVE instruction.

```r
open plot gdp.rgf
graph(header="US Real GDP")
# gdp90
open plot consump.rgf
graph(header="Consumption of Durable Goods")
# gcd
```

Section: Opening Saved Graphs

If you have saved a graph in RGF format, you can re-open it for viewing, printing, or exporting to another file format. Select the Open Graph... operation from the File menu and open the file you want to see. You can then use File–Print to print the graph, File–Save As... to save the graph in another format, or Edit–Copy to copy the graph for pasting into another application.
3.4 Printing Graphs

In Interactive Mode

To print a graph, do the following:

1. Make sure the desired graph window is active.
2. Select Print from the File menu, or click on the “Print” toolbar icon.

Alternatively, you can have RATS print the graphs automatically by including the instruction \texttt{ENVIRONMENT PRINTGRAPHS} in your program (prior to the \texttt{GRAPH} or \texttt{SCATTER} instructions). Use \texttt{ENVIRONMENT NOPRINTGRAPHS} to turn off the automatic graph printing.

To print a graph saved in RGF format in an earlier session, just use \textit{File–Open Graph...} to open the file, then do \textit{File–Print}.

In Batch Mode

To print graphs generated in batch mode, you can do one of the following:

- Save your graphs to disk, then open and print them using RATS in interactive mode as described above.
- Include an \texttt{ENVIRONMENT PRINTGRAPHS} instruction in your program, prior to your graph instructions. RATS will print the graphs automatically. Alternatively, you can include the switch “/PRINTGRAPHS” on the command line used to run the program.

Notes on Printing

The Windows and Macintosh versions of RATS use standard print drivers to communicate with the printer. Use the \textit{Print Setup...} or \textit{Page Setup...} operation on the \textit{File} menu to select a printer and set any of the options supported by your printer.
3.5 Preparing for Publication

You have three ways to include RATS graphs in a publication:

- Export the graph to one of the standard graphics file formats and then import the graph into your word-processor or page-layout program.
- Use Edit-Copy to copy a graph to the system clipboard, and then paste it into your word-processing application.
- Print a hard copy of your graph directly from RATS and include this in your publication.

The first two are obviously more flexible, since most word processors will allow you to size and shape the graph to fit your document. Many programs, particularly Windows and Macintosh applications, will also allow you to edit elements of the graph, add your own lines, symbols, or text.

PostScript is the preferred format because it allows for the most accurate translation of the graph, and works very well if you will be producing your output as a PDF file, or on a PostScript printer.

Windows Metafile format (Windows only) and PICT format (Macintosh only) can also produce good quality images in many situations. These are the formats used when copying and pasting a graph.

HPGL and PIC are older formats that are still supported in RATS, and which can sometimes be useful. They do have limitations though. For example, HPGL format cannot handle graphs created with the STYLE=POLYGONAL option.

To save a graph in a particular file format, just use the Save As... operation on the File menu, and select the desired file format from the dialog box before saving the file, or use an ENVIRONMENT instruction with the GSAVE and GFORMAT parameters prior to generating the graphs.
Copying and Pasting Graphs

Another way to get a RATS graph into a word processor or page-layout program is to copy the graph to the system clipboard and then paste the graph into your document.

To copy a graph to the clipboard:

1. Make sure the desired graph window is active.
2. Choose the Copy operation from the Edit menu.

Windows versions copy both bitmap and Windows metafile versions of the graph to the clipboard, while Macintosh versions copy PICT and PostScript versions.

To paste the graph, switch to the other application and select the Paste or Paste Special... operation from the Edit menu (Paste Special... allows you to select the format that is pasted).

Graph Orientation

By default, RATS creates graphs with “landscape” proportions, where the graph is wider than it is high. When you print a graph with the Windows or Macintosh version, you can use the Print Setup... (or Page Setup... operation) on the File menu to choose between a “portrait” or a “landscape” orientation on the page. RATS also allows you to choose between Portrait or Landscape orientation when saving graphs in PostScript format. Both choices are available through the “Save File as Type” field in the Save As... dialog box and the GFORMAT parameter on ENVIRONMENT. The resulting layouts are shown below:

**Portrait:**

**Landscape:**
You can get two additional layouts by using the PORTRAIT option on the GRPARM instruction (you do GRPARM before generating the graph) to produce a graph that is rotated so that it is taller than it is wide. When used in conjunction with the “Portrait” vs. “Landscape” printing/exporting choices described above, you can get the two orientations shown below:

Together with the PORTRAIT/NOPORTRAIT option on GRPARM, these produce basically the same four layouts described above.

Changing the Graph Proportions and Appearance

It is very easy to change the proportions of the graph. Whenever you resize a graph window, RATS automatically redraws the graph to fit the new window size. Fonts, key boxes, time axis labels, and other graph elements will be changed or rescaled as necessary to fit the new dimensions.

When you print, save, or export a graph, you have a choice as to how the resulting output will look. By default, RATS outputs the graph using standard size and proportions, regardless of how the graph window looks on the screen (see examples above). This allows you to arrange the windows on the screen in the way you find most convenient without affecting how the graph will look in printed output.

If you want to change the proportions of the graph, adjust the graph window to get the appearance you want, and then click on the “Fix” toolbar icon. This will fix or “freeze” the graph. After this, when you change the size or shape of the graph window, the size will change, but the proportions will not. Any exported or printed graph will have the proportions that you see in the graph window.
Chapter 3: Graphics

This allows you to select the desired proportions (height vs. width), and make use of RATS’ ability to optimize font sizes, label placements, etc. You can then use your word-processor’s ability to proportionally size the exported graph. This gives a graph of the desired proportions and size without any ill-effects from having your word-processor “stretch” the graph in either direction.

If you want to change the appearance of the graph again, just click on the “Unfix” icon to “unfreeze” the graph. Note that even if you fix the proportions, the final appearance of a printed or exported graph may be slightly different than what you see on the screen, because of font size substitutions.

Customizing Colors, Lines, and Patterns

The representation parameter on the supplementary cards of GRAPH and SCATTER allows you to control how series are displayed, including selecting the color used for color graphs or the patterns used for black and white graphs. The default choices are designed to work well in most circumstances, but, if desired, you can use “graph style sheets” to customize these choices to suit your preferences or organizational standards.

Style sheets are external files that allow you to define up to 30 different choices each for color lines, grayscale lines, color patterns, grayscale patterns, color symbols, and grayscale symbols. You have control over the following attributes:

- For lines in color graphs: you can set the line color, the pattern (solid line or one of seven dashed patterns), and thickness.
- For lines in grayscale graphs: the grayscale level, pattern, and thickness.
- For fill patterns (as used in bar graphs and similar styles) in color graphs: the color and the pattern, with seven patterns from which to choose (solid and six types of “hatching”).
- For fill patterns in grayscale: the grayscale level and the pattern.
- For symbols in color graphs: the shape (twelve choices), the color, and whether or not the symbol is filled or not filled.
- For symbols in grayscale graphs: the shape, the grayscale level, and whether or not the symbol is filled or not filled.

To load an existing style sheet into memory, use an OPEN command followed by a GRPARM instruction with the IMPORT option.

For information on creating graph style sheets or modifying existing style sheets, please see the PDF file entitled “RATS Graph Style Sheets” included with your copy of RATS.
3.6 Labeling Graphs

Introduction

To help you produce publication-quality graphs, RATS gives you a great deal of flexibility in labeling graphs. We look at these in the paragraphs below.

Major Labels

GRAPH, SCATTER, GCONTOUR, and SPGRAPH offer HEADER, FOOTER, SUBHEADER, HLABEL, and VLABEL options. Their functions are shown in the graph below.

Note that when these are used on SPGRAPH, they are placed at these locations relative to the full graph page, and not to any single graph on it.

Other Labels

When you are using SPGRAPH to put multiple graphs on a page, you can use the XLABELS and YLABELS options to label the rows and columns on the page.

The GRTEXT instruction adds a string of text inside a graph. It allows you to select the location, font, size, and style of the text string.

When doing a SCATTER plot or GCONTOUR graph, you can also use the XLABELS option to supply your own list of labels for the X-axis tick marks.
Supplying Labels

You can supply either a literal string of text enclosed in quotes, or a LABEL or STRING variable defined ahead of time (some options require a VECTOR of STRINGS). For example, the following GRAPH commands produce the same result:

```ratsscript
graph(header="US Real GDP")
# rgdp

compute hlab = "US Real GDP"
graph(header=hlab)
# rgdp
```

Constructing Strings

Quoted strings are fine for single graphs, but if you need to generate many standardized graphs, it can be rather tedious to duplicate code and hand edit it. Fortunately, all these options will permit you to use a STRING type variable instead of a quoted string. These can be constructed, or input (see below).

```ratsscript
compute header="Coherence of %l(x1) and %l(x2)"
graph(header=header,noticks)
# cohere

compute header="Autocorrelations +diffs+ x +sdiffs"
graph(header=header,max=1.0,min=-1.0,style=bargraph)
# corrs
```

Note: %L(series) is a special function which returns the label of a series.

Reading Strings From a File

If you have a standardized program, but the headers or labels can't be constructed easily as they have no obvious pattern, make up a separate file with a list of headers (one per line) and read them in using the instruction READ:

```ratsscript
open data labels.rgf
declare vector[string] header(230)
read header
do i=1,230
  ...
graph(header=header(i),...)
  # ...
end do i
```

Line Breaks in Strings

You can use two backward slash characters (\\) to insert line breaks in strings used in graphs. See the GRTEXT examples in Section 3.9.
Using Fonts

The **FONT** option on **GRPARM** and **GRTEXT** allows you to choose the font for the specified label(s). With the exceptions noted below, the fonts you select will be used when displaying, printing, and saving the graph. You can select from any font installed on your system (use the “Fonts” folder on the Windows Control Panel or Macintosh System folder, or a font-handling utility, to see a list of the installed fonts.)

You must type the font name exactly (although case does not matter). On a Windows system for example, you might use `FONT="Times Roman"` or `FONT="Arial"`.

If you export a graph to PostScript from Windows, RATS will automatically insert dashes between words in multi-word font names, as PostScript does not accept spaces in font names. It will also substitute standard PostScript fonts for their Windows counterparts (Helvetica for Arial, for example). If you want to change or add font substitutions, and are familiar with PostScript, you can edit the **PROLOG.PST** file (found in your RATS directory).

You can also supply exact PostScript names. RATS may not be able to use those fonts when displaying the graph, but will use them if you export the graph to a PostScript file. Be sure to use the full PostScript name. For example, the regular version of Adobe Garamond would be “AGaramond–Regular” rather than “AGaramond”.

On the Macintosh, font names are generally the same for both display and PostScript devices. This means you can use the same font name for displaying output to the screen, printing on a PostScript printer, or exporting to a PostScript file. UNIX systems are generally similar to Macintosh systems—use exact PostScript names for both display and printing purposes, as well as exporting to a file.

**Examples**

```
grparm(font="Symbol") axislabels 14

compute [vect[strings]] flab=$
     ||"0","p/6","p/3","p/2","2p/3","5p/6","p"||

scatter(style=line,vlabel="Coherence",vmin=0.0,vmax=1.0,$
    xlabels=flab) 1
# freq coher
```

The axis labels will be done in 14 point “Symbol” font. Although this choice affects both the horizontal and vertical axes, the numbers on the vertical axis will still be shown as numbers, since the Symbol font doesn’t remap the numerical characters. However, the p’s in the **XLABELS** option strings will be shown as π’s. The seven character strings in **FLAB** will be spaced equally (on center) from one end of the x-axis to the other.
3.7 Keys (Legends)

The KEY option on GRAPH and SCATTER allows you to add a key (legend) at a choice of several locations. Four of these are outside the graph box (KEY=BETWEEN, ABOVE, LEFT or RIGHT). Four of these are inside the graph box (UPLEFT, LOLEFT, UPRIGHT, LORIGHT). This second group doesn’t take space away from the graph itself, but you have to be careful to choose a location (if it exists) which doesn’t interfere with the graph. There is also KEY=ATTACHED, which puts the graph labels on the graph itself at locations where the labeling will be as clear as possible. This is available only for a few types of graphs (line, step, and symbols time series graphs).

By default, RATS uses the series names for the key labels. If these are cryptic database names, you can provide your own labels on the graph using the KLABELS option. KBOX controls whether a box is drawn around the key, while KHEIGHT and KWIDTH allow you to control the size and shape of the key. If you only want to see text in the key box (perhaps you don’t need to show a sample line because you are graphing only one series), use NOKSAMPLE.

This example labels the two series in the key with descriptive titles rather than the original coded series names. The key will be placed below the graph.

```
graph(key=below,klabels=||"30 Year Bonds","3 Month Bills"||) 2
# ftb30y 1998:1 1999:12
# ftbs3 1998:1 1999:12
```

This uses KEY=ATTACHED, and also KLABELS.

```
graph(style=line,key=attached,header="Figure 3.15",$ klabels=||"AGRICULTURE","RETAIL","SERVICES","MANUFACTURING"||) 4
# ga8gff
# ga8gr
# ga8gs
# ga8gm
```

![Figure 3.15](image)
3.8 Overlay (Two-Scale or Two-Style) Graphs

RATS makes it easy to plot series with two different scales on a single time series graph, or pairs of series with different Y-scales on a single scatter plot. The scale for the first set of series appears on the left side of the graph, while the scale for the second set of series appears on the right side of the graph.

Both \texttt{GRAPH} and \texttt{SCATTER} have an \texttt{OVSAMESCALE} option. If you use this (along with the \texttt{OVERLAY} option), the graph will be created with the same scale on both axes, but with different styles for different series.

Creating Overlay Graphs

Here is the basic procedure for creating an overlay graph:

- Use the \texttt{OVERLAY} option on your \texttt{GRAPH} or \texttt{SCATTER} instruction to tell RATS to do an overlay graph. \texttt{OVERLAY} offers the same choices as \texttt{STYLE}, but it sets the style used for the overlaying (right-scale) series. The \texttt{STYLE} option itself applies to the left-scale series.
- If you are only graphing two series (or two pairs of series for \texttt{SCATTER}), you don’t need to do anything else. Just list the series on separate supplementary cards, as you would for an ordinary graph.
- Use the \texttt{OVCOUNT} option if you want to graph more than one series using the right-side scale. For \texttt{OVCOUNT=n}, RATS graphs the last \textit{n} series (pairs) listed using the right-side scale.

Notes

When you use the \texttt{OVERLAY} option, RATS ignores the \texttt{AXIS}, \texttt{EXTEND}, and \texttt{SCALE} options on \texttt{GRAPH} and the \texttt{AXIS}, \texttt{EXTEND} and \texttt{VSCALE} options on \texttt{SCATTER}. This is because running an axis line or extending grid lines across the graph is likely to be very distracting, as they will apply to only one of the two scales. All other options, such as \texttt{HEADER}, \texttt{HLABEL}, and \texttt{GRID}, work normally.

You can use the \texttt{OVLABEL} option to label the right-side scale. Use the standard \texttt{VLABEL} option to label the left-side scale.

The \texttt{OMAX} and \texttt{OMIN} options set the maximum and minimum values for the right-side scale. They function just like the \texttt{MAX} and \texttt{MIN} options, which will control only the left-side scale in a two-scale graph.

If you want to use one of the “painted” styles, such as \texttt{POLYGONAL} or \texttt{BARGRAPH}, you should probably use that as the \texttt{STYLE} (left-scale). If you use a paint style for the overlaying series, they may cover up the first block of series.

You can use the \texttt{OVRANGE} option to offset the two scales so they aren’t directly across from each other. \texttt{OVRANGE=value between .5 and 1} gives each scale the given fraction of the vertical size. \texttt{OVRANGE=.6} for instance will have them overlap only slightly (the left scale getting the bottom 60% and the right getting the top 60%).
Chapter 3: Graphics

Example

This is a simple example graphing two series. The series HSF (housing starts) is graphed using the POLYGONAL style, and its scale appears on the left side of the graph. The series FCME (mortgage rates) is graphed as a LINE, and uses the right-side scale.

You will probably need to experiment with the symbol choice fields to get overlay graphs to look good. Here, we have used fill pattern 2 (medium gray) for the HST series, and line type 1 (solid line) for the FCME series. Without these, HST would be graphed using a solid black pattern (pattern 1), and FCME, which would be a dashed line, would not be as visible.

We also used the KLABEL option to supply custom key labels.

```
open data haversample.rat
calendar(m) 1972
data(format=rats) 1972:1 1988:12 fcme hst
graph(key=below,header="Housing Starts vs Mortgage Rates",$
  klabel=||"Housing Starts","Mortgage Rate"||,$
  style=polygonal,min=0.0,overlay=line) 2
# hst / 2
# fcme / 1
```
3.9 Multiple Graphs on a Page

You can put two or more individual graphs on a single picture using \texttt{SPGRAPH} with the \texttt{HFIELDS} (horizontal fields) and \texttt{VFIELDS} (vertical fields) options. For example:

\begin{verbatim}
spgraph(vfields=2,hfields=2)
  four graph, scatter, or gcontour instructions
spgraph(done)
\end{verbatim}

This \texttt{SPGRAPH} divides the available space in half in each direction, creating four zones of equal size. The graphs (by default) fill the zones by rows beginning at the top left. Note, however, that the characters on the small graphs are sized to be proportional to what they would be on a full page; that is, each quarter graph will look like a reduced version of the same graph done by itself. The labeling (particularly on the scales) can sometimes get too small to be readable, particularly if you get to four or more graphs in either direction. Depending upon the situation, you may want to increase the character sizes using \texttt{GRPARM}, or you might want to suppress the axis labeling altogether. To do this, use \texttt{NOTICKS} and \texttt{SCALE=None} on \texttt{GRAPH} and \texttt{VSCALE=None} and \texttt{HSCALE=None} on \texttt{SCATTER} or \texttt{GCONTOUR}.

You also need to be careful that you do not leave false impressions by allowing each graph to find its own range. This could inflate the appearance of what may be, in reality, some very small effects. Most of the work in doing a matrix of graphs is, in fact, just preparation.

The following example graphs the exchange rate of the U.S. dollar versus four foreign currencies. These are first given a common reference by converting them to percentage appreciation of dollars per foreign unit since the start of the sample. As all but the UK series (\texttt{GBRUSXSR}) are originally stated in foreign currency per dollar, they have to be flipped. We then use \texttt{TABLE} to get the maximum and minimum values attained across \textit{all} the countries, and use these as the \texttt{MAX} and \texttt{MIN} on each \texttt{GRAPH}. This ensures that the very modest movements relative to the Canadian dollar aren’t exaggerated relative to the much larger movement versus the yen.

The \texttt{LABELS} instruction is used to give a more readable label to the series, naming them by their country instead of the database coding. Their new labels are used in the \texttt{GRAPH} instructions within the loop on the \texttt{HEADER} option.

\begin{verbatim}
cal(m) 1996
open data oecdsample.rat
data(format=rats) 1996:1 1998:12 canusxsr frausxsr $
  jpnusxsr gbrusxsr
Flip to dollars per foreign unit and assign descriptive labels

set canusxsr = 1.0/canusxsr
set frausxsr = 1.0/frausxsr
set jpnusxsr = 1.0/jpnusxsr
labels canusxsr frausxsr jpnusxsr gbrusxsr
# "Canada" "France" "Japan" "UK"
\end{verbatim}
Chapter 3: Graphics

Convert to percent appreciation

dofor i = canusxsr frausxsr jpnusxsr gbrusxsr
    compute base=([series]i)(1996:1)
    set i = 100*(i{0}/base - 1.0)
end dofor i

Use TABLE to get the maximum and minimum across all series

table(noprint) / canusxsr frausxsr jpnusxsr gbrusxsr

Set up the SPGRAPH for a 2x2 matrix

spgraph(vfields=2,hfields=2,$
    header="U.S. Dollar vs Major Currencies",$
    subheader="Percent appreciation over the sample")

dofor i = canusxsr frausxsr jpnusxsr gbrusxsr
    graph(max=%maximum,min=%minimum,header=%l(i)) 1
    # i
end dofor

spgraph(done)
Here is another example using **SPGRAPH**. This is taken from an example in Diebold (2004). The example program file (**DIEBP235.PRG**) is included with RATS.

```plaintext
spgraph(hfields=2,vfields=2,$
header="Figure 9.17 Recursive Analysis: Breaking Parameter Model")
scatter # x y3
rls(cohist=cohist,sehist=sehist,sighist=sighist,csum=csum) $
   y3 / resids
   # x
set upperco = cohist(1)+sehist(1)*2.0
set lowerco = cohist(1)-sehist(1)*2.0
graph(header="Recursive Estimates") 3
   # cohist(1)
   # upperco 10 *
   # lowerco 10 *
set upperres = 2.0*sighist
set lowerres = -2.0*sighist
graph(header="Recursive Residuals") 3
   # resids
   # upperres
   # lowerres
set cusum = cusum/sqrt(%seesq)
set upper5 startr end = .948*sqrt(%ndf)*(1+2.0*(t-startr)/%ndf)
set lower5 startr end = -upper5
graph(header="CUSUM test") 3
   # cusum
   # upper5
   # lower5
spgraph(done)
```

**Figure 9.17 Recursive Analysis: Breaking Parameter Model**
3.10 GRTEXT—Adding Text to Graphs

Using GRTEXT

The **GRTEXT** command allows you to add text anywhere inside a graph image, with options allowing you to specify the position of the text, as well as the alignment and font style. To use **GRTEXT**, you need to enclose the graphing and **GRTEXT** command(s) inside an **SPGRAPH** block.

The code excerpt below is from the **HISTOGRM.SRC** procedure included with RATS, which generates histogram plots. We use **GRTEXT** to include computed statistics on the graph. Note the use of “\" symbols to put line breaks into a single string—this is often easier than using multiple **GRTEXT** commands to achieve the same effect.

```rats
spgraph
display(store=s) "Mean" %mean \"Std Error\" sqrt(%variance) $ 
"\Skewness" %skewness \"Exc Kurtosis\" %kurtosis
if distrib==2 {
    set nx = 1.0/sqrt(%variance)*$
        %density((fx-%mean)/sqrt(%variance))
    scatter(style=bargraph,overlay=line,ovsamescale) 2
    # fx dx
    # fx nx
} else {
    scatter(style=bargraph) 1
    # fx dx
}
grtext(position=upright) s
spgraph(done)
```

![Graph with computed statistics](image)
3.11 Graphing Functions

If you want to graph a function \( y = f(x) \), use the instruction `SCATTER`. Create a grid series with \( x \) values and a corresponding series of \( y \) values. `SCATTER` with the option `STYLE=LINE` will create the graph. You can also use other style choices, such as `BAR` and `POLYGONAL`. In general, you should not need more than about 100 grid points on the \( x \)-axis for a smooth function. Note that the `LINE`, `BAR` and `POLYGONAL` styles on `SCATTER` only work correctly if the \( x \) series is sorted into increasing order.

The following computes a series of AR1 regressions for a grid of values for \( \rho \) between .5 and 1.0 and graphs the residual sum of squares against \( \rho \):

```plaintext
set rhos 1 101 = .5+.005*(t-1)
set rss   1 101 = 0.0
do i=1,101
   ar1(noprint,rho=rhos(i)) invest 1950:1 1985:4
   # constant ydiff{1} gnp rate{4}
   compute rss(i)=%rss
end do i
scatter(style=line,vlabel="Residual Sum of Squares",$
      hlabel="Value of rho",header="Multiple Mode in AR1")
# rhos rss
```

![Graph of residual sum of squares against \( \rho \) for AR1 regressions.](image)
3.12 Contour Graphs

The GCONTOUR instruction generates contour plots. Its design is quite different from GRAPH and SCATTER, because it needs the x and y values to be in the form of a grid, and the function values must be a matrix with dimensions size of x grid by size of y grid. This matrix is usually created using the instruction EWISE, which isn’t discussed until Chapter 4.

The following is taken from the file GCONTOUR.PRG. This analyzes the log likelihood of a model with a break point in the mean as a function of the break point (shown on the y axis) against the mean of the post break process (x axis).

Set up the grids for breaks (1,...,100) and mus (.2,.4,...,20)

```
compute breaks=%seqa(1.0,1.0,100)
compute mus   =%seqa( .2, .2,100)
```

Generate the log likelihood for all the combinations of mus and breaks.

```
dec rect f(100,100)
edwise f(i,j)=-50.0*log((x2-2*mus(i)*over(fix(breaks(j)))+(100-
breaks(j))*mus(i)**2)/100.0)
```

Do the contour graph with a grid line across the actual break (50)

```
gcontour(x=mus,y=breaks,f=f,vgrid=||50||)
```

![Contour Graph Example](image)
3.13 Miscellaneous Topics

Autocorrelations

Autocorrelations and partial autocorrelations are bounded between -1.0 and 1.0. Graphing them without fixing the plot limits can produce confusion, because the plot ranges for different sets of correlations will probably be quite different. We have found the following set of options to be very helpful:

```
graph(number=0, style=bargraph, max=1.0, min=-1.0) 1
# corrs
```

NUMBER=0 causes the time axis to be labeled with 0,1,2,etc. STYLE=POLYGONAL also works well for correlations (see Section 9.4 for more details on computing and graphing correlations).

This shows a very basic graph of correlations. If you use any RATS procedures which compute and graph autocorrelations (such as the BJIDENT.SRC procedure included with RATS), you are likely to run into some graphs which supply additional information. For instance, it’s possible to highlight the correlations which are statistically significant using the SHADING option, which we demonstrate in another context on page 131.
High-Low-Close Graphs

You can use the option `STYLE=VERTICAL` for high-low-close graphs and the like. At each entry, it draws a line connecting the highest and lowest values and puts hash marks at the locations of the series. If there are three or more series, it puts a filled circle on the first series so it can be picked out easily.

The following pulls monthly values of the Dow Jones Industrial Average into annual series. It uses three `DATA` instructions, one to get the maximum, one the minimum and one the final value during the year. Since each `DATA` instruction will reset `SPDJI`, the data are copied to a new series name after each. The “close” series should be listed on the first supplementary card so it gets tagged with a special symbol. For this graph, we look at results from 1982 through 2006.

```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
graph(style=vertical,header="DJIA: 1982-2006") 3
# djlast
# djmin
# djmax
```

![Graph of DJIA: 1982-2006](image.png)
Highlighting Entries

The options GRID and SHADING permit you to draw attention to particular entries. GRID does this by drawing a vertical line from top to bottom of the graph at specific entries. SHADING paints a shaded box over any set of consecutive non-zero entries.

Shading or grid lines are done before any of the series. If the graph uses one of the painted styles (POLYGONAL, BAR, STACKED or OVERLAP), it may cover the highlighting, so be careful if you want to use those.

SCATTER and GCONTOUR have separate options for grids or shading in each direction. Note that, unlike GRAPH, where the highlighting information is provided in data series, for SCATTER, the grid lines are provided using a VECTOR in the HGRID and VGRID options and the shading zones are in a Nx2 RECTANGULAR for HSHADE and VSHADE.

```
     t>=1978:3.and.t<=1981:11
graph(shading=raterise,header="U.S. Private Housing Starts","$
     subheader="With Rising Mortgage Rate Intervals") 1
# hsf 1972:1 1987:12
```

Forecasts

If you graph forecasts with the last few observations of historical data (as separate series), you will find there will be a break in the graph between the end of the historical data and the start of the forecasts. One way to improve the appearance is to add the final historical value to the beginning of the forecast series. That way, its line will connect smoothly with the historical series.

The following uses `ESMOOTH` (exponential smoothing) to forecast from 2007:1 to 2008:12. The final actual data value from 2006:12 is added at that entry to the forecast series (`JPNFORE`). A grid line is added to the graph at 2006:12.

```
cal(m) 1960:1
open data oecdsample.rat
data(format=rats) 1960:1 2006:12 jpniptottrs

esmooth(trend=select,seasonal=select,\
    forecast=jpnfore,steps=24) jpniptottrs
set jpnfore 2006:12 2006:12 = jpniptottrs
graph(header="Forecasts of Japanese IP",grid=t==2006:12) 2
# jpniptottrs 2004:1 2006:12
# jpnfore 2006:12 2008:12
```

![Forecasts of Japanese IP](image)
This is another example of graphing forecasts, taken from pages 349-360 of Diebold (2004). It includes upper and lower confidence bands, and shades the “forecast” area. The full example program (DIEBP348.PRG) and associated procedure files are included with RATS.

Set dummy variable FOREZONE to 1 for 1995:1 and later:

```
set forezone * 2010:12 = t>=1995:1
```

Estimate the model through 1994:12

```
boxjenk(const,diffs=1,ar=1,define=diffeq) logyen * 1994:12 resids
```

Use UFORECAST to generate forecasts and standard errors:

```
uforecast(equation=diffeq,stderrs=sefore) fyen 1995:1 1996:7
```

Generate upper and lower bound series from results:

```
set upper 1995:1 1996:7 = fyen+2.0*sefore
set lower 1995:1 1996:7 = fyen-2.0*sefore
```

Graph the actual values, forecasts, and upper and lower bounds. Shading is applied where FOREZONE is non-zero. The symbol choice parameter is used to specify line style 3 for both the upper and lower series:

```
graph(header=$
"Figure 12.17. Log Yen/Dollar Rate: History, Forecast and Realization",$
shading=forezone) 4
# logyen 1990:1 1996:12
# fyen 1995:1 1996:7
# upper 1995:1 1996:7 3
# lower 1995:1 1996:7 3
```

![Figure 12.17. Log Yen/Dollar Rate: History, Forecast and Realization](image)
Chapter 3: Graphics

Fan Charts

Fan charts provide a useful way to display confidence bands, particularly with multiple levels. The confidence bands are displayed in shades of the same color, which are darkest nearest the center and growth fainter towards the outer bands. With the `GRAPH` instruction, it’s usually best to do these as an overlay, with the point forecast being done as a line. Make sure that you use `OVSAME` with `OVERLAY=FAN` so the bands are located in the correct location.

The example below is from Makradakis, et. al (1998), part of textbook example file `MWHP366.PRG`. It computes point forecasts and upper and lower 80% and 95% confidence bands. The pre-forecast data and the point forecasts are done as line graphs; the confidence bands as an overlay. The order of listing of the series covered by the fan isn’t important. At each point, the values are ordered from bottom to top. With four series, there are three bands. The center is done in a darker gray (for black and white) and the outer ones in a lighter shade.

```
uforecast(equation=weq,stderrs=stderrs) wfore 1973:1 1974:12
set lower95 1973:1 1974:12 = wfore+%invnormal(.025)*stderrs
set upper95 1973:1 1974:12 = wfore+%invnormal(.975)*stderrs
set lower80 1973:1 1974:12 = wfore+%invnormal(.1)*stderrs
set upper80 1973:1 1974:12 = wfore+%invnormal(.9)*stderrs

* graph(footer="Figure 7-26 Forecasts and Prediction Intervals",$ovcount=4,overlay=fan,ovsame) 6
# writing 1969:1 1972:12
# wfore
# lower95
# lower80
# upper80
# upper95
```

Figure 7-26 Forecasts and Prediction Intervals
Chapter 4
Scalars, Matrices and Functions

This chapter describes the many data types which RATS supports and the functions and operators which can be applied to them. In addition to simple integer and real scalars, RATS supports complex numbers, strings, and arrays of various configurations. In addition, certain objects which are important in the program, such as equations, sets of equations (MODELS) and sets of parameters for non-linear estimation (PARMSETs) can be defined and manipulated. Scalar and matrix calculations allow you to extend the capabilities of RATS far beyond what is provided by the standard statistical instructions.
4.1 Working with Scalars

Overview

In Chapter 1, we looked at RATS primarily as a program to input, display, transform and analyze entire data series. This, however, is only part of what RATS can do. Much of the power and flexibility of RATS comes from its ability to work not just with entire data series, but with real- and integer-valued scalars and arrays, as well as other types of information, such as string and label variables.

We’ll discuss arrays and matrix manipulations in Section 4.6. First, we’ll look at scalar computations. These allow you to compute a wide variety of test statistics, set up loops and conditional instructions, and do difficult transformations which just can’t be done with SET.

The Instruction COMPUTE

COMPUTE is the principal instruction for working with scalar and matrix variables. It takes the form:

```
compute variable name = expression
```

For example, to store the real value 4.5 in a variable called A, do

```
compute a = 4.5
```

COMPUTE supports the standard numerical operators (see Section 1.8). The following adds the values of four entries of the series IP, saves the result of the expression in the variable SUM, and then uses SET to create a benchmarked series (IP95). Note that you refer to an element of a series (or array) by putting the element reference in parentheses immediately following the variable name:

```
set ip95 = 400*ip/sum
```

You can evaluate multiple expressions with a single COMPUTE instruction by separating the variable=expression fields with commas (blank spaces before and after the commas are optional):

```
compute temp1=1.0, temp2=2.0, temp3=3.0
```

Data Types and COMPUTE

Every variable you use in RATS has a data type which determines what kind of information it can hold. The variables constructed with DATA and SET instructions are examples of the data type SERIES. Once the type of a variable has been set, it cannot be changed (except by deleting it from the symbol table, which should only be done very carefully).

RATS has a wide variety of data types, which we discuss in detail in Section 4.3. We are interested here principally in two of them: INTEGER and REAL.
INTEGER whole numbers, which are used primarily as counters, subscripts and entry numbers or dates. The \texttt{SET} index $T$ is an INTEGER.

REAL general real values, used for data, sums, test statistics, etc. The variables $A$ and $\text{SUM}$ in the examples above are type REAL.

When you first use a variable name in a \texttt{COMPUTE}, RATS needs to assign it a data type. You can do this in advance by using the instruction \texttt{DECLARE} to create the variable (see Section 4.3), but that is rarely necessary for scalars. Instead, you can often let RATS determine what type of variable you need based upon the context. In the example above, TEMP1, TEMP2 and TEMP3 will all be REAL variables because the expressions are real-valued (1.0 with a decimal point is real, 1 without it is integer).

You can also do an explicit type assignment in the \texttt{COMPUTE} by inserting the desired type in brackets \{\ldots\}.

\begin{center}
\textbf{Instruction:} \hspace{2cm} \textbf{Variable type:}
\end{center}

\begin{tabular}{ll}
\texttt{compute [real] tempvalue1 = 26} & \texttt{Real} \\
\texttt{compute tempvalue2 = 35.0} & \texttt{Real, because expression is real.} \\
\texttt{compute [integer] tempvalue3 = 99} & \texttt{Integer} \\
\texttt{compute tempvalue4 = 125} & \texttt{Integer, because expression is integer}
\end{tabular}

Note that RATS will convert an integer expression to a real value if the variable is type REAL. It will \textit{not}, however, convert decimal values to integers automatically. You have to use the \texttt{FIX} function for that.

If there is a conflict between the type of the variable and the result of the expression, RATS will give you an error message. For example:

\begin{center}
\texttt{compute [integer] a = 1000.5}
\end{center}

will produce the error:

\begin{center}
Expected type INTEGER, Got REAL instead
\end{center}

\section*{Date Expressions and Entry Numbers}

Dates, series entry numbers, and array elements are all treated as integers in RATS. In the example below, \texttt{START} and \texttt{END} are INTEGER variables.

\begin{verbatim}
cal(m) 1960
compute start=1965:1, end=2007:2
allocate end
\end{verbatim}

Note that \texttt{START} and \texttt{END} are set to the entry numbers associated with the dates 1965:1 and 2007:2 \textit{given the current CALENDAR setting} (entry numbers 61 and 566 in this example). If, for some reason, you change the \texttt{CALENDAR} later in the program, the integer values of \texttt{START} and \texttt{END} will \textit{not} change, so those entry numbers may no longer refer to the same dates.
An important advantage of storing dates as integer entry numbers is the ability to do arithmetic with dates. For example, suppose you have set the following `CALENDAR`:

```
calendar(m) 1990
```

This maps the date 1990:1 (January, 1990) to entry number 1. The following computations are all legal, and produce the results indicated:

```
compute start = 1990:1
compute end = start + 23
compute lags = 4
compute startreg = start+lags
```

This makes it easy to do rolling regressions and similar loops:

```
compute start = 1990:1, end = 2005:12
do i=0,11
   linreg y start+i end+i
   # x1 x2 x3
end
```

This does twelve separate regressions, over the ranges 1990:1 through 2005:12, 1990:2 through 2006:1, and so on.

**Functions**

RATS provides many built-in functions which can be used in `COMPUTE` expressions, `SET` instructions, and formulas for doing everything from simple matrix operations, to generating random draws, to specialized programming operations. See Section 1.1.10 for a general introduction to using functions. You can even define your own functions using the `FUNCTION` instruction (see Section 7.11 and Chapter 16).

We’ll provide a quick overview of the various categories of functions and mention a few of the most important examples here. We discuss the matrix functions in some detail later in this chapter, as they are so important in econometrics work, and most of the probability and random-draw functions are covered in Chapter 13. For a complete listing and description of all the functions available in RATS, please see Section 2 in the `Reference Manual`.

Here are just a few examples using functions in `COMPUTE` instructions:

```
com sqx = sqrt(x)  // Takes the square root of x
compute loga = log(a)  // Takes the natural log of a
compute random = %ran(1.0)  // Generates a random number from a Normal distribution with a standard deviation of 1.0.
```

The menu-driven `Functions Wizard` (on the Wizards menu) provides easy access to the built-in functions. You can see all the functions listed alphabetically, or grouped by category. Clicking on a specific function displays the syntax for the function along with a short description. Click on OK to open a dialog box allowing you to enter arguments for the function and then paste it into the input window. You can use the “Syntax Check” button to verify that your syntax is correct before clicking OK.
The various categories of functions available are listed below.

**Common Mathematical Functions**
These include such common functions as `LOG` for natural logs, `ABS` for absolute values, and `SORT` for taking square roots, as well as `%IF` for conditional expression evaluation, `%ROUND` for rounding numbers, and more.

**Trigonometric Functions**
The standard trig functions are available (`COS`, `SIN`, and `TAN`), as well as the hyperbolic (`%COSH` and `%SINH`) and inverse variations. Note that most function names in RATS begin with the “%” character, but of few of the most common do not—largely for backwards compatibility with RATS code written for older versions of the programs.

**Matrix Functions**
The two most common are `INV` and `TR` (inverse and transpose). This is, by far, the largest category: it includes functions for sorting (`%SORT`, `%SORTC`), extracting submatrices (`%XDIAG`, `%XSUBMAT`), factoring and decomposing (`%DECOMP`, `%SVDECOMP`) and specialized matrix multiplies (`%QFORM`, `%OUTERXX`).

**Distribution/Probability Functions**
This group includes probability distribution and density functions for a large number of standard distributions, such as Normal, Student’s t, chi-squared, F. There are inverse distributions functions for many of these as well. The Normal family, for instance, includes `%DENSITY` (standard Normal density), `%CDF` (standard Normal CDF), `%INVNORMAL` (inverse standard Normal) and `%ZTEST` (two-tailed standard Normal test).

**Random Number Generation Functions**
This includes `%RAN` (draws standard Normals), `%UNIFORM` (uniform random numbers), `%RANGE` and `%RANBETA` for random gammas and betas. It also includes “convenience functions” which do draws from particular types of posterior distributions. See chapter 13 for more information.

**String, Label, and Output Formatting Functions**
Functions like `%LEFT`, `%MID`, and `%RIGHT` extract substrings from larger string variables, while `%S` and `%L` are useful for automating repetitive operations involving series. There are also functions for string comparisons, converting numbers in a string to a value and vice versa.

**Date, Entry Number Functions**
These functions can be very useful in automating handling of dates and sample ranges, and performing transformations. For example, you can use `%YEAR`, `%MONTH`, and `%DAY` to get the year, month, or day, respectively, of a given entry number. The `%INDIV` and `%PERIOD` functions are useful for creating panel-data dummy variables. A more advanced function like `%CLOSESTWEEKDAY` can help you set up adjustments for floating holidays.
Chapter 4: Scalars, Matrices, and Functions

Financial Functions
RATS provides %ANNUITY and %PAYMENT functions for financial computations.

Complex Numbers and Matrices
This is an extensive collection of functions for complex-valued computations, such as complex conjugate (%CONJG), and functions for getting the real and imaginary parts of a complex number (%REAL and %IMAG). See Chapter 15 for details on spectral methods.

EQUATION/Regression List functions
These are functions for working with the EQUATION data type, regression lists and tables.

MODEL functions
These are functions for working with the MODEL data type.

Polynomial functions
Functions for working with polynomials. See Section 4.7.2 for more information.

Utility Functions
These are functions which don’t fit neatly into one of the previous groups. The most important of these is %IF, which was described in Section 1.7.

Displaying Values of Variables and Expressions
The instruction DISPLAY allows you to express results in a very flexible way: you can include formatting information and descriptive strings.

display "Breusch-Pagan Test" cdstat "P-Value" #.### signif

produces output like

Breusch-Pagan Test    3.45000 P-Value 0.063

Examples
linreg longrate
# constant shortrate{0 to 24}
compute akaike =-2.0*%logl+%nreg*2.0
compute schwarz=-2.0*%logl+%nreg*log(%nobs)
display "Akaike Crit. = " akaike "Schwarz Crit. = " schwarz

computes Akaike and Schwarz criteria for a distributed lag regression (Example 5.5). This uses “reserved” variables set by the LINREG instruction. You can use the RATS Variables Wizard, or see Appendix B in the Reference Manual, for a complete list of these. See Section 4.2 for more information on reserved variables.

DISPLAY can even do computations on the fly. Note that you must not use any blank spaces between parts of an expression, as otherwise DISPLAY will try to parse the separate pieces of the expression as separate items to be displayed.
display "Significance level" #.#### (sigcount+1)/(ndraws+1)
display "90% confidence interval" $
(sampmean+(testmean-%fract95)) (sampmean+(testmean-%fract05))
display "LR Test of Overidentification"
display "Chi-Square(" ### degrees ") =" * teststat $
"Signif. Level =" ###### %chisqr(teststat,degrees)

This last example will produce output like:

LR Test of Overidentification
Chi-Square(  5  ) =       8.56000 Signif. Level = 0.1279546

The ### strings are picture clauses, which show how you want a number to be displayed. A picture clause stays in effect (on a single DISPLAY instruction) until you use a new one. Each # character shows one character position, so #.### requests one digit left of the decimal and three digits right. *.### requests three digits right and as many digits left as are needed. * by itself requests a standard format which generally shows five digits right of the decimal.

The Assignment Operator (=)

The arithmetic and logical operators supported by RATS were introduced in Chapter 1, and are described in more detail in Appendix A in the Reference Manual. Here we will take a special look at the assignment operator.

In an expression, the = operator stores the value of the expression to its right into the variable immediately to its left (the target variable). By embedding = operators within an expression, you can accomplish several tasks at once. Some general points about the assignment operator:

- Unlike the other operators, in the absence of parentheses RATS does the rightmost = operation first.
- = has the lowest precedence of the operators.

Note: a single = symbol is the assignment operator. Use == (two equal signs) to test for equality.

compute y(1)=y(2)=y(3)=0.0
set zhat = (x2(t)=z2/y2) , x2 * v2/y2

X2, v2, Y2, and Z2 are series
4.2 Getting Information from RATS

Variables Defined by RATS Instructions

You can access directly most of the information displayed by the RATS statistical instructions. For example, LINREG computes variables such as \%RSS (residual sum of squares), \%NDF (number of degrees of freedom) and \%DURBIN (Durbin-Watson). You can display these values, save them in other variables, or use them in calculations.

All of the reserved variable names (except for the integer subscripts T, I, and J) begin with the \% symbol, to help distinguish them from user-defined variables. You cannot define your own variables using any of these reserved names.

You can use the RATS Variables operation on the Wizards menu to see a complete, categorized listing of the variables defined by RATS. You will also find an alphabetized listing of these variables in Appendix B in the Reference Manual.

Examples

This computes an \( F \)-statistic using \%RSS and \%SEESQ (standard error of estimate squared). The first COMPUTE instruction saves the \%RSS value in the variable RSSRESTR. The second COMPUTE takes RSSRESTR and the \%RSS and \%SEESQ from the second regression to compute the \( F \)-statistic. (Note that this is just an illustration—RATS has built-in instructions for doing tests like this. See Chapter 6 for details.)

```
linreg pcexp
# constant pcaid
compute rssrestr=%rss

linreg pcexp
# constant pcaid pcinc density
compute fstat=(rssrestr-%rss)/(2*%seesq)
```

Remember that each time you use a statistical instruction (LINREG, STATISTICS, etc.), the newly-computed values will replace the old ones. We needed to preserve the \%RSS from the first regression, so we had to save it somewhere before doing the second regression. If we didn’t think ahead, we would have had to type the value back into the FSTAT formula.

You could just punch the numbers from the regression output into a calculator. However, RATS does the computations at full precision, which may make a substantial difference if, say, RSSRESTR and the second \%RSS agree to four or five significant digits.

Note also that with RATS, you have a powerful calculator all set to go. For example:

```
display (1.34759-.33153)*46/(2*.33153)
```

displays the result of the calculation. This has the added feature of being 100% reproducible, unlike computations typed into an external calculator, which are often impossible to verify without going through the entire process again.
4.3 The RATS Data Types

Background
Much of the computational power of RATS comes from the wealth of data types which allow you to work with data in the most efficient or convenient form. For instance:

- RATS treats data series differently from simple arrays of numbers, by keeping track of defined entries and allowing varying entry ranges.
- RATS differentiates between integer and real computations. This allows programs which do extensive looping to run faster because integer calculations are much more efficient.
- RATS can perform matrix arithmetic, with the matrices treated as individual “variables.”

The Data Types
There are two categories of variable types in RATS: basic and aggregate.

- The basic data types store a single value or object, such as a single real value, a single integer value, or single equation.
- The aggregate data types store several data objects in a single variable. For example, a RECTANGULAR array of integers or a SERIES of real numbers.

These are the basic and aggregate data types available in RATS:

Basic Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>a single integer number</td>
</tr>
<tr>
<td>REAL</td>
<td>a single real number</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>a single complex number</td>
</tr>
<tr>
<td>LABEL</td>
<td>a single string of up to 16 characters</td>
</tr>
<tr>
<td>STRING</td>
<td>a single string of up to 255 characters</td>
</tr>
<tr>
<td>EQUATION</td>
<td>a linear equation</td>
</tr>
<tr>
<td>MODEL</td>
<td>a group of FRMLs or EQUATIONS</td>
</tr>
<tr>
<td>PARMSET</td>
<td>a description of free parameters for non-linear estimation</td>
</tr>
</tbody>
</table>

Note: The MODEL and PARMSET types are a bit odd in that they can contain multiple objects, but they otherwise have more in common with the basic types than with the aggregate types and so are listed here.

 Aggregate Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR</td>
<td>a one-dimensional array</td>
</tr>
<tr>
<td>RECTANGULAR</td>
<td>a general two-dimensional array</td>
</tr>
<tr>
<td>SYMMETRIC</td>
<td>a symmetric two-dimensional array</td>
</tr>
<tr>
<td>SERIES</td>
<td>a “time” series</td>
</tr>
<tr>
<td>FRML</td>
<td>a coded function of time</td>
</tr>
</tbody>
</table>
Chapter 4: Scalars, Matrices, and Functions

Denoting Data Types

Where you need to refer to a basic data type, you need to use three or more letters of the data type name from the previous page. For instance, you can use EQU to abbreviate EQUATION or COMP for COMPLEX.

To refer to an aggregate data type, again you can abbreviate to three or more letters. Use the construction aggregate type [data type]. Examples are VECTOR [INTEGER] (read VECTOR of INTEGER), RECT [COMPLEX] (RECTANGULAR of COMPLEX), VECTOR [SYMM [REAL]], (VECTOR of SYMMETRIC of REAL).

Creating Variables

You can create variables of any of the types using the DECLARE instruction, in the sense that you can define its type to RATS and have space set aside for it. A variable, however, is not of much use until you give it a value. When you set a variable, such as with a SET or COMPUTE instruction, RATS is usually able to tell what type is needed from the context. If so, you do not need an explicit DECLARE. If you need to use DECLARE, or you want to use it so that you can keep better track of the variables used in a program, use the following syntax:

```
declare variable type variable name(s)
```

As examples:

```
declare real a b c
```

declares three real scalar variables.

```
declare vector x(10)
```

declares a ten-element VECTOR of reals. (VECTOR is shorthand for VECTOR [REAL]).

```
declare rectang [series] impulses (nvar, nvar)
```

declares a RECTANGULAR array, where each element of the array is a SERIES.

Restrictions on Variable Names

Any variable name must satisfy the following restrictions:

- It must begin with a letter or %. % is used mainly for names defined by RATS itself. Our standard practice is to use a prefix of “%%” for variables defined in RATS procedures, rather than in any of the built-in instructions.
- It must consist only of letters, digits, $, %, or _.
- It can be up to sixteen characters in length. You can use more than sixteen characters, but only the first sixteen determine which variable you are using.
Instructions That Set Variables

These are the main instructions used to set and manipulate the data types (other than SERIES—many different RATS instructions can create and use series):

- **COMPUTE**: assigns values to scalar and array variables.
- **DECLARE**: declares one or more (global) variables. In many cases, you do not need to declare a variable before you use it. However, your programs may be easier to read and modify if you DECLARE your variables at the beginning of the program.
- **DIMENSION**: sets the dimensions of an array. You can also provide array dimensions when you DECLARE them.
- **EQUATION**: defines a linear equation (an EQUATION variable).
- **EWISE**: sets all the elements of an array as a function of the row and column subscripts. You must DECLARE and dimension an array before using it in an EWISE instruction.
- **FRML**: defines a formula using a function of entry numbers (a FRML variable). This can be a non-linear equation.
- **GROUP**: “groups” formulas (FRMLS) and/or equations (EQUATIONS) into a description of a model (a MODEL variable).
- **GSET**: sets elements of a SERIES of elements other than reals.
- **NONLIN**: creates or modifies a PARMSET for non-linear estimation.

Input/Output Instructions

- **DISPLAY**: displays various types of variables, strings and expressions.
- **REPORT**: creates a spreadsheet-style report using a sequence of instructions. This can be very helpful in (accurately) extracting results for publication.
- **WRITE**: displays the contents of arrays or scalars of integer, real, complex, label or string information.
- **INPUT, READ**: read numeric or character information into arrays or scalars.
- **ENTER**: reads information from supplementary cards into arrays or scalars of integer, real, complex, label or string information.
- **MEDIT**: displays arrays in “spreadsheet” windows for input or output.
- **QUERY**: gets scalar information from the user through a dialog box.
- **DBOX**: generates more sophisticated user-defined dialog boxes.

Because INPUT, READ, ENTER, MEDIT and QUERY can process many types of information, you must DECLARE any variables which you read using them.
4.3.1 Basic Data Types

**INTEGER** stores single whole numbers. These can range in value up to approximately (plus and minus) two billion. The **SET** and **FRML** subscript T and the **EWISE** subscripts I and J are **INTEGERS**. **DO** loop index variables also must be **INTEGERS**, and, dates are stored as integer entry numbers. **INTEGER** variables are usually set using **COMPUTE**:

```r
compute start=1947:1, end=1991:2
compute k=%nobs-10
```

**REAL** stores a single floating point number. Reals are stored as double-precision, and on PCs, Macintoshes and most workstations, a real value can be as small (in absolute value) as $10^{-308}$ and as large as $10^{308}$. Real variables are important if you go beyond the simple RATS commands to do your own programming. Virtually all calculations of test statistics, data values, sums, etc. are done with real-valued data. **REALS** are usually set with **COMPUTE**:

```r
compute fstat=(rssr-rssu)/(q*%seesq)
compute [real] c = 10
compute d = 11.5
```

**COMPLEX** stores two floating point values, one each for the real and imaginary parts of a complex number. They usually are set by **COMPUTE**, though the **COMPLEX** type is relatively unimportant itself. You will usually work with complex numbers through complex series. Most of the uses of complex-valued data is in frequency domain analysis. See Chapter 15.

A **COMPLEX** is displayed as a pair of real numbers separated by a comma and enclosed in parentheses. For instance `(1.00343, 3.53455)`

**LABEL** is a character variable up to 16 characters long, which is, of course, the length of RATS variable and series names. You can set a **LABEL** with a **COMPUTE** instruction. **INPUT**, **READ** and **ENTER** are also common.

**STRING** is a character variable which can have up to 255 characters. They are often used for graph headers or file names. You can set them with:

- **COMPUTE**
- **DISPLAY (STORE=STRING variable)**
- **INPUT, READ, ENTER, DBOX** or **QUERY**. You must **DECLARE** a variable to be type **STRING** before you can do this.
**EQUATION** is a description of a linear relationship. These are used in computing forecasts and impulse response functions. Equation coefficients can be estimated using instructions like **LINREG** and **SUR**. You can create **EQUATIONS** using the **EQUATION** instruction, or with the **DEFINE** options of instructions such as **LINREG**, **BOXJENK** and **AR1**. If you want to assign values to the equation coefficients, you can do that using the **ASSOCIATE** command. You can edit or display an existing equation using **MODIFY**.

An **EQUATION** can be defined and referenced by name, or by its sequence number.

**MODEL** is a collection of **FRMLS**, **EQUATIONS** and related information, used for systems simulation. The **GROUP** instruction defines a **MODEL** by specifying which formulas and equations to include in the model. **MODELS** can also be created as a description of a **VAR** with **SYSTEM** and its subcommands. **MODELS** can be combined using the “+” operator.

**PARMSET** is a specialized data type which consists of a list of parameters for nonlinear estimation and constraints on them. **PARMSETs** are constructed and maintained using the instruction **NONLIN**. **PARMSETs** can be combined using the standard “+” operator. See Section 7.6 for more on the creation and use of these.
### 4.3.2 The Aggregate Data Types

You can make an aggregate of any data type, subject to the following restrictions:

- The individual elements of an aggregate must be of the same data type. RATS does not allow you to define general structures containing a mix of types.
- You can make an aggregate of an aggregate: `VECTOR[SYMMETRIC[REAL]]` for instance. You cannot go beyond two layers of aggregation.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTOR</td>
<td>One-dimensional arrays. You refer to the elements of a VECTOR using the notation <code>name(subscript)</code>. Subscripts start at 1.</td>
</tr>
<tr>
<td>RECTANGULAR</td>
<td>Two-dimensional arrays. You refer to their entries with the notation <code>name(row, column)</code>. The subscripts start at 1. RECTANGULAR arrays are stored by columns, that is, in an $M \times N$ array, the first $M$ elements are the first column, the next $M$ are the second column, etc. (This matters only for a few specialized operations).</td>
</tr>
<tr>
<td>SYMMETRIC</td>
<td>Symmetric two-dimensional arrays. RATS stores only the elements located on and below the diagonal, which saves approximately half the space. RATS uses more efficient routines for computing inverses, determinants and eigenvalues than are available for RECTANGULAR arrays. You refer to their entries by <code>name(row, column)</code>. The subscripts start at 1. SYMMETRIC arrays are stored by the rows of the lower triangle. The subscripts of the elements are, in order, (1,1), (2,1), (2,2), (3,1),..... Because SYMMETRIC arrays are stored in this “packed” format, you may find the <code>%SYMMROW</code>, <code>%SYMCCOL</code>, and <code>%SYMMPOS</code> functions helpful if you need to extract a particular row, column, or element.</td>
</tr>
<tr>
<td>SERIES</td>
<td>SERIES are essentially one-dimensional arrays of objects, but they have a more complex structure than a simple array (VECTOR): their entries do not have to run from 1 to $N$ and they have “defined” and “undefined” ranges.</td>
</tr>
<tr>
<td>FRML</td>
<td>A FRML is a coded description of a (possibly) non-linear relationship. It is a function of the time subscript $T$. FRMLs are used for non-linear estimation and general model simulation. You can define FRML’s which evaluate to types other than REAL, but you have to declare their type before using a FRML to define them.</td>
</tr>
</tbody>
</table>

You will usually have to DECLARE arrays. In a matrix expression (Section 4.6), the specific type of result is usually ambiguous: VECTORS and SYMMETRICS, after all, are really special cases of two-dimensional arrays. We use the special types for efficiency.
4.3.3 Using Arrays

Real-valued arrays are useful directly in matrix computations (Section 4.6). Arrays of all types are useful for “bookkeeping” when we are examining a possibly varying number of items. The ability to construct arrays of SERIES can be particularly helpful. For example, several of the programs and procedures which work with Vector Autoregressions (such as IMPULSES.PRG) use VECTORS and RECTANGULARS of SERIES to organize the calculations for large numbers of series.

You can specify array dimensions either as part of the DECLARE instruction, or with a separate DIMENSION instruction. If you have an array of arrays, you must dimension the element arrays individually.

Arrays Within Lists

Whenever RATS expects a “list” of objects of some type, it will accept any array of that type as well. This is very useful for writing general procedures which can handle problems for varying sizes.

For instance, the instruction KFSET needs a list of SYMMETRIC arrays. Instead of

\[
\text{kfset \ xxx1 \ xxx2 \ xxx3}
\]

when you need three arrays and

\[
\text{kfset \ xxx1 \ xxx2 \ xxx3 \ xxx4}
\]

when you need four, you can use

\[
\text{compute \ nvar=4 (or 3)}
\]
\[
\text{declare \ vector[\text{symmetric}] \ xxx(nvar)}
\]
\[
\text{kfset \ xxx}
\]

The arrays will be `XXX(1),...,XXX(NVAR)`.

We use this extensively for supplementary cards within procedures. The entries on a supplementary card for LINREG, for instance, can be put in a VECTOR[INTEGER] to be recalled as a whole. RATS also provides a set of functions for manipulation such “regression lists” (see Section 2 of the Reference Manual).

For instance, if you have set up a VECTOR[INTEGER] called REGLIST, the instruction

\[
\text{linreg \ y / \ resids}
\]
\[
\text{# \ reglist}
\]

regresses \(Y\) on the regressors listed in REGLIST and

\[
\text{linreg \ resids}
\]
\[
\text{# \ reglist \ resids\{1 to lag\}}
\]

uses the REGLIST regressors plus lags of RESIDS.
4.4 Data Series (The SERIES Type)

Most of the RATS commands use data series. Because of the special nature of time series data, we have distinguished between a data series and a simple one-dimensional array of numbers.

When you use a new name, say in a **DATA** or **SET** instruction, you are creating a variable of type **SERIES**. Internally, a **SERIES** consists of some bookkeeping information, such as the defined length and a pointer to the data.

When you use a series, you are actually specifying not just the series itself, but also the range of data to use. This may not always be apparent, due to the use of default ranges and **SMPL** settings.

**Series Numbers vs. Names**

Every data series has a **sequence number** as well as a name. This is a remnant of early versions of RATS, but we have retained it because series numbers can still be useful in some cases. In general, though, you will probably find it easier to use arrays of series or the **%S** function, described on the next page, for handling these kinds of repetitive tasks. Nonetheless, it is helpful to understand how series numbers work.

The first series created in a RATS session or program is series number 1, the second is number 2, etc. Suppose your program began with these three lines:

```plaintext
calendar(q) 1980
data(format=rats) 1980:1 2006:4 gdp wages m1 price
```

**GDP** would be series 1, **WAGES** series 2, **M1** series 3 and **PRICE** series 4.

Anywhere RATS expects a series name, you can also use an integer value, or an **INTEGER** variable. When RATS expects a set of series, you can supply a list of integers, a set of **INTEGER** variables, or even **VECTORS** of **INTEGERS**. For example:

```plaintext
print / 1 2
```

prints series 1 and 2 (**GDP** and **WAGES** using the above example). If you execute the instruction

```plaintext
print / gdp to price
```

RATS interprets “**GDP TO PRICE**” as all series from 1 (**GDP**) to 4 (**PRICE**). Notice that the series numbers are based purely upon the order in which the series were created.

If you want to check the number associated with a series name, just use **COMPUTE** to set an integer equal to the series number, or just use **DISPLAY** with the series name:

```plaintext
compute sernum = gdp
display gdp
```
Be Careful With Series Numbers!

Using series numbers in places where RATS always expects a series type variable is easy. For example, series on a `STATS` instruction expects a SERIES type so

```plaintext
do i=1,3
    statistics i
end do
```

produces statistics for series numbers 1, 2, and 3 as you would expect.

The same is true for any series parameter, such as on `SET` instructions. For example, if $X$ is series 1 and $Y$ is series 2, the following instructions are identical:

```plaintext
set y = 100 + .3*x
set 2 = 100 + .3*x
```

However, the same is not true on the right hand side of a `SET`, `FRML`, or `COMPUTE` instruction, where RATS accepts a variety of variable and expression types. So:

```plaintext
set 2 = 1
```

simply sets entries of series 2 equal to the constant value 1, not the values of series number 1! If you want RATS to treat an integer expression as a series number in such cases, you need to indicate this explicitly, using either lag notation (`SET` or `FRML`) or the type modifier `[SERIES]` (on `SET`, `FRML`, `COMPUTE`, `DISPLAY`, etc.). For example:

```plaintext
set y = 1{0}
set y = ([series]1)
```

To refer to a specific entry of a series referenced by number, use the type modifier in parentheses, followed immediately by the entry number in another set of parens:

```plaintext
compute test= ([series]1)(3)
```

Arrays of Series and the %S Function

RATS also allows you to set up arrays of series, which can be very helpful for automating tasks involving many series, or for simply keeping track of a large number of series, particularly when those series are related to each other in some manner. As a simple example, suppose you want to generate series containing the natural logs of a list of other series. You can do something like:

```plaintext
declare vector[series] loggdp(10)
compute count = 1
dofor s = usdgdp to ukgdp
    set loggdp(count) = log(s{0})
    compute count=count+1
end dofor
```
The %S function provides another handy way to refer to or create series. You supply a LABEL variable or expression as the argument for the function. If a series with that label exists, the function returns the number of that series. Otherwise, it creates a new series, using the supplied label as the series name. So the above example could also be handled using:

```r
dofor serlist = usgdp to ukgdp
  set %s("log"+%l(serlist)) = log(serlist{0})
end do
```

The %L function returns the label of series, so we use it to get the label of the current GDP series, and prefix that with the text “LOG” to create the name for our new series. For example, the first series will be LOGUSGDP. Because the SERLIST index variable only contains the number of the current series, we use the lag notation inside the LOG function so that we get the log of the associated series, not the log of that integer value.

See Section 4.5 for more on the %S and %L functions, and Section 16.3 for more on using these features for automating repetitive tasks.

More on Creating Series

SERIES variables are almost always set from context (or using %S as shown above), but there are other ways to create them:

- The *series* parameter on ALLOCATE creates a block of series numbered 1 to *series* (see Section 16.3 for examples). The first series created after the ALLOCATE is *series*+1.
- The SCRATCH and CLEAR instructions both create new series. SCRATCH creates a block of consecutively numbered series. CLEAR creates series from a list of names or numbers.
- DECLARE SERIES *name(s)* can also be used to create series. Note that unlike the other methods listed above, the declared series are not automatically defined over any range (they have length zero). You won’t be able to refer to a particular element of the series, or set an element using COMPUTE, until the series has been defined over the appropriate range, such as by setting it with a SET instruction.
Scalar Values from Series—Using SSTATS

Often, we need to distill a series down into one or more scalar statistics. While there are some specialized instructions for this (STATISTICS and EXTREMUM, for instance), the instruction generally used for this type of “query” is SSTATS. It has several advantages over the other instructions:

1. It can do calculations on several series in parallel.
2. It can do types of statistics (such as the product) that can’t be done using the standard instructions.
3. It can be applied directly to data as they are computed, while STATISTICS and EXTREMUM required you to create an actual series to analyze.

By default, SSTATS computes the sum of the values of an expression. You use the notation >> to store the results into a real-valued variable. So, for instance,

\[ \text{sstats nbeg nend log(series)} \rangle \text{logy} \]

will make LOGY equal to the sum of the logs of SERIES over the range from NBEG to NEND, and

\[ \text{sstats 6 36 resid} \text{sol} \times \text{2}} \rangle \text{olssqr resid} \text{exp} \times \text{2}} \rangle \text{expsqr } \]
\[ \text{resid} \text{par} \times \text{2}} \rangle \text{parsqr} \]

will make OLSSQR equal to the sum of squares of RESIDSOLS over the range 6 to 36. Similarly, EXPSQR is created from the squares of RESIDSEXP, and PARSQR from the squares of RESIDSPAR.

\[ \text{sstats(mean) 1 100000 (stats<target)} \rangle \text{pvalue} \]

STATS<TARGET is 1 if that is true and 0 otherwise, so the mean of this calculation will be the fraction of the 100,000 entries for which STATS<TARGET. This is put into the variable PVALUE.

\[ \text{sstats(min) 2 ntp extract-extract} \{1\} \rangle \text{minvalue} \]

computes the minimum value of the period to period changes of the series EXTRACT.
4.5 Strings and Labels

String and label variables can be used for graph headers, for labeling output or for picture codes. Anywhere that RATS expects an undelimited string (such as for the file name on an OPEN instruction or a picture code on DISPLAY), you can use a LABEL or STRING variable prefixed with the & symbol. For instance, OPEN DATA &FNAME will open the file whose name is stored in FNAME.

You can read strings or labels from an external file using the instruction READ. You can create string variables using the instruction DISPLAY with the STORE option. You can also create them with COMPUTE.

Operators

The + operator, when applied to two strings, concatenates them. If there is a space at the end of the first string or the beginning of the second, it will be retained in the combined string. If not, no blanks will be inserted and the two strings will run together. If you need spacing, you may need to add it, for instance, with s1+" "+s2. If there is more than one blank between the two, the extras will be removed, leaving just one blank as spacing.

You can also use the + operator to add (the character representation of) an integer to the end of a string. For instance "ABC"+I will produce the string ABC13 if I=13.

String Functions

%CONCAT(L1,L2) Concatenates two labels.

%LEFT(S,n) Returns the leftmost n characters from the string S.

%RIGHT(S,n) Returns the rightmost n characters from the string S.

%MID(S,m,n) Returns n characters from the string S starting at position m (with the first character position numbered 1). If n is zero or negative, %MID returns all characters from position m until the end of the string.

%VALUE(S) Returns the value of the (real) number whose character representation is in S.

%STRING(n) Returns a string with the integer n in character form.

%STRVAL(x,f) Returns a string with the real value x in the picture format f. For instance, %strval(11.9, "*.##") will return the string 11.90.

%STRCMP(S1,S2) Compares two strings. Returns 0 if they are identical, 1 if S1 > S2 in lexicographical order, and -1 if S1 < S2.

%STRCMPNC(S1,S2) Compares two strings without regard to case. Returns 0 if they are identical, 1 if S1 > S2, and -1 if S1 < S2.
%STRLEN(S)  Returns the length of a string to the last non-blank.
%STRLOWERS  Returns a copy of S converted to all lower-case characters
%STRUPPER(S)  Returns a copy of S converted to all upper-case characters
%STRREP(S,n)  Returns a string which repeats the substring S n times.
%DATELABEL(t)  Returns the date label (in the current CALENDAR) for entry t. For instance, if the calendar is quarterly, begin- ning 1953:1, %DATELABEL(6) is 1954:02.
%DATEANDTIME()  Returns the current date and time. This is useful for date stamping output. (Use the %TODAY function to get today’s date translated into an entry number in the current CALENDAR.)
%L(S)  Returns the label for series S
%LABEL(v)  Returns the label of the variable, array or series v.
%S(L)  Returns a reference for a series with label L. This can be a new name if it is used at a location where a new series could be created. For instance,

\[ \text{SET } %S(\text{"LGD}P\text{"}) = \text{LOG}(GDP) \]

would create a new series named LGDP. (The argument, will, in practice, be a more complicated function—see the previous section).

**Examples**

```
compute header="Transformations of "+%l(series)
```

creates in HEADER a string such as Transformations of FYGM3

```
disp(store=keylabel) %l(series)+"\" *.### value
```

creates in KEYLABEL a string such as GDP\7.391. (The \ is used in several situa-
tions to show a line break).

```
declare string fname
compute fname=%l(i)+".XLS"
open copy &fname
copy(format=xls,org=col,dates) / i
```

creates an Excel spreadsheet whose name is formed by adding “.XLS” to the name of series I, and which has the data for series I. Note that if you don’t put the & in front of FNAME, the file name is literally “FNAME”, not the string that FNAME holds.
Chapter 4: Scalars, Matrices, and Functions

4.6 Matrix Calculations

Overview
The following pages cover the instructions and some of the functions available for evaluating matrix expressions. Most of these apply only to real-valued arrays.

Accessible Arrays
Among the variables which RATS makes available to you are several arrays. The most important of these are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BETA</td>
<td>VECTOR</td>
<td>The vector of coefficients from the most recent regression.</td>
</tr>
<tr>
<td>%STDERRS</td>
<td>VECTOR</td>
<td>The vector of standard errors from the most recent regression.</td>
</tr>
<tr>
<td>%XX</td>
<td>SYMMETRIC</td>
<td>The estimated covariance matrix for most instructions, or the $(X'X)^{-1}$ matrix for single equation regressions without ROBUSTERRORS.</td>
</tr>
<tr>
<td>%CMOM</td>
<td>SYMMETRIC</td>
<td>The cross-moment matrix of regressors, defined by CMOMENT or MCOV.</td>
</tr>
</tbody>
</table>

There is a major difference between the scalar variables defined by RATS instructions, such as %RSS, and the arrays. You are permitted to change the internal values of these arrays. You cannot do this with the scalars. For instance, the command

```plaintext
COMPUTE %XX(2,1)=5.0
```

changes element (2,1) of the program’s own internal copy of the %XX matrix, which is used by the hypothesis testing instructions.

Matrix Instructions
You can create matrices in a variety of ways. You could set individual elements of an array with a whole set of COMPUTE instructions. However, RATS also offers several instructions (including COMPUTE) which can create and set matrices as a whole. Use these wherever possible, because they are much faster than brute force methods.

These are instructions which exist primarily to do matrix calculations. There are many others (such as LINREG) that create matrices as a side-effect of another calculation.

```plaintext
COMPUTE
```

can set entire matrices using algebraic formulas and special matrix functions. `COMPUTE XXMATRIX = TR(X)*X` is an example. Most of this section is a description of the use of COMPUTE for matrices.

```plaintext
EWISE
```

sets matrices using element-wise functions of the subscripts I and J. For instance, `EWISE SIGMA(I,J)=.9**(I-J)`

```plaintext
CMOM
```

computes a cross product or correlation matrix of a set of variables.
Chapter 4: Scalars, Matrices, and Functions

**MCOV** computes a long-run variance matrix for a set of variables.

**EIGEN** computes eigen decompositions of \( N \times N \) arrays.

**QZ** computes a generalized Schur decomposition of a pair of matrices.

**MAKE** creates an array from a collection of data series.

**FMATRIX** creates a “filter matrix”: a matrix with repeating patterns which shifts over one column per row.

**Literal Matrices**

You can create an array by listing its entries explicitly in an expression. For example:

\[
\text{compute } [\text{vect}] \ a = || 1.0, 2.0, 3.0 ||
\]

defines \( A \) as a 3-element VECTOR and

\[
\text{declare symmetric } s
\]

\[
\text{compute } s = || c_{11} | c_{21},c_{22} | c_{31},c_{32},c_{33} ||
\]

defines \( S \) as a 3x3 SYMMETRIC. The || ... || delimits the entire array, | separates rows of the array, and the commas separate elements within a row. You can use either constants or expressions for the elements.

**Variable Types**

You can mix arrays, series, and scalar variables in matrix expressions, subject to:

**Arrays** New arrays created in an expression will be of type RECTANGULAR by default, unless you either DECLARE the array prior to using it, or include an explicit type assignment before the matrix name. (See the examples above). You do not need to dimension arrays set in the course of the expression, except when you use the functions \%RAN, \%UNIFORM, \%RANGAMMA, \%RANCHISQR, \%RANBETA, \%CONST or \%MSCALAR.

**Series** COMPUTE treats data series as column vectors. It is a good idea to use a SMPL instruction to indicate the range to use. If you don’t, RATS uses the defined range of the series. You cannot set a series directly using matrix operations.

**Scalars** Several of the matrix functions return scalar values, and you can use \( A \times x \) or \( x \times A \) to multiply all elements of an array \( A \) by a scalar \( x \), \( A / x \) to divide all elements of \( A \) by \( x \), \( A + x \) to add \( x \) to every element and \( A - x \) to subtract \( x \) from every element.

If you have a calculation which returns a 1x1 matrix and you want to treat the result as a scalar, use the \%SCALAR function.
Operators

**COMPUTE** supports the following operators in matrix expressions. **A** and **B** can be array variables or expressions of the proper form.

- **A*B**
  Multiplication. **A** or **B** may be scalar (if both are, the result is scalar).

- **A+B, A-B**
  Addition and subtraction. If **A** and **B** are matrices, they must have compatible dimensions. **B** can be scalar; if it is, it is added to or subtracted from each element of **A**.

- **A/B**
  Division. **B** must be a scalar. **A** can be an array or scalar. If it is an array, each element of **A** is divided by **B**.

- **A^n or A**\(^n\)**
  Exponentiation. If **A** is an array, it just be square and \(n\) must be an integer (positive or negative). This does repeated multiplications of **A** (or its inverse) with itself.

- **A.*B, A./B**
  Elementwise multiplication and division, respectively. **A** and **B** must both be arrays of the same dimensions.

- **A.^x**
  Elementwise exponentiation, taking each element of **A** to the \(x\) power. \(x\) must be real.

- **A-B**
  Does a horizontal concatenation of two arrays (columns of **A** to the left of columns of **B**). **A** and **B** must have the same number of rows.

- **A~~B**
  Does vertical concatenation of two arrays (rows of **A** above rows of **B**). Both arrays must have the same number of columns.

- **A~\B**
  Diagonal concatenation of two arrays. The result is block diagonal with **A** in the top left and **B** in the bottom right. There are no restrictions on the dimensions.

- **A=B**
  Assignment. **A** must be an array, scalar variable or series or array element.

  - If **A** is an array, you do not need to dimension it (except when you use the functions %RAN, %UNIFORM, %RANGAMMA, %RANCHISQR, %RANBETA, %CONST or %MSCALAR.). The result **B** may not be scalar.
  
  - If **A** is scalar, you do not need to declare it; a new name will be created as type REAL. **B** may not be an array: use the %SCALAR function to turn an array **B** into a scalar.
Chapter 4: Scalars, Matrices, and Functions

Basic Functions

\textbf{TR}(A) \quad \text{Computes the transpose of } A. \\
\textbf{INV}(A) \quad \text{Computes the inverse of a SYMMETRIC or general square matrix. SYMMETRIC arrays should be positive definite.} \\
\%\text{DIAG}(A) \quad \text{Creates an } n \times n \text{ diagonal matrix from a vector. } A \text{ may be either a VECTOR, or any } n \times 1 \text{ or } 1 \times n \text{ array.} \\
\%\text{GINV}(A) \quad \text{Computes the generalized (Moore-Penrose) inverse of } A. \text{ This is a matrix } A^\dagger \text{ which solves } AA^\dagger A = A \text{ and } A^\dagger AA^\dagger = A^\dagger. \text{ } A \text{ does not have to be square; } A^\dagger \text{ will have the same dimensions as } A^\dagger. \\
\%\text{SOLVE}(A, b) \quad \text{Returns the solution } x \text{ of } Ax=b, \text{ where } A \text{ is } n \times n \text{ and } b \text{ is } n \times 1. \\

Reshaping Functions

\%\text{VEC}(A) \quad \text{Returns } A \text{ unraveled into a vector. A RECTANGULAR } A \text{ is stacked by columns, and a SYMMETRIC is stacked by the rows of the lower triangle.} \\
\%\text{VECTORECT}(V, r) \quad \text{Returns a RECTANGULAR array with } r \text{ rows, containing the values of vector } V. \text{ Elements 1 through } r \text{ of } V \text{ go into column one of the RECTANGULAR, elements } r+1 \text{ through } 2r \text{ go into column two, and so on. If the dimensions of } V \text{ are not a multiple of } r, \text{ values that would only partially fill the last column are omitted.} \\
\%\text{VECTOSYMM}(V, r) \quad \text{Similar to } \%\text{VECTORECT}, \text{ except that it returns a SYMMETRIC rather than a RECTANGULAR.} \\

Matrix Creation Functions

\%\text{ZEROS}(m, n) \quad \text{Creates an } m \times n \text{ matrix with zero in every position.} \\
\%\text{FILL}(m, n, x) \quad \text{Creates an } m \times n \text{ array with the value } x \text{ in every position.} \\
\%\text{IDENTITY}(n) \quad \text{Creates an } n \times n \text{ identity matrix.} \\
\%\text{SEQA}(\text{start}, \text{incr}, n) \quad \text{Creates a vector with a real-valued additive sequence.} \\
\%\text{UNITV}(n, i) \quad \text{Creates a vector with } n \text{ elements, with element } i \text{ set to one, and the other elements set to zero.} \\
\%\text{RANMAT}(m, n) \quad \text{Creates an } m \times n \text{ array filled with independent draws from the standard Normal.}
Chapter 4: Scalars, Matrices, and Functions

Elementwise Functions
These apply element by element to arrays, and also can be applied to scalar values.

- %LOG(A): Returns the elementwise natural logarithm.
- %EXP(A): Returns the elementwise exponential function.
- %ABS(A): Returns the elementwise absolute value.
- %SQRT(A): Returns the elementwise square root.

Scalar Functions of Arrays

- %AVG(A): Returns the average of the entries of A.
- %DET(A): Returns the determinant of a symmetric or square matrix.
- %DOT(A,B): Returns the dot product of two arrays A and B, which must have the same dimensions: $n \times 1$ and $1 \times n$ are considered equivalent, but an $n \times m$ array can be dotted only with another $n \times m$ matrix if both $n$ and $m$ are bigger than 1.
- %MAXVALUE(A): Returns the maximum value of the entries of A.
- %MINVALUE(A): Returns the minimum value of the entries of A.
- %QFORM(A,B): Returns $B'A B$ where A is $n \times n$ and B is $n \times 1$. It will also accept a $1 \times n$ B and transpose it.
- %SCALAR(A): Returns $A(1,1)$ (or $A(1)$ for a VECTOR).
- %SUM(A): Returns the sum of all elements of the array A. For a symmetric array, it computes the sum of the lower triangle only.
- %TRACE(A): Returns the trace of a symmetric or square matrix A.

Matrix Functions of Scalar Values
These must always be preceded by array =, as the functions get the dimensions from the array that you are setting.

- A=%CONST(x): Fills the array with the value x.
- A=%MSCALAR(x): Creates a “scalar” matrix—the identity matrix times the value x. A must be symmetric or a general square matrix.
- A=%RAN(x): Fills the array with draws from a Normal distribution with mean zero and standard deviation x.
- A=%UNIFORM(x1,x2): Fills the array with draws from a Uniform distribution on $(x1,x2)$. 

160 RATS User’s Guide
Chapter 4: Scalars, Matrices, and Functions

Specialized Products

\%INNERXX(A) \hspace{1cm} \text{Computes } A'A

\%KRONEKER(A,B) \hspace{1cm} \text{Creates the Kroneker product } A \otimes B

\%KRONID(A,B) \hspace{1cm} \text{Computes } (A \otimes I)B. A \text{ must be a square matrix; } B \text{ must have a first dimension which is a multiple of the dimension of } A.

\%KRONMULT(A,B,C) \hspace{1cm} \text{Computes } (A \otimes B)C.

\%MQFORM(A,B) \hspace{1cm} \text{Computes } B'AB. A \text{ must be } n \times n, B \text{ must be } n \times m. \text{ Use } \%QFORM \text{ if } B \text{ is } n \times 1.

\%MQFORMDIAG(A,B) \hspace{1cm} \text{Computes the diagonal only of } B'AB. A \text{ must be } n \times n, B \text{ } n \times m. \text{ Returns an } m \times 1 \text{ array.}

\%OUTERXX(A) \hspace{1cm} \text{Computes } AA'

Extraction and Related Functions

\%PSUBMAT(A,r,c,B) \hspace{1cm} \text{Copies information in matrix } B \text{ into matrix } A, \text{ starting at } (r,c) \text{ of } A. \text{ The copied values will overwrite any existing values in the affected entries of } A. \text{ Note that } A \text{ must be sufficiently large to allow all the entries in } B \text{ to be copied into the existing dimensions of } A.

\%PSUBVEC(VA,pos,VB) \hspace{1cm} \text{Puts the information in vector } VB \text{ into vector } VA, \text{ beginning at entry } pos \text{ of } VA. \text{ The copied values will overwrite any existing values in the affected entries of } VA. VA \text{ must be sufficiently large to support the operation.}

\%XCOL(A,n) \hspace{1cm} \text{Returns column } n \text{ of } A.

\%XDIAG(A) \hspace{1cm} \text{Returns the diagonal of an } n \times n \text{ matrix as an } n \times 1 \text{ array.}

\%XROW(A,n) \hspace{1cm} \text{Returns row } n \text{ of } A.

\%XSUBMAT(A,i,j,k,l) \hspace{1cm} \text{Returns the sub-matrix of } A \text{ from } (i,k) \text{ to } (j,l), \text{ that is, } i \text{ to } j \text{ is the row range and } k \text{ to } l \text{ is the column range.}

\%XSUBVEC(A,i,j) \hspace{1cm} \text{Returns the sub-vector } \{A(i),...,A(j)\}. 

RATS User's Guide 161
Decompositions/Factorizations

%DECOMP(A) Computes the Choleski decomposition of a SYMMETRIC matrix A. The result is a lower triangular (RECTANGULAR) array S which satisfies SS' = A.

%SVDECOMP(A) Computes a singular value decomposition of A. It returns a VECTOR [RECTANGULAR] with three components, the U, W and V matrices in A = UWV' where U is row orthonormal (that is U'U = I), V is column orthonormal and W is diagonal. The second element in the return is just the diagonal of W. Its values are non-negative and are sorted in decreasing order.

Sort/Rank Functions

%RANKS(A) Returns the ranks of the elements of A. The ranking is in increasing order, with 1.0 assigned to the smallest value. Ties are assigned the average of their ranks.

%SORT(A) Returns a sorted version of A. The sort is in increasing order. A RECTANGULAR array is filled by columns, so the first column will have the lowest values and the last will have the highest.

%SORTC(A, c) Returns a copy of array A sorted based upon the values in column c. This normally sorts in increasing order. Make c negative to sort on a column in decreasing order.

Miscellaneous

%COLS(A) Returns the number of columns in A. A can be an array of any type, not just real-valued.

%CVTOCORR(A, V) Returns the correlation matrix C created from a covariance matrix V: C_{ij} = V_{ii} / \sqrt{V_{ii} V_{jj}}

%ROWS(A) Returns the number of rows in A. A can be an array of any type, not just real-valued.
Examples

If \( A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \) and \( B = \begin{bmatrix} 1 & -2 \\ -4 & 8 \end{bmatrix} \)

compute \( \text{horiz}=a-b \)
compute \( \text{vert}=a\langle a \rangle - b \)
compute \( \text{diag}=a\langle a \rangle \backslash b \)
compute \( \text{emult}=a*.b \)
compute \( \text{add}=a+6.0 \)

return:

\[
\begin{align*}
\text{HORIZ} &= \begin{bmatrix} 1 & 2 & 1 & -2 \\ 3 & 4 & -4 & 8 \end{bmatrix}, \\
\text{VERT} &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 1 & -2 \\ -4 & 8 \end{bmatrix}, \\
\text{DIAG} &= \begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & -4 & 8 \end{bmatrix} \\
\text{EMULT} &= \begin{bmatrix} 1 & -4 \\ -12 & 32 \end{bmatrix}, \\
\text{ADD} &= \begin{bmatrix} 7 & 8 \\ 9 & 10 \end{bmatrix} \\
\end{align*}
\]

compute \( \logdet = \log(%\text{det} (\text{vcv})) \)

\( \logdet \) is equal to the log of the determinant of \( \text{vcv} \).

compute \( \text{convcrit} = \%\text{dot}(\text{sigmainv},s) \)

\( \text{convcrit} \) is the dot product of \( \text{sigmainv} \) with \( s \).

declare rect ddiag
compute \( \text{ddiag}=%\text{diag}([| 4.5, 5.6, 6.7 |]) \)

produces

\[
\begin{bmatrix}
4.5 & 0.0 & 0.0 \\
0.0 & 5.6 & 0.0 \\
0.0 & 0.0 & 6.7
\end{bmatrix}
\]

make \( x \)
# constant x1 x2 x3
make \( y \)
# y
compute \( \text{beta}=\text{inv}(\text{tr}(x)*x)*\text{tr}(x)*y \)

calculates the coefficient vector for a regression using matrix operations.
Chapter 4: Scalars, Matrices, and Functions

Optimization

Since matrix operations operate much faster than the equivalent set of operations done using loops and single-value COMPUTE's, you should use them in place of brute force computations wherever possible. This section suggests some ways to organize matrix manipulations in order to optimize the computations. In a program where there are extensive matrix manipulations, the savings can be significant.

Note, however, that the first goal is to get the computations correct. You can optimize later if execution is too slow.

Scalar Multiplication

Arrange scalar multiplication operations so the multiplication applies to the smallest array possible. If \( W \) is a 100x4 RECTANGULAR, \((1./\text{NOBS}) \times \text{TR}(W) \times W\) should be written as \((1.0/\text{NOBS}) \times (\text{TR}(W) \times W)\). In the former, the 400 entries of \( \text{TR}(W) \) are multiplied by \( 1./\text{NOBS} \), while in the latter, the scalar multiplication applies only to the 16 elements of \( \text{TR}(W) \times W \).

Addition and Subtraction

Addition, subtraction and elementwise multiplication operate most efficiently when the arrays are of similar form: either both SYMMETRIC or both RECTANGULAR or VECTOR with neither transposed.

Matrix Multiplications

Use the specialized matrix multiplication functions (\%QFORM, \%MQFORM, \%MQFORMDIAG, \%INNERXX and \%OUTERXX) wherever possible. \%MQFORMDIAG is particularly helpful, since it computes only the diagonal elements and thus reduces the order of complexity substantially.

If \( A \) and \( B \) are RECTANGULAR or VECTOR, \( \text{TR}(A) \times B \) is superior to the same computation done as a product \( C \times D \) or \( E \times \text{TR}(F) \). The operation \( \text{TR}(A) \times A \) can be done using \%INNERXX(\( A \)), while \( A \times \text{TR}(A) \) is \%OUTERXX(\( A \)). If \( A \) is SYMMETRIC and \( B \) is RECTANGULAR or VECTOR, \( A \times B \) is superior to the same done as \( A \times \text{TR}(C) \).

In a product of several arrays, try to have a SYMMETRIC array postmultiplied by a RECTANGULAR rather than premultiplied. For example, if \( C \) and \( D \) are RECTANGULAR and \( XX \) SYMMETRIC, \( \text{TR}(C) \times (XX \times D) \) is preferable to \( \text{TR}(C) \times XX \times D \).

Kroneker Products

You should avoid the \%KRONEKER function if at all possible: \%KRONID and \%KRONMULT are designed to take advantage of the structure of a Kroneker product in doing multiplications.
4.7 Special Families of Functions

4.7.1 The Sweep Functions

The %SWEEP function, and related %SWEEOPTOP and %SWEEOPLIST functions, are very useful, and probably don’t get as much use as they deserve.

\( \mathbf{B} = \text{%SWEEP}(\mathbf{A}, k) \) is the following:

\[
\begin{align*}
\mathbf{B}(k,k) &= 1/\mathbf{A}(k,k) \\
\mathbf{B}(i,k) &= -\mathbf{A}(i,k)/\mathbf{A}(k,k) \quad \text{for } i \neq k \\
\mathbf{B}(k,j) &= \mathbf{A}(k,j)/\mathbf{A}(k,k) \quad \text{for } j \neq k \\
\mathbf{B}(i,j) &= \mathbf{A}(i,j) - \mathbf{A}(i,k)\mathbf{A}(k,j)/\mathbf{A}(k,k) \quad \text{otherwise}
\end{align*}
\]

%SWEEP is reversible: \%SWEEP(%SWEEP(\mathbf{A}, k), k) = \mathbf{A}.

The point of %SWEEP is the following: if \( \mathbf{A} \) begins as a cross-product matrix of a set of variables, successive sweeps will do a regression. If we fix 1 as the dependent variable, after sweeping on 2 and 3, \( \mathbf{B}(2,1) \) and \( \mathbf{B}(3,1) \) will be the regression coefficients and \( \mathbf{B}(1,1) \) will be the residual sum of squares. Note that the sweep operator does not produce a symmetric matrix.

The “sweep” functions are:

\[
\begin{align*}
\text{%SWEEP}(\mathbf{A}, k) & \quad \text{Returns the result of “sweeping” } \mathbf{A} \text{ on pivot } k. \\
\text{%SWEEOPTOP}(\mathbf{A}, k) & \quad \text{Returns the result of “sweeping” } \mathbf{A} \text{ on each of the first } k \text{ rows.} \\
\text{%SWEEOPLIST}(\mathbf{A}, \text{iv}) & \quad \text{Returns the result of “sweeping” } \mathbf{A} \text{ on each of the pivots listed in the VECT[INTEGER] IV.}
\end{align*}
\]

Consider the properties of the sweep function when applied to a cross product matrix:

1. The submatrix defined by the rows and columns of the pivots is \( (\mathbf{X}'\mathbf{X})^{-1} \) from a regression on those variables.
2. The submatrix defined by the non-pivot columns with the pivot rows are the projection (regression) coefficients of the non-pivots on the pivots.
3. The submatrix defined by the pivot columns with the non-pivot rows are minus the projection coefficients.
4. The submatrix defined by the non-pivots by non-pivots gives the sums of the outer products of the projection residuals. (When divided by the number of observations, this will be the estimated covariance matrix of residuals).
If we take a regression of trend on constant,

```r
set trend 1 10 = t
cmom
# constant trend
disp %sweep(%cmom,1)
```

The output is:

\[
(X'X)^{-1} \text{ matrix } \begin{pmatrix} 0.10000 & 5.50000 \end{pmatrix} \quad \text{(regression coeff)}
\]

\[
\text{(-regression coeff)} \begin{pmatrix} -5.50000 & 82.50000 \end{pmatrix} \quad \text{(Residual Sum of Squares)}
\]

One of the exercises in Hayashi (2000) is to do a Monte Carlo analysis of unit root tests, doing one Dickey-Fuller test allowing for intercept and one with intercept and trend. The sweep functions offer an efficient way to do this because the two regressions can be done by sweeping first on the lag and intercept, then again on the trend.

```r
cmom
# y y{1} constant trend
```

We sweep on 2 and 3 to regress on \( Y{1} \) and \text{CONSTANT}. The regression coefficients are going to be in column 1 (non-pivot column). The sum of squared residuals is in (1,1). The \( t \)-stat can be put together using these and the (2,2) element.

```r
compute ss=%sweeplist(%cmom,||2,3||)
compute tmu=(ss(2,1)-1)/sqrt(ss(1,1)*ss(2,2)/(%nobs-2))
```

Now sweep on the trend as well and repeat

```r
compute ss=%sweep(ss,4)
compute ttau=(ss(2,1)-1)/sqrt(ss(1,1)*ss(2,2)/(%nobs-3))
```
4.7.2 Working With Polynomials

RATS provides a suite of functions for manipulating polynomials. These are represented as a VECTOR of the form \(v(1), \ldots, v(n + 1)\), which denotes the polynomial:

\[v(1) + v(2)x + \ldots + v(n + 1)x^n\]

- \%EQNLAPOLY(EQN,S) Extracts a lag polynomial from an equation.
- \%POLYADD(P1,P2) Adds two polynomials.
- \%POLYCXROOTS(P) Returns a VECTOR[COMPLEX] with all roots of P, represented as complex numbers.
- \%POLYSUB(P1,P2) Subtracts two polynomials.
- \%POLYDIV(P1,P2,D) Divides P1 by P2, expanded out to degree d.
- \%POLYMULT(P1,P2) Multiplies two polynomials.
- \%POLYROOTS(P) Returns a VECTOR with all the real roots of P.
- \%POLYVALUE(P,X) Returns the value of P at x.

The most important of these is \%EQNLAPOLY(eqn,series), which pulls a lag polynomial out of an equation or regression (the most recent regression if eqn=0). If series is the dependent variable, it extracts the standard normalization for the autoregression polynomial, that is, 1–lag polynomial from the right side.

For example, this implements an ARDL (autoregressive-distributed lag) model

```
linreg(define=ardl) logc
# logc{1 to 3} logy{0 to 3} seas{0 to -3}
*
compute arpoly=%eqnlagpoly(0,logc)
compute dlpoly=%eqnlagpoly(0,logy)
*
* Compute the long run effect
*
compute ardllt=%polyvalue(dlpoly,1)/%polyvalue(arpoly,1)
*
* Compute 8 lags of the expansion (degree 7 polynomial)
*
compute ardllest=%polydiv(dlpoly,arpoly,7)
```
4.7.3 Calendar Functions

RATS includes a perpetual calendar which is valid for dates from 1600 on. There are quite a few functions which allow you to access this. The first group of these analyze the date information for entries (E argument) in the current CALENDAR setting. A second group looks directly at year:month:day (Y,M,D arguments). Some of the functions take a day of the week as an argument, or return day of the week information. This is coded as 1=Monday to 7=Sunday.

%YEAR(E)     Year of entry
%MONTH(E)    Month of entry
%DAY(E)      Day of month of entry
%TODAY()     Entry for today’s date
%WEEKDAY(E)  Day of week of entry
%PERIOD(E)   Period number (panel data) or period within year

%JULIAN(E)  Days from Jan 1, 0001 to entry
%TRADEDAY(E, DOW)  Trading days for day of week in an entry
%DAYCOUNT(E)  Number of days in an entry
%DOW(Y,M,D)   Day of week of Y:M:D.
%EASTER(Y)   Dates after March 22 for (Western) Easter in year Y.

%FLOATINGDATE(Y,M,DOW,D)  Day of month for floating holiday which is a particular (D) day of the week within a month.

%CLOSESTDATE(Y,M,D,DOW)  The observance day of the month which is the closest DOW to Y:M:D.
%CLOSESTWEEKDAY(Y,M,D)  The observance day of the month which is the closest weekday (Monday-Friday) for Y:M:D.

Examples

cal(d) 1990:1:2
set wednesday = %weekday(t)==3

creates a dummy variable for the Wednesdays (weekday #3) in a daily data set.

cal(m) 1979:1
set d11 = %if(%month(t)==11,30-%floatingdate(%year(t),11,4,4),0.0)

creates a series which for the Novembers in a monthly data set has the number of days in November after U.S. Thanksgiving, which falls on the 4th (final argument in %FLOATINGDATE) Thursday (3rd argument) in November (2nd argument).
4.8 Creating Reports

As was mentioned earlier in the chapter, one of the advantages of using RATS is that you can do all the auxiliary calculations within the program, allowing you to reproduce results exactly, and eliminating the error-prone transfer of numbers from computer output to paper or a calculator. By using the instruction `REPORT`, you can take this a step farther and create full tables ready or nearly ready to be copied directly into a word processor. This again has the advantage of allowing you to avoid typing numbers.

You can do most of what `REPORT` can do by an ingenious use of the `DISPLAY` instruction, but `REPORT` is better designed to create the types of tables which are commonly used in reporting statistical results.

You can build only one report at a time. A report is designed as a table anchored at row 1 and column 1, similar to a spreadsheet, except both are numbered, rather than columns being lettered. And, like a spreadsheet, it’s effectively unlimited in size in each direction. You fill in entries by specifying the row and column at which data are to be placed. This can be done in whatever order you find convenient.

You will need to use at least three `REPORT` instructions to create a report:

1. `REPORT(ACTION=DEFINE)` initializes the report.
2. `REPORT(ACTION=MODIFY)` adds information to the report. (You actually don’t need to indicate `ACTION=MODIFY`, since that’s the default).
3. `REPORT(ACTION=SHOW)` indicates that the report is done.

There’s also `REPORT(ACTION=FORMAT)` which allows you to format the numerical data.

**ACTION=DEFINE**

There are two options you can use with `ACTION=DEFINE`: `HLABELS` defines header labels, and `VLABELS` defines row labels. Both are vectors of strings. While the header labels are considered separate from the data, the `VLABELS` are just the first column of data, and can be altered later by using `ACTION=MODIFY` on column one.

**ACTION=MODIFY**

`ACTION=MODIFY` fills one or more cells in a row at a row and column position in the table that you indicate. There are several ways to set the row and column positioning; you choose each separately:

1. You can do it directly by using `ATROW=row` and `ATCOL=column`.
2. You can open a new row or column with `ROW=NEW` or `COLUMN=NEW`.
3. You can use the same row or column as the last `REPORT` instruction with `ROW=CURRENT` or `COLUMN=CURRENT`.
4. (For `ROW` only) You can use `ROW=FIND` with `STRING=search string` to look for a match in column one. If no match is found, a new row is opened.
Beginning at the specified position, the expressions on the `REPORT` instruction are evaluated and, by default, placed into the cell positions running across the row. If you instead want the information inserted running down a column, use the option `FILLBY=COLUMNS`. Data in a `RECTANGULAR` or `SYMMETRIC` array will be inserted both in rows and columns in the same order as they appear in the array itself.

The following is a simple example of the use of `REPORT`: it runs autoregressions with lags 1 to 12, and displays a table showing the lag number in column one and the $R^2$ in the second column.

```plaintext
report(action=define,hlabels=\mid"Lags","R**2"\mid)
do lags=1,12
    linreg(noprint) lgdp
    # constant lgdp{1 to lags}
    report(row=new,atcol=1) lags %rsquared
end do lags
report(action=show)
```

You can also add some commonly used tags to the cells using the `SPECIAL` option. The choices are `PARENS`, `BRACKETS`, `ONESTAR` and `TWOSTAR`. `PARENS` encloses the cell in parentheses, as is often done for standard errors or $t$-statistics, while `BRACKETS` uses [ and ]. `ONESTAR` attaches a single * and `TWOSTAR` adds **. Finally, the `ALIGN=LEFT/CENTER/RIGHT` option lets you set the justification of text strings.

**ACTION=MODIFY,REGRESSION**

With `ACTION=MODIFY`, there is a special option which is designed specifically to handle the typical types of reports generated from several regressions. This is the `REGRESSION` option. For example,

```plaintext
report(action=define)
linreg foodcons
# constant prretail dispinc trend
report(regression)
linreg foodcons
# constant prretail dispinc{0 1}
report(regression)
report(action=show)
```

This produces the output shown below:

```
Constant  105.093801  90.277042
         (5.687192)  (6.085338)
PRRETAIL -0.325062 -0.149396
         (0.071043)  (0.067939)
DISPINC  0.315237  0.199063
         (0.032067)  (0.044460)
TREND   -0.224430
         (0.061217)
DISPINC{1}  0.063716
            (0.051319)
```
Chapter 4: Scalars, Matrices, and Functions

You use this after running a regression. It adds a new column and inserts the coefficients and (by default) the standard error for each of the regressors into it. The default behavior is shown above: it searches the first column for a match with each of the explanatory variables. If a match is found, the information is put into that row. If there is no match, it adds a new row and puts the regressor’s label in it, so that if a future regression includes that regressor as well, REPORT will find it and put the coefficient from the new regression in that row. Here, both regressions have Constant, PRRETAIL and DISPINC. The first includes TREND but not DISPINC{1}, and the second DISPINC{1} but not TREND.

If you’d rather have the table arranged so that the regressors in the same position within the regression are to be considered “parallel”, you can add the option ARRANGE=POSITION. With that, you also need the ATROW option to fix the row for the first regressor. This is necessary because you might very well want to include some summary statistics in addition to the regression coefficients, and those should go at the top to allow room for the table to grow at the bottom.

Your choices for the extra information to be included with the coefficient are specified with the EXTRA option. The default for that is EXTRA=STDERRS. You can also choose TSTATS, BOTH and NEITHER. If you choose BOTH, each regressor will get three lines in the table, while if EXTRA=NONE, each will get just one.

**ACTION=FORMAT**

This sets the numerical format for a selected set of cells in the table. You can either set a specific format using a picture code, using the PICTURE option, or set a column width with the WIDTH option, when REPORT will figure out the best format to display the covered data in the width you choose.

By default, this applies to the entire table; to format only a subset of the entries, use the options ATROW and TOROW to fix the row range and ATCOL and TOCOL to set the column range. If you do ATROW without a TOROW, the range will be all rows from the set row on, and similarly for ATCOL without TOCOL.

For instance, in the last example, if we decide that we want just three digits right of the decimal, we would insert the instruction

```plaintext
report(action=format,picture="*.###")
```

after the regressions, but before the ACTION=SHOW. Note that this only affects the formatting used in displaying data. The data themselves are maintained at full precision.

You can also use the TAG and SPECIAL options to flag the maximum or minimum values with stars or parentheses, and the ALIGN=DECIMAL option to decimal-align the values.
ACTION=SHOW

REPORT(ACTION=SHOW) displays the constructed table. By default, the output will be directed to the output unit (usually the output window). You can also display the report in a separate spreadsheet-style report window by including the option WINDOW="title of window". One advantage of this is that you can copy and paste the information from the window into a word processor or spreadsheet, or export it in a spreadsheet or HTML format. Finally, you can use the UNIT and FORMAT options to direct the report to a specific I/O Unit (usually a file opened with OPEN).

Examples

This makes some improvements on the two regression example from before. It includes the $R^2$ and Durbin-Watson for each regression, formats the numbers to three decimal places, and displays the output in a window.

Note that the $R^2$ and Durbin-Watson values are added after the command REPORT(REGRESSORS). This is done because REPORT(REGRESSORS) adds a column, so by doing that first, you can just use COLUMN=CURRENT to place the summary statistics.

```
report(action=define)
report(atrow=1,atcol=1) "R**2"
report(atrow=2,atcol=1) "DW"
linreg foodcons
# constant prretail dispinc trend
report(regressors)
report(atrow=1,column=current) %rsquared
report(atrow=2,column=current) %durbin
linreg foodcons
# constant prretail dispinc{0 1}
report(regressors)
report(atrow=1,column=current) %rsquared
report(atrow=2,column=current) %durbin
report(action=format,picture="*.###")
report(action=show,window="FOODCONS Regressions")
report(action=show)
```

The example on the next page reproduces Table 11.1 from Greene (2003), which shows a regression with standard errors computed using four different formulas. The header labels are the five regressor labels. Note the empty label at the start of that, since the first column has the row labels. The first row has the sample means of three of the variables. The second has the coefficient, the third the OLS standard errors, the fourth the $t$-statistics. All of these are inserted easily because the %BETA, %STDERRS and %TSTATS vectors have the information for all five regressors. Standard errors are computed using three other methods, and each one gets added into a new row. Finally, each of the columns is given a separate format.
REPORT(action=define, $
   hlabels=\text{"","Constant","Age","OwnRent","Income","Incomesq"\text{"}})$

linreg(smpl=posexp) avgexp
# constant age ownrent income incomesq
stats(noprint) age
compute agemean=%mean
stats(noprint) ownrent
compute ownmean=%mean
stats(noprint) income
compute incmean=%mean

REPORT(atcol=1,atrow=1) "Sample Mean\text{" " agemean ownmean incmean
REPORT(atcol=1,atrow=2) "Coefficient\text{" %beta
REPORT(atcol=1,atrow=3) "Standard Error\text{" %stderrs
REPORT(atcol=1,atrow=4) "t-ratio\text{" %tstats

linreg(smpl=posexp,robusterrors) avgexp
# constant age ownrent income incomesq

REPORT(atcol=1,atrow=5) "White S.E.\text{" %stderrs
compute %xx=%xx*(float(%nobs)/%ndf)
linreg(create,lastreg,form=chisquared)

REPORT(atcol=1,atrow=6) "D. and M.(1)\text{" %stderrs

linreg(smpl=posexp) avgexp / resids
# constant age ownrent income incomesq
prj(xvx=xvx)
set resdm = resids/sqrt(1-xvx)
mcov(lastreg) / resdm
compute %xx=%xx*%cmom*%xx
linreg(create,lastreg,form=chisquared)

REPORT(atcol=1,atrow=7) "D. and M.(2)\text{" %stderrs
*
REPORT(action=format,atcol=2,picture="*.##")
REPORT(action=format,atcol=3,picture="*.####")
REPORT(action=format,atcol=4,picture="*.#####")
REPORT(action=format,atcol=5,picture="*.####")
REPORT(action=format,atcol=6,picture="*.#####")
*
REPORT(action=show,window="Table 11.1")
Regression techniques lie at the heart of econometrics. This chapter covers a wide variety of methods for estimating linear models. Included are discussions of heteroscedasticity, serial correlation and instrumental variables.

This chapter focuses upon methods for estimating linear models. Chapter 6 describes testing procedures for examining your specifications more carefully, and Chapter 7 discusses non-linear models.
Chapter 5: Linear Regression Models

5.1 Annotated Regression Output

Overview
We begin by examining the standard regression output produced by RATS in some detail. We will also discuss some of the issues relating to the interpretation of these results, and describe the covariance/correlation matrix produced by the VCV option.

Caveat Emptor
A word of caution about regression results. RATS doesn’t give you a complicated set of options for choosing which regression summary statistics to display. Instead, RATS produces essentially the same set of results (with a few exceptions) for any type of regression. You should just ignore any information that has little or no value or meaning in the context of your regression model. You can’t assume that a particular test result is statistically valid just because it appears in the regression output. If you have any doubts, consult a good econometrics text to find out what types of information are applicable to your work.

The exceptions mentioned above are the $F$-test for the null hypothesis that all coefficients are zero, and the Ljung-Box $Q$ test for high-order serial correlation in the residuals. RATS only does the $F$-test for ordinary least squares regressions with a constant, since it is meaningless in most other situations. If you would like to see this tested in other situations, you can use an EXCLUDE instruction with all of the explanatory variables. Also, RATS computes $Q$ only in certain situations, such as estimates of an ARIMA model.

Sample Regression Output
We show here an annotated sample of the output from a LINREG instruction. Other regression instructions (such as AR1 and NLLS) produce similar output.

The output was produced using the first regression described in Chapter 1:

```
linreg rate 1960:2 1980:12
# constant ip m1diff ppisum
```

We will use the following notation in the descriptions of the various statistics:

- $y$ The vector of the dependent variable.
- $\tilde{y}$ The vector of deviations from the mean of the dependent variable:
  $$\tilde{y}_i = y_i - \bar{y}, \quad \text{where } \bar{y} = \frac{1}{T} \sum_{i=1}^{T} y_i$$
- $e$ The vector of residuals.
- $T$ The number of observations.
- $K$ The number of regressors.
Chapter 5: Linear Regression Models

(a) Linear Regression - Estimation by Least Squares
(b) Dependent Variable RATE
(c) Monthly Data From 1960:02 To 1980:12
(d) Usable Observations 251 Degrees of Freedom 247
(e) Centered $R^2$ 0.674259 (f) $R^2$ 0.670303
(g) Uncentered $R^2$ 0.945263 (h) $T \times R^2$ 237.261
(i) Mean of Dependent Variable 5.4658173331
(j) Std Error of Dependent Variable 2.4613624744
(k) Standard Error of Estimate 1.4132960737
(l) Sum of Squared Residuals 493.35923057
(m) Regression $F(3,247)$ 170.4239
(n) Significance Level of $F$ 0.00000000
(o) Log Likelihood -440.96454
(p) Durbin-Watson Statistic 0.189315

<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>-2.59825936</td>
<td>0.50111595</td>
<td>-5.18495</td>
<td>0.00000045</td>
</tr>
<tr>
<td>2. IP</td>
<td>0.16946855</td>
<td>0.01273584</td>
<td>13.30643</td>
<td>0.00000000</td>
</tr>
<tr>
<td>3. M1DIFF</td>
<td>-0.10954083</td>
<td>0.04099987</td>
<td>-2.67174</td>
<td>0.00804792</td>
</tr>
<tr>
<td>4. PPISUM</td>
<td>31.42821440</td>
<td>7.30011996</td>
<td>4.30516</td>
<td>0.00002408</td>
</tr>
</tbody>
</table>

Description
(a) The type of model and the estimation technique used
(b) The name of the dependent variable
(c) If you are using a CALENDAR, RATS will list the frequency of the data and the beginning and ending dates of the estimation range. If you are not using a CALENDAR, this line will be omitted.
(d) The number of usable entries in the estimation range and the number of degrees of freedom. The number of usable entries is the total number of entries in the range minus any missing observations. If observations are dropped from the middle of the estimation range (due to missing values or a SMPL option), RATS will also report the total number of observations, and the number of observations that were skipped or missing.

(e) The centered $R^2$ statistic:

\[
1 - \frac{e'e}{y'y}
\]

(f) The adjusted $R^2$ statistic ($\bar{R}^2$):

\[
1 - \left( \frac{e'e}{(T-K)/T-1} \right)
\]

(g) The uncentered $R^2$ statistic:

\[
1 - \frac{e'e}{y'y}
\]

(h) $T$ times the uncentered $R^2$:

\[
T \times \left( 1 - \frac{e'e}{y'y} \right)
\]
(i) The mean of the dependent variable:
\[ \frac{1}{T} \sum_{t=1}^{T} y_t \]

(j) The standard error of the dependent variable:
\[ \sqrt{\frac{\tilde{y}^2}{T - 1}} \]

(k) The Standard Error of the Estimate:
\[ \sqrt{\frac{e' e}{T - K}} \]

(l) The Sum of Squared Residuals:
\[ e'e \]

(m) The regression F statistic:
\[ \frac{\tilde{y}^2 - e'e}{(K - 1) / (T - K)} \]

RATS only does the F-test for a least squares regression with a constant.

(n) The marginal significance level of the F-statistic, with (K–1) and (T–K) degrees of freedom. The F-statistic tests the hypothesis that all coefficients in the regression (other than the intercept) are zero.

(o) The log-likelihood:
\[ -\frac{T}{2} \left( \log \left( \frac{e' e}{T} \right) + 1 + \log(2\pi) \right) \]

(p) Durbin–Watson statistic:
\[ \frac{\sum_{t=2}^{T} (e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2} \]

The Durbin–Watson statistic tests for first-order serial correlation in the residuals. The ideal result is 2.0, indicating the absence of first-order serial correlation. Values lower than 2.0 (and particularly below 1.0) suggest that the residuals may be serially correlated. RATS always computes a Durbin–Watson statistic. However, you should keep in mind the fact that the tabled values for the Durbin-Watson statistic are known to be invalid in a variety of circumstances, such as the presence of lagged dependent variables.

(q) The names of the explanatory variables. Lags are shown as name{lag}.

(r) The estimated coefficients. How these, and the standard errors, are computed depends upon the instruction and estimation method.

(s) The standard error of the coefficient estimate.

(t) The t-statistic = (Coefficient/Std. Error of Coefficient)

(u) The marginal significance level of a (two-tailed) test for a zero coefficient. For LINREG, NLLS, AR1, STWISE, BOXJENK, ESTIMATE, and ITERATE, the t-statistic in the preceding column is treated as a t distribution with T – K
degrees of freedom. For other instructions (such as **SUR** and **MAXIMIZE**), or when you use the **ROBUSTERRORS** option with **LINREG** or **NLLS**, the \(t\)-statistic is treated as a Normal distribution. See below for information on interpreting the significance level.

**Interpreting Test Results**

RATS reports probability results for test statistics by giving the marginal significance level (often called the \(P\)-value). This is the probability that a random variable with the appropriate distribution will exceed the computed value (in absolute value for Normal and \(t\) distributions). A \(P\)-value close to zero leads you to reject the null hypothesis. For instance, a marginal significance level of .02 would cause you to reject a hypothesis at the .05 level, but not at the .01 level. If you want to do a one-tailed test for a Normal or \(t\)-distributed statistic, divide the reported marginal significance level by two.

In the sample output shown on page 177, we can reject the zero-coefficient hypothesis at the one percent (.01) level for the **CONSTANT**, **IP**, and **PPISUM** coefficients. For the **M1DIFF** term, we can only reject the hypothesis at the five percent level.

**Covariance/Correlation Matrix**

If you use the **VCV** option available with **LINREG** and the other estimation instructions, RATS will print a covariance/correlation matrix of the coefficients. This is a square table with covariances on and below the diagonal, and correlations above the diagonal.

For example, in the table below, the covariance between the **PPISUM** and **M1DIFF** coefficients is 0.0069725731 (below the diagonal), and the covariance between the **M1DIFF** and **IP** coefficients is -0.0002032196. The correlation between the **PPISUM** and **M1DIFF** coefficients is 0.0247076367 (above the diagonal), and the correlation between the **M1DIFF** and **IP** coefficients is -0.5458365072.

Note that we use a backwards slash (\(\backslash\)) between the words “Covariance” and “Correlation” in the header because it looks like the diagonal of a matrix. The word “Covariance” lies to the left of the slash, reminding you that the covariance results appear on and to the left of the diagonal of the output matrix. Likewise, “Correlation” lies to the right of the slash, just as the correlation results appear to the right of the diagonal in the output.

<table>
<thead>
<tr>
<th></th>
<th>Constant</th>
<th>IP</th>
<th>M1DIFF</th>
<th>PPISUM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constant</strong></td>
<td>0.2384638018</td>
<td>-0.9603387865</td>
<td>0.3934354658</td>
<td>0.4714743624</td>
</tr>
<tr>
<td><strong>IP</strong></td>
<td>-0.0044362555</td>
<td>0.0000894874</td>
<td>-0.5458365072</td>
<td>-0.5523331605</td>
</tr>
<tr>
<td><strong>M1DIFF</strong></td>
<td>0.0075614818</td>
<td>-0.0002032196</td>
<td>0.0015489748</td>
<td>0.0247076367</td>
</tr>
<tr>
<td><strong>PPISUM</strong></td>
<td>1.6508553029</td>
<td>-0.0374646487</td>
<td>0.0069725731</td>
<td>51.4137629654</td>
</tr>
</tbody>
</table>
Chapter 5: Linear Regression Models

5.2 The SMPL Option: Outliers and Sample Selection

The SMPL Option

RATS provides two related methods for dropping observations out of the middle of a sample. **LINREG** and many other instructions have a **SMPL** option (short for SaMPLe) which allows you to include and omit selected observations within the *start* to *end* range of the instruction.

\[
\text{smpl=} \text{SMPL series or formula}
\]

The **SMPL series** is a series or formula with non-zero values at the entries (between *start* and *end*) you want to include in the estimation, and zero values at the entries to omit. It can be an existing series, or a formula like that used in a **SET** instruction. It’s usually a dummy variable series of some form. It may be a dummy in the data set, or one constructed for this particular purpose.

You can also set a **SMPL** series which applies to *all* instructions that support the **SMPL** option. You do this with

\[
\text{smpl(series=} \text{SMPL series})
\]

We prefer to use the **SMPL** options in most situations. If you use the **SMPL instruction**, you must remember to reset it when you are done with the analysis which uses the reduced sample.

Missing Values

RATS will automatically omit observations which have a missing value in the dependent variable or any of the regressors, so you don’t need a **SMPL** option for them. If you have a regression with lags, you could end up losing several data points since any observation which needs an unavailable lag of a variable will be dropped.

Skipping a Time Period

Suppose you want to leave the years 1942 to 1946 out of a regression over 1919 to 2002. You could construct a **SMPL series** for this with:

\[
\begin{align*}
\text{set notwar} & \ 1919:1 \ 2002:1 \ = \ 1.0 \quad \text{Set all entries to one} \\
\text{set notwar} & \ 1942:1 \ 1946:1 \ = \ 0.0 \quad \text{Set entries for war years to zero} \\
\text{linreg(smpl=notwar)} & \ \text{depvar} \\
\quad \# \ \text{regressors}
\end{align*}
\]

You can do the same thing with the option \text{smpl=}(t<1942:1.or.t>1946:1)

Using the **Start** and **End** Parameters

The *start* and *end* parameters, in conjunction with the **SMPL** option, provide further control over the estimation sample. To repeat the above regression but also omit observations after 1989, just add *start* and *end* parameter values:

\[
\begin{align*}
\text{linreg(smpl=notwar)} & \ \text{depvar} \ 1919:1 \ 1989:1 \\
\quad \# \ \text{regressors}
\end{align*}
\]
Subsamples Based on a Dummy

Suppose you have a cross-section data set and one of the variables is the dummy variable `MALE`, which is 1 for males and 0 for females. You can run separate regressions for the males and the females by doing the following:

```
linreg(smpl=male)  depvar  Use only the Males
#  regressors
linreg(smpl=.not.male)  depvar  Use only the Females
#  regressors
```

If you also have the data set classified by age and want to use a subsample of Males, age 18-25, the `SMPL` series for that (with `D1825` being the dummy for 18-25) would be `male*d1825`.

Subsample Based Upon Value

To run a regression for just those entries where a series exceeds some value or meets some similar criterion, use relational operators. This generates dummies for three subsamples based upon the series `POP`, then runs the regressions over those.

```
set small  = pop<2000
set medium = pop>=2000.and.pop<=6000
set large  = pop>6000
linreg(smpl=small)  depvar  #  regressors
linreg(smpl=medium)  depvar  #  regressors
linreg(smpl=large)  depvar  #  regressors
```

Subsample Within a Panel

To include only a certain set of time series entries within a panel data set, use the `%PERIOD` function (Section 2) to create the dummy variable.

```
calendar(panelobs=12)
allocate 103//12
...
linreg(smpl=%period(t)<=6) ...
```

will use only the first six time periods in each cross section.
Chapter 5: Linear Regression Models

Outliers

To omit outliers, you have to run a preliminary regression and save the residuals. Define what you mean by outlier and set a SMPL series which omits any observation which meets your definition.

This drops any observation for which the residual is larger than 3 standard errors.

```r
linreg expend / residual
# constant income
compute cutoff=3*sqrt(%seesq)
linreg(smpl=abs(residual)<cutoff) expend
# constant income
```

If the regression involves lags and you want to omit any observation which requires a particular data point, the best thing to do is to set that point as missing, and let RATS do the rest:

```r
compute product(1978:3)=%na
linreg(define=areq) product
# constant product{1 2 3}
```

Here, we set the 1978:3 observation of the dependent variable to the missing value code (%NA) because we want to omit it from the regression. RATS will automatically drop this observation from the regression due to the NA. Because the model depends on the one, two, and three-period lags of PRODUCT, it will also drop the 1978:4, 1978:5, and 1978:6 observations from the regression as one of the lag terms would be unavailable in all three cases.

Technical Information

In a regression using SMPL, the summary statistics ignore all omitted observations. The Durbin-Watson statistic is $d'$ from Savin and White (1978).

$$d' = \frac{\sum (u_t - u_s)^2}{\sum (u_t)^2}$$

where $u_s$ is the last included entry before $t$. 
5.3 Extensions to Linear Regression: A Framework

The Standard Normal Linear Model

The first multiple regression model examined in almost any econometrics text takes the form

\[ y_t = X_t \beta + u_t, \] with

\[ u_t \] assumed to be i.i.d. Normal with mean 0 and constant variance \( \sigma^2 \).

Assumption (2) is usually made conditional upon the full \( X \) matrix, or, even stronger, \( X \) is assumed to be independent of \( u \) entirely. Under those assumptions, least squares has all kinds of optimal properties: it’s the best linear unbiased estimator (BLUE), it’s the maximum likelihood estimator. Under very weak assumptions, it is consistent, and the least squares estimator is Normally distributed, even in finite samples.

However, these are assumptions which are unlikely to be met with non-experimental data. And they are particularly likely to fail with time series data. For instance, including a lag of \( y \) among the explanatory variables violates these assumptions. We need models which are appropriate more broadly than this.

Generalized Method of Moments

Much of the modern econometric theory of linear regression can be understood most easily by writing the model’s assumptions in the form:

\[ y_t = X_t \beta + u_t, \]

\[ EZ_t'u_t = 0 \]

This is made operational by replacing the expectation by the sample average, thus solving for \( \beta \)

\[ \Theta_T \frac{1}{T} \sum Z_t'(y_t - X_t \beta) = 0 \]

where \( \Theta_T \) is a weighting matrix, which comes into play if the dimensions of \( Z \) are bigger than those of \( X \). This is known as the Generalized Method of Moments (GMM for short). The unified theory of these estimators (covering non-linear models as well) was developed originally in Hansen (1982). “Generalized,” by the way, refers to the presence of the weighting matrix.

Assumption (4) sometimes comes directly from assumptions about the variables, or it is sometimes derived as the first order conditions of an optimization problem. For instance, least squares minimizes

\[ \sum (y_t - X_t \beta)^2, \] which has as its first order necessary conditions
Apart from constant multipliers, this is the same condition as would be derived from equation (3) plus the assumption

\[ EX' u = 0 \]

This is a relatively weak assumption, as it assumes nothing about the distribution of the residuals, merely requiring that the residuals be uncorrelated with the regressors. Under (3) plus (8), plus some regularity conditions which ensure good behavior of the sample averages, least squares gives consistent estimates of \( \beta \).

Consistent, however, doesn’t mean that the output from a LINREG instruction is correct. The standard errors, \( t \)-statistics and covariance matrix of coefficients (and thus any hypothesis tests which follow) are based upon stronger assumptions than merely (8). In particular, those calculations assume that \( u \) is homoscedastic (constant variance) and serially uncorrelated.

It’s possible to reformulate the model to deal with heteroscedasticity or serial correlation with specific forms, as will be discussed in Sections 5.4 and 5.5. However, if the precise form of this isn’t known, applying a generalized least squares (GLS) technique may, in fact, make matters worse. Plus, particularly in the case of serial correlation, the standard GLS method may give inconsistent estimates, as it requires much stronger assumptions regarding the relationship between \( X \) and \( u \).

ROBUSTERRORS

An alternative is to apply the simpler least squares technique for estimation, and then correct the covariance matrix estimate to allow for more complex behavior of the residuals. In RATS, this is controlled by the ROBUSTERRORS option, which is available on most of the regression instructions. The following is a (somewhat) technical description of this process: if you’re interested in the actual conditions required for this, consult a graduate level text like Hayashi (2000).

If \( \beta_0 \) is the true set of coefficients and \( \beta_T \) is the solution of (5), then (assuming that the estimates are consistent and that \( T \) is big enough), a first order Taylor expansion of (5) gives

\[ \Theta_T \left( \frac{1}{T} \sum Z_i' X_i (\beta_T - \beta_0) \right) \approx \Theta_T \left( \frac{1}{T} \sum Z_i' (y_i - X_i \beta_0) \right) \]

which can be rewritten:

\[ \sqrt{T} (\beta_T - \beta_0) \approx \left( \Theta_T \left( \frac{1}{T} \sum Z_i' X_i \right) \right)^{-1} \left( \Theta_T \left( \frac{1}{\sqrt{T}} \sum Z_i' (y_i - X_i \beta_0) \right) \right) \]

Assume that \( \Theta_T \) converges to a constant (full rank) matrix \( \Theta_0 \) (it’s just the identity matrix if we’re doing least squares). Assuming that \( X \) and \( Z \) are fairly well-behaved, it doesn’t seem to be too much of a stretch to assume (by a Law of Large Numbers) that the first factor is converging in probability to a constant matrix which can be
estimated consistently by (the inverse of)

$$A_T = \Theta_T \frac{1}{T} \sum Z'_t X_t,$$

The second factor is $1/\sqrt{T}$ times a sum of objects with expected value 0. Under the correct assumptions, some Central Limit Theorem will apply, giving this term an asymptotically Normal distribution. The tricky part about the second term is that the summands, while they are assumed to be mean zero, aren’t assumed to be independent or identically distributed. Under the proper conditions, this second term is asymptotically Normal with mean vector zero and covariance matrix which can be estimated consistently by

$$B_T = \Theta_T \frac{1}{T} \left\{ \sum_{k=-L}^{L} \sum_t Z'_t u_{t-k} Z_{t-k} \right\} \Theta_T,'$$

If there is no serial correlation in the $u$'s (or, more accurately in the $Zu$'s), $L$ is just zero. The choice of $L$ is governed by the option LAGS=correlated lags. We will call the term in braces in (9) $mcov(Z,u)$—short for matrix covariogram. This is the name of the RATS instruction which can compute it, although most of the time the calculations using it will be handled automatically within another instruction. The end result of this is that (in large samples) we have the following approximation for the distribution of the estimator:

$$\sqrt{T}(\beta_T - \beta_0) \sim N(0, A_T^{-1} B_T A_T'^{-1})$$

This general form for a covariance matrix occurs in many contexts in statistics and has been dubbed a “sandwich estimator.”

**Newey-West**

There is one rather serious complication with this calculation: when $L$ is non-zero, there is no guarantee that the matrix $B_T$ will be positive definite. As a result, hypothesis tests may fail to execute, and standard errors may show as zeros. (Very) technically, this can happen because the formula estimates the spectral density of $Z'_t u_t$ at 0 with a Dirichlet lag window, which corresponds to a spectral window which is negative for some frequencies (see Koopmans (1974), p. 306).

RATS provides a broader class of windows as a way of working around this problem. While the default is the window as shown above (which has its uses, particularly in panel data), other windows can be chosen using the LWINDOW option. Newey and West (1987) prove several results when the $k$ term in (12) is multiplied by

$$1 - \frac{|k|}{L + 1}$$

As a result, the covariance matrix computed using this is known in econometrics as Newey-West. The options ROBUSTERRORS, LAGS=lags and LWINDOW=NEWEYWEST will give you the Newey-West covariance matrix. See the instruction MCOV in the Reference Manual for a description of the other choices for LWINDOW.
5.4 Heteroscedasticity

We are using the term *heteroscedasticity* to mean that the equation errors are uncorrelated across observations, but have unequal variances in some systematic way. (It is sometimes used to define *any* departure from i.i.d. errors.) There are two common ways to deal with heteroscedasticity: weighted least squares (WLS) estimation, and the White (or Eicker-White) covariance matrix correction. Both are easily implemented in RATS, via the SPREAD and ROBUSTERRORS options respectively.

Note that neither of these methods is an appropriate remedy for heteroscedasticity that is the result of a poorly specified model. It’s possible that a better approach than “correcting” for the problem is to adjust the actual model, usually through a rescaling such as conversion to per capita values, to produce more equal variances.

Heteroscedasticity: WLS using the SPREAD Option

Use the SPREAD option on an estimation instruction to perform weighted least squares, that is, to correct for heteroscedasticity of a known form:

\[ y_i = X_i \beta + u_i, \quad \text{Var}(u_i) = V_i \]

To do weighted least squares, you only need to know a series which is proportional to \( V_i \). Weighted least squares improves the precision of the estimates by “downweighting” observations which are known to have a high residual variance.

**spread=** residual variance series

The residual variances, \( V_i \), are proportional to the entries of the residual variance series. Like the SMPL option, you can use a series or a formula.

Note that SPREAD specifies the *variances*, or at least a series proportional to them. The “WEIGHT” option used in many other statistics packages requests \( p_i = 1 / \sqrt{V_i} \).

The SPREAD option is available on the estimation instructions LINREG, STWISE, SUR, and ESTIMATE, and the related instructions MAKE, VCV, CMOM, MCOV, RATIO, PANEL, and PSTATS.

The SPREAD option implements WLS as follows: If \( p_i = 1 / \sqrt{V_i} \), and *weighted* variables are those multiplied by \( p_i \) and *unweighted* variables are not scaled, then:

- **Coefficients** computed by regressing \( p_i y \) on \( p_i X_i \) (*weighted* regression)
- **\( R^2 \)**, other goodness of fit computed from the *weighted* regression
- **Durbin-Watson** computed using *unweighted* residuals
- **Residuals** are *unweighted*

RATS leaves the residuals saved using the *residuals* parameter in unweighted form so they correspond with the actual data. If you use them later in a VCV or RATIO instruction, you must again use the SPREAD option.
Heteroscedasticity: The ROBUSTERRORS Option

With the ROBUSTERRORS option, LINREG computes the regression using least squares, but then computes a consistent estimate of the covariance matrix allowing for heteroscedasticity, as in Eicker (1967) and White (1980). Note that the coefficients themselves do not change, only their standard errors. For the case of a simple linear regression, the covariance matrix is computed using the results of Section 5.3 with $Z = X$, $L = 0$ and $\Theta = I$. This simplifies the formula quite a bit to

$$ (\beta_T - \beta_0) \sim N \left( 0, \left( \sum X'_t X_t \right)^{-1} \left( \sum u_t^2 X_t \right) \left( \sum X'_t X_t \right)^{-1} \right) $$

If you know the form of heteroscedasticity, this will not be as efficient as weighted least squares. The advantage is that you do not need to know the form. It’s possible to combine both of these ideas: for instance, if you think that the variance is related to population, but are unsure of the form of that relationship, you might estimate using a “best guess” form for the SPREAD option, then include ROBUSTERRORS to correct for any remaining problems.

Feasible GLS

While sometimes there may be good theoretical reasons for the spread function to take a particular form, more often all that may be known is that it is likely to be some function of some of the variables in the data set. Under those circumstances, you may be able to estimate the function and apply the weights obtained from that. This process is called Feasible or Estimated GLS (FGLS or EGLS). Greene (2003, Chapter 11) has an especially detailed look at the subject.

Because the spread function has to be positive, the most common form used for it is an exponential. The free parameters are estimated by a regression of the log of the squared residuals on some set of variables. If, for instance, you think the variance is proportional to some power of $POP$, the following can be used to estimate the unknown power and generate a series which can be used in a SPREAD option to correct for this.

```
set lusq = log(resids**2)
set logpop = log(pop)
linreg lusq
# constant logpop
prj logvar
set spread = exp(logvar)
```
Example 5.1 Heteroscedasticity

This is part of example 11.4 from Greene (2003). It estimates a model of credit card expenditures as a function of income and other variables. With this rather simple specification, it would be hard to imagine that the residuals aren’t larger for individuals with larger incomes, so the weighted least squares estimators examine a number of functions of income as spread series.

This is example HETERO.PRG.

```
open data incexp.txt
data(format=prn,org=columns) 1 100 mdr acc age income $
  avgexp ownrent selfempl *
set incomesq = income**2
set posexp   = avgexp>0

Restrict all estimates to the observations with positive expenditures

smpl(series=posexp)

  Linear regression with heteroscedasticity robust standard errors. The residuals are saved in the series RESIDS for use in the FGLS procedure below

linreg(robusterrors) avgexp / resids
  # constant age ownrent income incomesq

  Variance assumed proportional to income

linreg(spread=income) avgexp
  # constant ownrent income incomesq

  Variance assumed proportional to income squared

linreg(spread=incomesq) avgexp
  # constant ownrent income incomesq

  This is a FGLS (feasible GLS) procedure. The variance is assumed to be proportional to exp(a0+a1*income+a2*income**2) where a0, a1 and a2 are unknown. The parameters are estimated by a regression of log(resids**2) on 1, income and income**2. The spread series is the exp of the fitted values from that regression.

set logusq = log(resids**2)
linreg logusq
  # constant income incomesq
prj logvar
linreg(spread=exp(logvar)) avgexp
  # constant ownrent income incomesq
```
5.5 Serial Correlation

Correlation between errors at different time periods is a more serious problem than heteroscedasticity. Again, there are two basic ways to treat this: estimate by Generalized Least Squares (GLS) taking into account a specific form of serial correlation, or estimate by simple least squares, then compute a covariance matrix which is robust to the serial correlation.

For GLS, RATS offers only one “packaged” form of serial correlation correction, which is for first order. There is a well-developed literature on estimation with AR1 errors. More complicated error structures can be handled using filtered least squares (see FILTER in the Reference Manual), non-linear least squares (see Section 7.7), or an ARIMA–X model (see BOXJENK in the Reference Manual). Note, however, that econometric practice is moving away from tacking the dynamics onto a static model through the error term towards models which are designed to produce serially uncorrelated errors by incorporating the dynamics directly using lagged variables.

Serial Correlation: The AR1 Instruction

AR1 computes regressions with correction for first order autocorrelated errors by estimating a model of the form:

\[ y_t = X_t \beta + u_t, \quad u_t = \rho u_{t-1} + \epsilon_t, \]

Its syntax is very similar to LINREG, except that it has some options for choosing the estimation method for the serial correlation coefficient \( \rho \).

You have a choice between two optimization criteria, and within each, two ways of computing \( \rho \) (iterative or search):

- Simple least squares, skipping the first observation (METHOD=CORC and METHOD=HILU, Cochrane–Orcutt and Hildreth–Lu respectively).
- Maximum likelihood (METHOD=MAXL and METHOD=SEARCH).

(For completeness, RATS also offers Prais-Winsten by METHOD=PW).

The iterative methods (CORC and MAXL) are much faster; however, they don’t guarantee that you have found the global optimum. CORC, in particular, can converge to the wrong root if you have lagged dependent variables. As, with modern computers, speed is no longer much of an issue in these models, the default estimation procedure is the slower but steadier HILU.

In large samples, there should be only a slight difference between the methods, unless the results show multiple optima. However, in small samples, there can be a substantial difference between maximum likelihood and the least squares estimators. Maximum likelihood includes an extra data point, which can have a major impact when there aren’t many data points to start. Maximum likelihood generally steers the estimates of \( \rho \) away from the boundaries at plus and minus one, with the difference becoming more noticeable as \( \rho \) approaches those boundary values.
Serial Correlation: The ROBUSTERRORS Option

On LINREG, NLLS, NLSYSTEM, SUR, GARCH, LDV, DDV and MAXIMIZE, you can use the ROBUSTERRORS option, combined with the LAGS=correlated lags option, to compute an estimate of the covariance matrix allowing for serial correlation up to a moving average of order correlated lags. This is sometimes known as the HAC (Heteroscedasticity and Autocorrelation Consistent) covariance matrix.

ROBUSTERRORS is important in situations where:

- You do not know the form of serial correlation, so a particular generalized least squares (GLS) procedure such as AR1 may be incorrect, or
- GLS is inconsistent because the regressors (or instruments) are correlated with past residuals. Brown and Maital (1981) and Hayashi and Sims (1983) are two of the earliest examples of the proper analysis of such settings.

In some situations, the proper value of correlated lags is known from theory. For instance, errors in six-step-ahead forecasts will be a moving average process of order five. If not known, it has to be set to catch most of the serial correlation.

In general, it’s a good idea to also include the LWINDOW=NEWEWYST to get the Newey-West covariance matrix, or one of the other non-truncated windows. Otherwise the covariance matrix produced can be defective in a way that isn’t noticed until you try to use it to do hypothesis tests. Note that in the Newey-West window, if LAGS=1, the contribution of the lag term is cut in half. In a situation like this, even if the correlations are known to be zero after the first lag, you might be better off adding some extra lags just to make sure the Newey-West window doesn’t cut off most of the attempted correction.

Example

Section 6.8 in Hayashi (2000) analyzes the relationship between spot and forward exchange rates. A market efficiency argument implies that the regression

\[ A_t = \alpha + \beta P_t + u_t, \quad A_t = \text{actual depreciation} \text{ and } P_t = \text{predicted depreciation} \]

should have coefficients satisfying \( \alpha = 0, \beta = 1 \). However, because these are weekly data of 30-day forward rates, the residuals will almost certainly be serially correlated, up to four moving average lags. However, an AR1 “correction” won’t work. AR1, in effect, runs OLS on

\[ A_t - \rho A_{t-1} = \alpha(1 - \rho) + \beta(P_t - \rho P_{t-1}) + (u_t - \rho u_{t-1}) \]

The transformed residual \( (u_t - \rho u_{t-1}) \) can’t be assumed to be uncorrelated with the transformed regressor \( P_t - \rho P_{t-1} \), since the information which led to the surprise \( u_{t-1} \) will get incorporated into \( P_t \). Instead, the equation is estimated by OLS, with standard errors corrected for the serial correlation:

```
linreg(robusterrors, lags=4, lwindow=neweywest) s30_s
# constant f_s
```
Example 5.2 Serial Correlation

This is based upon the examples in section 12.9 in Gujarati (2003). It estimates a regression by OLS, in first differences, and with AR1 using three of the four methods. It also (re)does the OLS, but with HAC standard errors.

This is example file AR1.PRG.

cal(a) 1959
open data ar1.prn
data(format=prn,org=columns) 1959:1 1998:1

OLS regression

linreg y
# constant x

First difference regression. This may be the best choice if the autocorrelation coefficient is close to 1.

set dx = x-x{1}
set dy = y-y{1}
linreg dy
# dx

AR1 regression using several methods. HILU and CORC should give almost identical answers unless there are multiple roots.

ar1(method=hilu) y
# constant x
ar1(method=corc) y
# constant x
ar1(method=maxl) y
# constant x

OLS with Newey-West standard errors. This allows for autocorrelation of up to four lags.

linreg(robust, lags=4, lwindow=neweywest) y
# constant x
Chapter 5: Linear Regression Models

5.6 Instrumental Variables and Two-Stage Least Squares

Applicability

The RATS instructions `LINREG`, `AR1` (autocorrelation correction), `NLLS` (non-linear least squares), `SUR` (linear systems estimation) and `NLSYSTEM` (non-linear systems estimation) all support instrumental variables estimation. For the first three, RATS does (a variant of) two-stage least squares; for the last two, it is done using the Generalized Method of Moments. See Section 7.8 for more on GMM.

In all cases, you do the instrumenting by setting up the list of available instruments using the instruction `INSTRUMENTS`, and then including the `INST` option on the estimation instruction.

The Instruction INSTRUMENTS

RATS uses the `INSTRUMENTS` instruction to maintain the list of instruments. With smaller models, you probably need to set this just once. With larger simultaneous equations models, you may not have enough observations to use all the exogenous and predetermined variables. If so, you will probably need to change the list for each equation. Use the `ADD` or `DROP` options to make small changes to the list. `NLSYSTEM` has a special option (`MASK`), which can do a joint estimation with differing sets of instruments.

Note that you must set the instrument list before you do the estimation. A procedure in which the instruments depend upon the parameters being estimated cannot be done (easily) with RATS. Also, note that no variable, not even the `CONSTANT`, is included in an instrument set automatically.

Example

This estimates Klein’s Model I (see, for instance, Greene (2003), p 381). This program is supplied with RATS on the file `KLEIN.PRG`:

```plaintext
calendar(a) 1920
open data klein.prn
data(for=prn,org=obs) 1920:1 1941:1
table
set wagebill = privwage + govtwage
set capital = klagged1+invst
set trend = year-1920

instruments constant trend govtwage taxes govtexp $
                capital{1} prod{1} profit{1}
linreg(inst) cons
# constant profit{0 1} wagebill
linreg(inst) invst
# constant profit{0 1} capital{1}
linreg(inst) privwage
# constant prod{0 1} trend
```
Technical Information

RATS does not, literally, do two sets of regressions, though the effect is the same as if it did. Instrumental variables for a linear regression is based upon the assumptions used in Section 5.3:

1. \[ y_t = X_t\beta + u_t \]
2. \[ EZ'_tu_t = 0 \]

where \( Z_t \) are the instruments. In solving

3. \[ \Theta_T \frac{1}{T} \sum_i Z'_i(y_i - X_i\beta) = 0 \]
   the weight matrix is chosen to be

4. \[ \Theta_T = \left( \sum_i X'_i Z_i \right) \left( \sum_i Z'_i Z_i \right)^{-1} \]

which is the matrix of regression coefficients of the \( X \)'s on the \( Z \)'s. This gives

5. \[ \hat{\beta} = \left( \sum_i X'_i Z_i \right) \left( \sum_i Z'_i Z_i \right)^{-1} \left( \sum_i Z'_i y_i \right) \]

The calculation of the covariance matrix depends upon the options chosen. If you don’t use \texttt{ROBUSTERRORS}, the assumption is made that

6. \[ E(u_i \mid Z_i) = \sigma^2 \]

which gives us the covariance matrix estimate

7. \[ \hat{\sigma}^2 \left( \sum_i X'_i Z_i \right) \left( \sum_i Z'_i Z_i \right)^{-1} \left( \sum_i Z'_i X_i \right)^{-1} \]

More generally the covariance matrix will be estimated according to the formulas from Section 5.3.

If you use the \texttt{SPREAD} option with instrumental variables, all the formulas above are adjusted by inserting the reciprocal of the “spread” series into each of the sums:

8. \[ \Theta_T = \left( \sum_i X'_i (1/V_i) Z_i \right) \left( \sum_i Z'_i (1/V_i) Z_i \right)^{-1} \]

with similar changes to all of the other sums.
Example 5.3 Instrumental Variables

This \texttt{(INSTRUMENT.PRG)} is based upon example 9.5 from Wooldridge (2000). It estimates labor supply and labor demand functions using 2SLS. The \texttt{HOURS} and \texttt{LWAGE} (log wage) variables are assumed to be endogenous. In addition to the 2SLS estimates, this also estimates the reduced form regressions.

\begin{verbatim}
open data mroz.raw
data(format=free,org=columns) 1 428 inlf hours kidslt6 kidsge6 $ age educ wage repwage hushrs huseduc huswage faminc mtr $ motheduc fatheduc unem city exper nwifeinc lwage expersq *
set lwage = log(wage)
set expersq = exper**2

Estimate the labor supply function by OLS

\texttt{linreg hours}
\# constant lwage educ age kidslt6 kidsge6 nwifeinc

Now by 2SLS

\texttt{instruments constant educ age kidslt6 kidsge6 nwifeinc $ exper expersq}
\texttt{linreg(instruments) hours / resids}
\# constant lwage educ age kidslt6 kidsge6 nwifeinc

Check the reduced form for LOG(WAGE) and see if the instruments excluded from the labor demand function (EXPER and EXPERSQ) work

\texttt{linreg lwage}
\# constant educ age kidslt6 kidsge6 nwifeinc exper expersq

Estimate the wage equation by 2SLS

\texttt{linreg(instruments) lwage}
\# constant hours educ exper expersq

Estimate the reduced form for HOURS.

\texttt{linreg hours}
\# constant educ age kidslt6 kidsge6 nwifeinc exper expersq}
\end{verbatim}
5.7 Recursive Least Squares

The instruction **RLS** does Recursive Least Squares. This is the equivalent of a sequence of least squares regressions, but done in a very efficient way. The main purpose of recursive estimation is to determine if a model is stable. If a model is correctly specified as

\[ y_t = \beta x_t + u_t, \quad u_t \text{ is i.i.d. } N(0, \sigma^2) \]

the series of *recursive residuals* is

\[ (y_t - \beta_{t-1} X_t) / \sqrt{1 + x_t (X'_{t-1} X_{t-1})^{-1} x'_t} \]

for \( t > K \), where \( K \) is the number of regressors, and \( X_{t-1} \) is the matrix of explanatory variables through period \( t - 1 \), and \( \beta_{t-1} \) the estimate of \( \beta \) through \( t - 1 \).

These residuals have the property that they are also i.i.d. \( N(0, \sigma^2) \), while least squares residuals have a finite sample correlation. A failure of the recursive residuals to behave like this type of process will lead to rejection of the base assumptions: either the coefficients aren’t constant through the sample, or the variance isn’t constant, or both. See Section 6.7 for more on this.

Note that **RLS** is designed for recursive estimation only. If you need to do something more than that (for instance, you want to generate forecasts at each stage), you can use the instruction **KALMAN**, which does the same type of sequential estimation, but only updates one entry for each instruction. **KALMAN** can also handle situations where the true coefficients aren’t assumed to be time-invariant as they are above.

**RLS** has much the same format as **LINREG**, and the output will largely match that from a **LINREG**, since the **RLS** over the full sample will be the same as full sample least squares. Note, however, that **RLS** will not do instrumental variables estimation.

The residuals produced by **RLS** using the *resids* parameter are the recursive residuals, not the finite sample OLS residuals.

The special options that distinguish **RLS** from **LINREG** are:

- **COHISTORY** saves the sequential estimates of the coefficients in a *VEC* SERIES.
- **SEHISTORY** saves the sequential estimates of the standard errors in a *VEC* SERIES.
- **SIGHISTORY** saves the sequential estimates of the standard error of the regression into a series.
- **CSUMS** saves the cumulated sum of the recursive residuals in a series.
- **CSQUARED** saves the cumulated sum of squared recursive residuals in a series.
- **DFHISTORY** saves the regression degrees of freedom in a series.
Example

```
rls(csq=cusumsq) c / rresids
# constant y
compute rresids(1951:1)= %na
```

Post-sample predictive test. The cumulative sum of squares is in the series CUSUMSQ. While the numerator could also be written more simply as RRESIDS(1993:1)**2, we write it this way as it will generalize easily to more than one step.

```
cdf(title="Predictive Test") ftest (cusumsq(1993:1)\$ cusumsq(1992:1))*(%ndf-1)/cusumsq(1992:1) 1 %ndf-1
```

Harvey-Collier functional misspecification test. This can be done by just computing the sample statistics on the recursive residuals and examining the t-stat and significance level provided therein.

```
stats rresids 1952:1 1993:1
```

The ORDER option

By default, RLS generates the estimates in entry sequence. It first works through the initial observations until finding a set which generates a full rank regression. Entries are then added one by one, and the “history” series are updated. If you think that the more likely alternative to stable coefficients and variance is an instability related to some variable other than the entry sequence, you can use the option ORDER to base the sequence on a different variable. Note that this does not rearrange the data set. Keeping the data set itself in the original order is very important if the model includes lags. If you use ORDER, the entries in any of the series created as described earlier are maintained in their original order. You can use the option INDEX to find out the order in which entries were added to the model. This is a SERIES[INTEGER], defined from the first entry in the regression to the last. If, for instance, you use INDEX=IX for a regression run over 1974:3 through 2003:4, IX(1974:3) will be the smallest entry in the regression range for the order series, while IX(2003:4) will be the largest.

This is an example of the use of RLS with the ORDER option. It does an arranged autoregression test for a TAR (threshold autoregressive) model. The RLS does recursive estimation ordered on the first lag of DLR. The recursive residuals are then regressed on the explanatory variables. Under the null of no threshold effect, the coefficients should pass a test for zero values.

```
diff lr / dlr
set thresh = dlr{1}
rls(order=thresh) dlr / rresids
# constant dlr{1 2}
linreg rresids
# constant dlr{1 2}
exclude(title="Arranged Autoregression Test for TAR")
# constant dlr{1 2}
```
5.8 Systems Estimation

Background Information

RATS can estimate systems of *linear* equations with the instruction **SUR**. This uses the technique of Seemingly Unrelated Regressions for standard regressions, and three-stage least squares for instrumental variables.

If you need to estimate a system of *non-linear* equations, or have restrictions across equations other than simple equality constraints, use the instruction **NLSYSTEM** (Section 7.9). If you can (reasonably) use **SUR**, you should do so, as it is much faster and is usually much easier to set up.

**SUR** is very different from the single-equation estimation instructions. You specify the system using a set of equations, not regressor lists on supplementary cards. You need to construct these equations in advance using **EQUATION**, or possibly by **LINREG** with the option **DEFINE**.

Technical Information

We use the following basic notation:

1. \( y_{it} = X_{it}\beta + u_{it}, \quad t = 1, \ldots, T; \ i = 1, \ldots, N \)
2. \( E[u_i'u_i'] = \Sigma, \quad u'_i = (u_{1i}, \ldots, u_{Ni}) \)

The variables should be defined over the same intervals across the cross-section elements \( i \). RATS drops from the calculation any entry \( t \) for which *any* variable involved in the regression is missing.

These estimators are described in Section 14.2 of Greene (2003). The GLS (SUR) estimator is

\[
\beta = \left( X'(\Sigma^{-1} \otimes I)X \right)^{-1} X'(\Sigma^{-1} \otimes I)y
\]

where \( \beta \) and \( y \) are formed by stacking vectors from the \( N \) equations, and \( X \) is formed by stacking augmented \( X_i \) matrices: matrices with columns of zeros for explanatory variables in the other equations. The covariance matrix of the estimates is

\[
\hat{\Sigma} = \frac{1}{T} \sum_{t=1}^{T} \hat{\epsilon}_i\hat{\epsilon}'_i
\]

If you specify a matrix with the **CV** option, RATS will use that matrix for \( \Sigma \). Otherwise, it computes OLS (for **SUR**) or 2SLS (for **SUR(INST)**) estimates of the equations and uses the estimated covariance matrix of the residuals:
Chapter 5: Linear Regression Models

Iterating

By default, RATS applies a single application of feasible GLS, as described above. However, if the \texttt{ITER} option is non-zero, RATS recomputes $\Sigma$ and reestimates the system. It repeats this process until either the iteration limit is exceeded, or the change in coefficients is small. The convergence criterion is controlled by the option \texttt{CVCRIT}. See Section 7.1 on how RATS determines convergence.

The \texttt{CVOUT} option saves the $\Sigma$ at the final estimates, which is not the same as the $\Sigma$ used in computing the estimates.

Inputting a $\Sigma$

You can input your own value for the $\Sigma$ matrix using the \texttt{CV} option. As noted above, by default RATS computes $\Sigma$ from a preliminary estimate of the model. If you have too few observations to estimate $\Sigma$ freely (you need at least as many observations as equations), you will have to input $\Sigma$ yourself.

Note that the estimates of the standard errors of coefficients will be incorrect if your input $\Sigma$ doesn’t provide estimates of the residual variances.

Notes on Estimation

Some programs use a slightly different estimate of $\Sigma$, preferring the formula

\begin{equation}
\hat{\Sigma} = \frac{1}{T - K} \sum_{t=1}^{T} \hat{u}_t \hat{u}_t'
\end{equation}

where $K$ is the number of regressors per equation (assuming a system with equations of a similar form). This doesn’t affect the coefficient estimates, since a scale factor in $\Sigma$ will cancel out. However, the standard errors will be slightly different.

Transforming Equations into FRMLs

If you need to transform the equations into formulas for system simulation (Chapter 11), you have two options.

1. You can use a set of \texttt{FRML} instructions with the \texttt{EQUATION} option after the \texttt{SUR} instruction. See below.

2. You can use the \texttt{FRML} option on the \texttt{EQUATION} instructions you use to define the model.

\begin{verbatim}
sur(inst) 3
# conseq
# investeq
# profeq
frml(equation=conseq) consfrml
frml(equation=investeq) invfrml
frml(equation=profeq) proffrml
\end{verbatim}
Chapter 5: Linear Regression Models

Hypothesis Tests
The test procedures of Chapter 6 work somewhat differently for **SUR** than for **LINREG** and other single-equation estimation procedures. The main differences are:

- You cannot use **EXCLUDE** and **SUMMARIZE** for a test involving a regressor which appears in more than one equation of the system. This is because these two instructions list restrictions according to the regressor names, which won’t uniquely identify a coefficient.
- With **RESTRICT**, **MRESTRICT** and **TEST**, keep in mind that the coefficient numbers are based upon the position in the “stacked” system. For instance, if the first equation has 3 variables, the coefficients in the second equation will have numbers 4, 5, etc.
- **RESTRICT(CREATE)** will not work because it is designed for a single equation and cannot properly handle the printing for a multivariate system. Do **RESTRICT(REPLACE)** followed by **SUR(CREATE)** instead.
- RATS uses the second set of formulas from Section 6.2. Thus, it reports the tests as $\chi^2$, not $F$.

Restrictions Across Equations
**SUR** can estimate a system subject to equality constraints across the equations using the **EQUATE** parameter fields:

```
equation geeq ige
# constant fge  cge
equation westeq iwest
# constant fwest cwest
sur 2 / equate 2 3
# geeq
# westeq
```

which forces the second (Fxx) and third (Cxx) coefficients to be equal across equations. See the example on the next page.

For more complicated restrictions, use **NLSYSTEM** instead. Example 7.3, for instance, is actually a linear system, but **NLSYSTEM** is used because of the restrictions.

Generalized Method of Moments
While the default behavior for **SUR** with the **INSTRUMENTS** option is three stage least squares, it has options for supporting more general weightings of the instruments. See Section 7.8 for more information about this.
Example 5.4 Grunfeld’s Investment Equations: Use of SUR

This example computes joint GLS estimates for Grunfeld’s investment equations (a data set which is used extensively in Chapter 7 of Theil (1971) and in Chapter 16 of Gujarati (2003)). This program computes OLS estimates, then invokes SUR, first without restrictions, then with the restriction that the second and third coefficients are identical across equations.

This is example file SUR.PRG.

```
cal 1935
open data grunfeld.wks
data(org=obs,format=wks) 1935:1 1954:1 ige fge cge $
   iwest fwest cwest *
equation  geeq  ige
   # constant  fge  cge
   equation  westeq  iwest
   # constant  fwest  cwest *
linreg(equation=geeq)  ige               OLS estimates
linreg(equation=westeq)  iwest *
sur(vcv) 2
   # geeq
   # westeq *
sur 2 / equate 2 3
   # geeq
   # westeq
```
5.9 Distributed Lags

Background

We are using the phrase distributed lag to refer to a model with a form such as

\[ y_t = \sum_{l=0}^{L} \beta_l X_{t-l} + u_t \]

where the explanatory variable \( X \) is different from \( y \). This distributes the effect of \( X \) on \( y \) across a number of lags, hence the name. We are not using this term when \( X \) is the same as \( y \). We refer to such models as autoregressions. They have very different statistical properties from the models which we are describing here.

Using RATS

RATS offers several major advantages for distributed lag estimation, among them:

- You can specify lag ranges using the compact and flexible notation: `series{lag1 TO lag2}`.
- RATS computes the estimates in a memory and time-efficient fashion.
- RATS can easily handle polynomial distributed lags, as well as splines and other restricted forms.
- You can use spectral methods for estimation or analysis of results.

Summary Measures

Because of multicollinearity, the individual coefficients in a distributed lag regression usually are poorly determined. Thus we are interested more in summary measures, in particular, the sum and the shape of the lag distribution. There are several instructions or options which can be very helpful in obtaining this information:

- The instruction `SUMMARIZE` is designed specifically for computing the sum of a set of lag coefficients.
- The lag coefficients themselves can be extracted from the `%BETA` vector defined by `LINREG`.

This little piece of code shows both at work. `LINREG` computes a regression of `LONGRATE` on `CONSTANT` and lags 0 to 24 of `SHORTRATE`. The lag coefficients are pulled into `LAGDIST`, using `SET`. The lag coefficients are in slots 2 through 26 of `%BETA`, corresponding to lags 0 to 24, in order. The `NUMBER` option on `GRAPH` causes it to label entry 1 as 0, 2 as 1, etc, thus getting the lag labels correct.

```
linreg longrate  
# constant shortrate{0 to 24}  
summarize  
# shortrate{0 to 24}  
set lagdist 1 25 = %beta(t+1)  
graph(header="Plot of Lag Distribution",number=0)  
# lagdist 1 25
```
Chapter 5: Linear Regression Models

Estimation Techniques

There are four basic ways to approach distributed lag estimation:

1. Unrestricted long lags, as advocated by Sims (1974a).
2. Data-determined lag length with no shape restrictions.
3. “Hard” shape restrictions, such as polynomial (Almon) distributed lags or splines.
4. “Soft” shape restrictions, such as Shiller’s smoothness prior (Shiller (1973)).

Unrestricted long lags are easy to estimate—see the code fragment on the previous page. The others take a bit more work, though we do have a procedure for doing standard polynomial distributed lags (PDL’s).

Data-determined lag lengths

These use one of the information criteria such as Akaike or Schwarz (Section 5.10), or a general-to-specific testing strategy to find the "optimal" lag length.

“Hard” restrictions

These are done using restricted least squares (Section 5.11) typically with ENCODE and LINREG(UNRAVEL...). You create the ENCODEing matrix with EWISE if a single formula can suffice, or, if not, with COMPUTE instructions. You should note that the t-statistics on individual coefficients may end up being extremely high in such a regression. This is because each tests whether its coefficient can be made zero while maintaining the shape restriction, which is often nearly impossible.

Geometric distributed lags (or Koyck lags), which are “hard,” infinite (rather than finite) lags, are a special case of transfer functions. See the instruction BOXJENK for more on that.

“Soft” restrictions

Compute these by mixed estimation (Section 5.12).

Section 5.10 through 5.12 discuss these methods in general. Each section is followed by an example which applies it to the same problem: estimating a distributed lag of long-term interest rate series (composite yield on long-term U.S. Treasury bonds) on a short-term one (yield on 90 day Treasury bills), monthly from 1947:1 to 2007:4. We will be computing (up to) 24 lag distributed lags. The code to estimate an unrestricted distributed lag is shown on the previous page.
5.10 Information Criteria

A number of criteria have been proposed for allowing the data to determine the length of a distributed lag (or for model selection more generally). When applied in this context, all involve using a function of the residual sum of squares combined with a penalty for large numbers of parameters. We select the lag length by minimizing the function over different choices for the length of lag.

Of these, the two most commonly used are the Akaike Information Criterion (AIC) (Akaike (1973)) and the Schwarz Criterion (Schwarz (1978)) which is variously known as the SC, SIC, SBC (Schwarz Bayesian Criterion) or BIC (Bayesian Information Criterion). There are a number of equivalent forms of these—the ones that we will use are

\[
\text{Akaike: } \log \left( \frac{RSS}{T} \right) + 2 \frac{K}{T}
\]

\[
\text{Schwarz: } \log \left( \frac{RSS}{T} \right) + \log \left( \frac{T}{T} \right) \frac{K}{T}
\]

where \( K \) is the number of regressors and \( T \) is the number of observations.

The Schwarz criterion puts a heavier penalty on additional parameters and so it will never choose a model larger than Akaike. There are conflicting theoretical results about which of these is “better.” If the correct model is included in the collection of models examined, SBC will, given enough data, choose it, while AIC won’t do so necessarily—even in very large samples, it can pick models which are too big. (SBC is “consistent”, AIC isn’t). However, if the correct model isn’t included (for instance, the actual lag length is infinite), then AIC proves to be better at picking an approximate model. (AIC is “efficient”, SBC isn’t). For more information, see the discussion in Brockwell and Davis (1991).

REGCRITS Procedure

The procedure REGCRITS is available to compute the information criteria after a regression. In addition to the AIC and SBC, this also computes the Hannan–Quinn and (log) FPE (Final Prediction Error) criteria, which have their own proponents.

To use this, just do

@REGCRITS

after running a regression (you can either SOURCE in REGCRITS.SRC explicitly first, or just execute the procedure and let RATS search for the procedure file).

Use the NOPRINT option if you want to suppress the output—the computed criteria are stored in the variables %AIC, %SBC, %HQCRIT and %LOGFPE (log of the FPE).
Example 5.5 Choosing Lag Length: Information Criteria

This computes the Akaike and Schwarz criteria for the distributed lag model used in the previous examples. The program does not print any of the regressions. Instead it simply prints a table of the two criteria. Once you see the table you can easily run LINREG with the appropriate lag length(s).

It’s very important that you run the regressions over the same interval so they will be directly comparable. The most common use of the criteria is to select a lag length (or lengths in the case of an ARIMA model). If you just run the regressions without prior planning, the regressions for the short lags will use more data points than those with long lags. If the early data points turn out to be harder to fit than the later ones, this will tend to favor longer lags, even if they are of little help in an “apples vs apples” comparison. The easiest and fastest way to carry out the set of regressions is to use CMOMENT combined with LINREG(CMOMENT): the regressions will come out of the same cross-product matrix, assuring uniformity in the sample and also cutting the calculation time.

This is example file AKAIKE.PRG:

\[
\begin{align*}
\text{open data haversample.rat} \\
\text{calendar(m) 1947} \\
\text{data(format=rats) 1947:1 2007:4 fltg ftb3} \\
\text{set shortrate = ftb3} \\
\text{set longrate = fltg} \\
\ast \\
\text{cmom} \\
\text{# constant shortrate\{0 to 24\} longrate} \\
\text{report(action=define,hlabels=\{$|Lags|,|Akaike|,|Schwarz|\$\})} \\
\text{do maxlag=0,24} \\
\quad \text{linreg(cmom,noprint) longrate} \\
\quad \text{# constant shortrate\{0 to maxlag\}} \\
\quad \text{compute akaike =log(\%rss/\%nobs)+\nreg*2.0/\%nobs} \\
\quad \text{compute schwarz=log(\%rss/\%nobs)+\nreg*\log(\%nobs)/\%nobs} \\
\quad \text{report(row=new,atcol=1) maxlag akaike schwarz} \\
\text{end do} \\
\text{Tag with a * the minimum in the 2nd and in the 3rd columns.} \\
\text{report(action=format,tag=minimum,special=onestar,atcol=2,tocol=2)} \\
\text{report(action=format,tag=minimum,special=onestar,atcol=3,tocol=3)} \\
\quad \text{Use a common format of three decimals to the right. Force the numbers to align on the decimal point for easy reading.} \\
\text{report(action=format,picture="*.###",align=decimal)} \\
\text{report(action=show)}
\end{align*}
\]
5.11 Restricted Regressions

For linear models, RATS provides two methods for computing restricted regressions. The first is to use **ENCODE** to construct new variables as linear combinations of regressors and estimate the model using these constructed variables. You can use the **UNRAVEL** option on the estimation instruction to report coefficient values and other statistics in terms of the original variables. The second method is to estimate the unrestricted model first, and then use **RESTRICT** or **MRESTRICT** with the **CREATE** option to compute the restricted regression. Both methods are described below.

For non-linear models, you can code restrictions directly into your **FRMLs** or impose restrictions on the **NONLIN** instruction. Inequality constraints can be handled only within a non-linear framework.

**With ENCODE and UNRAVEL**

The **ENCODE** instruction combined with the **UNRAVEL** option provides one of the two methods available in RATS for computing restricted regressions. They work by reparameterizing the regression to incorporate the restrictions, coding the restrictions into the regressors.

When you include the **UNRAVEL** option on the estimation instruction, RATS substitutes back for the original regressors. It does not print the regression in terms of the coded variables, just the final form. For example, a simple equality constraint on two coefficients can be handled using:

*Set up 1x2 matrix with a weight of 1.0 on both series:*

```plaintext
declare rect r(1,2)
compute r=||1.0,1.0||
```

*Create the encoded variable Z = X1 + X2:*

```plaintext
encode r / z
# x1 x2
```

*Estimate the regression. With UNRAVEL, results will be reported in terms of X1 and X2 rather than Z:*

```plaintext
linreg(unravel) rate
# constant z
```

The most common use of these instructions is computing restricted distributed lags such as polynomial distributed lags (see Example 5.4):

```plaintext
dec rect r(4,13)
ewise r(i,j)=j**(i-1)
encode r / enc1 enc2 enc3 enc4
# x{0 to 12}
linreg(unravel) y
# constant enc1 enc2 enc3 enc4
```
Chapter 5: Linear Regression Models

With RESTRICT or MRESTRICT

The similar instructions RESTRICT and MRESTRICT offer the other way for doing restricted linear regressions. They impose the restriction after the unrestricted model has been estimated.

After you do the unrestricted regression, apply either RESTRICT(CREATE) or MRESTRICT(CREATE) to add the restrictions. With CREATE, these do the following:

1. Compute the new coefficient vector and covariance matrix of coefficients subject to the restrictions. See Section 6.2 for the formulas used in this procedure.
2. Compute new summary statistics. RATS recomputes all the standard variables, such as %RSS and %NDF.
3. Replace the old regression with the new restricted regression: any further hypothesis tests will apply to the restricted regression, not the original one.

For example:

```
linreg logq
# constant logk logl loge
restrict(create) 1
# 2  3  4
# 1. 1. 1. 1.
```

forces the coefficients 2, 3, and 4 (the coefficients on the series LOGK, LOGL, and LOGE) to sum to 1.

Which to Choose

In general, the two methods are almost reciprocals of each other when it comes to the difficulty of setting them up. One or two restrictions (if they apply across a number of regressors) will require a large number of “encoded” variables. And an ENCODE/UNRAVEL which produces just a few series from a large number will require many direct restrictions. In general, the ENCODE and UNRAVEL combination will be useful mainly in the situations such as the tightly restricted distributed lags. Otherwise, use RESTRICT or MRESTRICT.

However, in many cases, the most straightforward way to handle a restricted estimation is to use non-linear methods (Chapter 7), even if the equation is fully linear. For instance, the last example can be done with NLLS, estimating both the unrestricted and restricted regressions using

```
nonlin(parmset=base) a0 bk b1 be
nonlin(parmset=constraint) bk+b1+be=-1.0
frml cobb logq = a0+bk*logk+b1*logl+be*loge
* Unrestricted:
nlls(frml=cobb,parmset=base) logq
* Restricted:
nlls(frml=cobb,parmset=base+constraint) logq
```
Example 5.6 Polynomial Distributed Lags

An unconstrained third degree PDL (lags 1 to L) takes the form:

\[(1) \sum_{k=1}^{L} \left( a + bk + ck^2 + dk^3 \right)X_{t-k} \]

where \(a, b, c\) and \(d\) are the free parameters. If you expand this sum, \(a\) is the coefficient on the sum of \(X_{t-k}\), \(b\) on \(kX_{t-k}\), etc. We use \texttt{ENCODE} to create these four series, which we then use in the regression.

- We impose an end constraint (polynomial is zero for lag \(L+1\)) by writing the polynomial as \((k-(L+1))(a+ bk + ck^2)\).
- We impose near (polynomial is zero for lag \(-1\)) and far constraints using \((k-(L+1))(k+1)(a+ bk)\).

The trick in handling the constrained PDL’s is to get the \texttt{EWISE} instruction, which creates the encoding matrix \(R\), correct. Because lag 0 \((k=0)\) corresponds to the first row in \(R\) \((J=1\) in \texttt{EWISE}), the roots should be at \(J=0\) \((k=-1)\) and \(J=L+2\) \((k=L+1)\). This example program is supplied with RATS on the file \texttt{PDL.PRG}. There is also a procedure which handles all the work included here, found on the file \texttt{PDL.SRC}.

```
open data rates.rat
cal(m) 1947
data(format=rats) 1947:1 1999:3 longrate shortrate
smpl 1951:1 1999:3
declare rect r

Unconstrained

dim r(4,25)
ewise r(i,j)=j**(i-1)
encode r / enc1 enc2 enc3 enc4
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc1 enc2 enc3 enc4

Far constraint

dim r(3,25)
ewise r(i,j)=(j-26)*j**(i-1)
encode r / enc1 enc2 enc3
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc1 enc2 enc3

Near and far constraints

dim r(2,25)
ewise r(i,j)=(j-26)*(j-0)*j**(i-1)
encode r / enc1 enc2
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc1 enc2
```
5.12 Ridge and Mixed Estimators

Background Information

Ridge regression (Hoerl and Kennard (1970)) and mixed estimation (Theil (1971), 347-352) are related procedures which are fairly easy to implement within RATS. These are Bayesian-like estimation techniques, in the sense that there are priors which will generate them as their posterior modes. See Chapter 13 for more precise methods of handling Bayesian estimation in RATS.

A ridge estimator of $\beta$ in $y = X\beta + u$ is defined as

$$
\hat{\beta} = (X'X + kI)^{-1}X'y
$$

for some positive $k$. Least squares is the special case of $k=0$. Setting $k>0$ has the effect of “shrinking” the estimate of the coefficients toward the origin, although individual coefficients may move away from zero.

In mixed estimation, non-sample information about $\beta$ is put in the form

$$
r = R\beta + v, \quad \text{Cov}(v) = V
$$

The mixed estimator (assuming $\text{Cov}(u) = \sigma^2 I$) is

$$
\hat{\beta} = (X'X + \sigma^2 R V^{-1}R)^{-1}(X'y + \sigma^2 R V^{-1}r)
$$

The ridge estimator is a special case of this where $R = I$, $r = 0$, and $V$ is scalar.

Computation Tools

Note that both techniques estimate $\beta$ as $(X'X + Q)^{-1}(X'y + q)$. The tools for implementing such estimators are the `CMOMENT` and `LINREG(CMOMENT)` instructions:

- `CMOMENT` computes the matrices $X'X$ and $X'y$ using the actual data. In practice, these will actually be segments of the same matrix. We then replace these matrices with $X'X + Q$ and $X'y + q$.

- `LINREG(CMOMENT)` computes the regressions using the altered versions of $X'X$ and $X'y$.

To simplify this process, on `CMOMENT` you should list the variables as regressors first, followed by the dependent variable. That way, the matrix %CMOM, as computed by `CMOMENT` will have the form

$$
\begin{bmatrix}
X'X & y'X \\
X'y & y'y
\end{bmatrix}
$$

(actually just the lower triangle of this), so $X'X$ is the top corner, and $X'y$ is all but the last element of the last row.)
Ridge Regression

Ridge regression is especially easy because all we need to do is add a constant to the diagonal elements of $X'X$. The simplest way to do this is with a `do` loop:

```plaintext
 cmoment
 # constant x1 x2 x3 x4 y
 linreg(cmoment) y
 # constant x1 x2 x3 x4
 do row=1,5
   compute %cmom(row,row)=%cmom(row,row)+k
 end do row
 linreg(cmoment) y
 # constant x1 x2 x3 x4
```

Choosing an appropriate value for $K$ is the real problem with ridge regression. See, for example, the discussion in Judge, Griffiths, Hill and Lee (1980).

Mixed Estimation

We need to construct the matrices $R$, $r$ and $V$ in some fashion. $r$ is often zero and $V$ is almost always diagonal, and is usually a scalar matrix (a constant times the identity), which simplify these considerably.

Since we do not know $\sigma^2$, in general, we must use an estimate $s^2$ of it. We do a preliminary `linreg` to get the estimate $s$ of $\sigma$, then use matrix commands to replace

\[
X'X \text{ by } X'X + s^2 R' V^{-1} R, \quad \text{and} \quad X'y \text{ by } X'y + s^2 R' V^{-1} r.
\]

Where $V$ is scalar (say $\lambda^2 I$), this can be done quite simply (if the `%cmom` array has been arranged as described on the previous page) by stacking $R$ on top of $r$ into $Rr$ and then adding to `%cmom` the array $(s^2 / \lambda^2)(Rr)'(Rr)$. This is what is done in Example 5.5. If $V$ is more general, it’s necessary to change the $X'X$ and $X'y$ matrices separately. This is done most easily using the `overlay` instruction.

Because the general case involves a bit of extra programming, we provide with RATS a procedure named `mixed` (on the file `MIXED.SRC`) which does the entire calculation. Its syntax is

```plaintext
@mixed( options ) depvar start end
# explanatory variables in regression format
```

The options are

- `capr=R matrix`
- `lowr=r matrix`
- `scalar=scalar for V being a scalar matrix`
- `V=V matrix in general`
Example 5.7 Shiller's Smoothness Prior

Shiller (1973) proposed a method for distributed lags which reduces the variability in the lags seen in least squares estimates, while imposing a less stringent shape restriction than the PDL. His smoothness prior (of degree $d$) takes the form:

$$(1 - L)^{d+1} \beta_i = v_i$$

where $\beta$ is the vector of lag coefficients. We do this using mixed estimation (Section 5.8). Since the dummy observations have a fairly simple form (in particular, the $V$ matrix is scalar), we will do the estimates directly.

FMATRIX creates these dummy observations. Note that the rows of $R$ in $R\beta=v$ take precisely the FMATRIX form with the option DIFF=$d+1$. We use several values for the weight put on these.

This example uses a first degree prior ($d=1$) with lag zero of SHORTRATE left free. Thus, there are 22 rows in the $R$ matrix: the 24 affected lags minus ($d+1$). FMATRIX starts at entry (1,3) because the CONSTANT and lag zero are the first two regressors in the cross moment matrix. Since we need the original $\%CMOM$ array each time through the DOFOR POWER loop, we store it in the array CMOMHOLD.

The example is supplied on the file SHILLER.PRG. See also GIBBS.PRG, which does the same model using Gibbs sampling.

```plaintext
open data haversample.rat
calendar(m) 1947
data(format=rats) 1947:1 2007:4 fltg ftb3
set shortrate = ftb3
set longrate = fltg
cmom
   # constant shortrate{0 to 24} longrate
linreg(cmom) longrate
   # constant shortrate{0 to 24}
*
declare rect dummy(22,27)
declare symm cmomhold(27,27) dummyxx(27,27)
fmatrix(diff=2) dummy 1 3
compute cmomhold=%cmom
compute dummyxx =tr(dummy)*dummy
*
compute scalefac=%seesq/.01
dofor [real] power = 1.0 2.0 4.0 8.0
   compute kfactor=power*scalefac
   compute %cmom =cmomhold+kfactor*dummyxx
   compute title=
      "Shiller Smoothness Prior with k="+%strval(kfactor,"*.##")
   linreg(cmom,title=title) longrate
   # constant shortrate{0 to 24}
end dofor
```
5.13 Recomputing a Covariance Matrix

Background Information

Current statistical practice includes a large set of procedures which produce consistent estimates for coefficients, while giving inconsistent estimates of the covariance matrix of the estimated coefficients. For example:

- Least squares with heteroscedasticity or serial correlation of the residuals.
- Robust estimators computed with iterated weighted least squares.

RATS can handle the first automatically if you use the ROBUSTERRORS option, as described in Section 5.3. You need to do the computations yourself in other situations. This relies upon an extension of the general results from that section.

Computational Strategy

A basic strategy for computing the covariance matrix which works in a wide variety of situations is the following: if the estimator solves the first order conditions

\( \sum X_i' f(y_i - X_i\beta) = 0 \)

for some function \( f(u) \), the covariance matrix of \( \hat{\beta} \) can be estimated by \( A^{-1}BA^{-1} \) where

\( B = \sum X_i'X_i f^2(y_i - X_i\beta), \) and

\( A = \sum X_i'X_i f'(y_i - X_i\beta) \)

You might have to change the sign on \( A \) to make it positive definite. A sign change will cancel out of the calculation because \( A^{-1} \) appears twice. You can compute \( B \) and \( A \) with the instruction MCOV using as residuals (in the notation of MCOV):

- the series \( f \) for computing \( B \)
- the series \( f' \) with the option NOSQUARE for computing \( A \).

Use the following steps:

1. Compute your estimates.
2. Compute the series of values \( f(u) \) and \( f'(u) \).
3. Compute \( A \) and \( B \) using MCOV.
4. Replace \%XX with \( A^{-1}BA^{-1} \) using the instruction
   \( \text{compute } \%xx=\%mqform(b,\text{inv}(a)) \)
5. Use LINREG with the options CREATE and FORM=CHISQUARED to reprint the regression, using the new standard errors and covariance matrix. We describe these options on the next page.
Chapter 5: Linear Regression Models

Using LINREG(CREATE)

If you use LINREG with the option CREATE, LINREG does not compute the regression. Instead, it prints standard regression output using 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=f/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, as described on the next page.

title="Title to identify the estimation procedure" ("User")
You can use this to include a phrase identifying the estimation method inside the regression output.

reccorr=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 is different from the preceding one (meaning you cannot 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.
The FORM Option

Section 6.2 lists two sets of formulas used by the hypothesis testing instructions:

1. FORM=F: These use as the covariance matrix of coefficients $\sigma^2 (X'X)^{-1}$ (or its equivalent). $(X'X)^{-1}$ is stored in the array %XX by LINREG, NLLS or AR1. The regression instruction computes an estimate $\hat{\sigma}^2$ of $\sigma^2$ and stores it internally.

2. FORM=CHISQUARED: These use as the covariance matrix the array %XX alone, that is, %XX does not need to be scaled by the residual variance. This method is used for DDV, LDV, GARCH, SUR, MAXIMIZE, NLSYSTEM, CVMODEL or DLM, and also either LINREG or NLLS with the ROBUSTERRORS option.

Normally, RATS automatically chooses which to use based upon which estimation technique was used last. However, procedures such as those described in this section change the situation to one in which FORM=CHISQUARED is correct. Use that option to inform RATS about this change. FORM=F/CHISQUARED is an option for LINREG with the CREATE option, and on EXCLUDE, SUMMARIZE, TEST, RESTRICT and MRESTRICT. Note that if you use FORM=CHISQUARED on LINREG(CREATE), you do not need to use it on hypothesis tests which follow it.

Example

This does one of the Davidson-MacKinnon (1993) refinements on White standard errors. We first do the standard White errors using LINREG with ROBUSTERRORS.

```
open data states.wks
data(org=cols,format=wks) 1 50 expend pcaid pop pcinc

set pcexp = expend/pop

linreg(robusterrors) pcexp / resids
# constant pcaid pcinc

Use PRJ to help compute the standardized residuals. Those are used on the MCOV instead of the simple OLS residuals. (If you just put resids on the MCOV instead of resdm, you’d be computing the White covariance matrix).

prj(xvx=xvx)
set resdm = resids/sqrt(1-xvx)
mcov(lastreg) / resdm
compute %xx=%xx*%cmom*%xx
linreg(create,lastreg,form=chisquared,$
title="OLS with DM(2) Standard Errors")
```
5.14 Robust Estimation

Background

We concern ourselves with estimating $\beta$ in the model

$$y_i = X_i \beta + u_i$$

While least squares gives consistent and asymptotically Normal estimates of $\beta$ under a fairly broad range of conditions, it’s also well known that it is sensitive to outliers or a fat-tailed $u$ distribution. See, for instance, the discussion in Greene (2003), Chapter 16. It’s possible to drop outliers from the data set. However, if those observations can reasonably be assumed to be valid within the model (1), and not just recording errors or data points where the model simply fails, it might make more sense to choose an alternative estimator which won’t be as sensitive as least squares to tail values.

One such estimator is LAD (Least Absolute Deviations), which is

$$\hat{\beta} = \text{minimizer } \sum_{t} |y_t - X_t \beta|$$

This is provided by the RATS instruction RREG (Robust Regression). LAD is consistent and asymptotically Normal under broader conditions than are required for least squares. It will be less efficient if the $u$’s are better behaved, with about 60% efficiency for Normal $u$’s. Its main drawback relative to least squares is that it is much more difficult to compute—the minimand isn’t differentiable, and it requires a specialized variant of linear programming to compute the estimator. There is also some question about the best way to estimate the covariance matrix. See the description of the RREG instruction for technical details.

Some alternative robust estimators can be generated which have behavior in the tails similar to LAD, but near zero are more like least squares. By keeping the minimand differentiable, they can be estimated more simply: either by iterated weighted least squares or by a standard nonlinear estimation routine.

If you’re using iterated weighted least squares, computing the estimators themselves is rarely a problem, since you just need to put LINREG with the SPREAD option inside a loop which recalculates the SPREAD series. The more complicated part is computing the covariance matrix of the coefficients. The result of the iteration process gives an inconsistent estimate of this, as it does not take into account the dependence of the SPREAD series on $\beta$. 
Sample Procedure

For example, the following uses a convex function, twice differentiable everywhere, which is asymptotically $|u|$. 

\[
\min_{\beta} \sum_i \left( c^2 + u_i(\beta)^2 \right)^{1/2}
\]

$c$ is some constant. As $c \to 0$, this approaches LAD. The first-order necessary conditions (FONC's) for solution are

\[
\sum_i \frac{X'u_i}{(c^2 + u_i^2)^{1/2}} = 0
\]

which are the same as the FONC's for weighted least squares with $\text{SPREAD} = (c^2 + u_i^2)^{1/2}$

We can thus estimate $\beta$ by iterated weighted least squares.

Using the results from Section 5.3, we can compute the covariance matrix as $A^{-1}BA^{-1}$, where

\[
B = \sum_i X'_iX_i \frac{u_i^2}{(c^2 + u_i^2)}
\]

\[
A = \sum_i X'_iX_i \frac{c^2}{(c^2 + u_i^2)^{3/2}} \quad (\text{Note: we changed the sign on this})
\]

Example 5.8 Robust Estimation

This estimates a regression using the process described above. Note the use of the $\%\text{TESTDIFF}$ function inside the loop. This is a special function designed precisely for such situations. It computes the same convergence criterion used by the non-linear estimation instructions in RATS using “before and after” coefficient estimates.

This is based upon example 16.6 from Greene (2003). It’s example file $\text{ROBUST.PRG}$.

open data zellner.prn
data(format=prn,org=columns) 1 25 valueadd capital labor nfirm *
set logy = log(valueadd)
set logk = log(capital)
set logl = log(labor)

Compute linear regression, and the standardized residuals

linreg logy / resids
# constant logk logl
prj(xvx=px)
set stdresids = resids/sqrt(%seesq*(1-px))
Rerun the regression, omitting the outliers, defined here as observations with
standardized residuals greater than 2.5 in absolute value.

\begin{verbatim}
linreg(smpl=abs(stdresids)<=2.5) logy
# constant logk logl

Now do LAD estimator

rreg logy
# constant logk logl

Iterated WLS M-estimator

compute c = sqrt(%seesq)
dec vector beta0

Start with least squares (spread = constant). Do 10 iterations or until convergence

set spread = 1.0
do iters=1,10
    compute beta0=%beta
    set spread = sqrt(c**2+resids**2)
    linreg(noprint,spread=spread) logy / resids
# constant logk logl
    if %testdiff(beta0,%beta)<.0001
        break
end do iters

disp "Iterations Taken" iters

Compute the sandwich estimator for the covariance matrix

set f      = resids/spread
set fprime = c**2/spread**3
mcov(matrix=b,lastreg) / f
mcov(matrix=a,lastreg,nosquare) / fprime
linreg(create,lastreg,form=chisquared,$
    covmat=%mqform(b,inv(a)),$
    title="Iterated Weighted Least Squares")
\end{verbatim}
5.15 Miscellaneous

The Option DFC

dfc=Degrees of Freedom Correction

The DFC option is used to enter the degrees of freedom lost in the dependent variable in previous processing.

When you use regression residuals as the dependent variable, the standard errors of the estimated coefficients of the regression will be understated, because the first regression extracts some degrees of freedom. The Degrees of Freedom Correction should be equal to the number of regressors in the first regression, minus the number of those regressors which are repeated in the second regression.

\begin{verbatim}
linreg ip / detrend
# constant  trend  seasonal{  -10 to 0 }  13 regressors
linreg(dfc=12) detrend
# constant  intrate
\end{verbatim}

The ENTRIES Option

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

You can use the ENTRIES option on any instruction which has a supplementary card in regression format: \texttt{LINREG}, \texttt{CMOMENT}, \texttt{STWISE}, \texttt{PREGRESS}, \texttt{AR1}, \texttt{EQUATION} (applies to the supplementary card listing the \texttt{MORE} variables), \texttt{LDV}, \texttt{DDV}, \texttt{PRB}, \texttt{LGT}, \texttt{MAKE}, \texttt{ENCODE}, \texttt{MCOV}, \texttt{EXCLUDE} and \texttt{SUMMARIZE}. You can also use it for the supplementary card for \texttt{TEST}.

ENTRYs lets you put one of these instructions inside a loop and change the number of regressors on the supplementary card to use. ENTRIES is largely obsolete because you can use the \texttt{ENTER} instruction or one of the “reglist” functions to build supplementary cards in a \texttt{VECTOR[INTEGER]}.

An entry is any of the objects on the supplementary card, except for the control characters \# and \$. You must count the \{ and \} of a lag field. Thus \texttt{M1{1 \ TO \ 10}} counts as 6 entries (M1, {, 1, TO, 10, and }). RATS will use the number of entries specified by the ENTRIES option, starting from the first entry on the card.

The code below is part of a procedure which computes something similar to a vector autoregression, but with varying number of lags among the variables. NUMVAR is the number of variables in the system, VARS is a \texttt{VECTOR[INTEGERS]} with the list of variables, and LAGS is a \texttt{VECTOR[INTEGERS]} with the number of lags for each. This can handle anything up to a six variable system simply by changing NUMVAR.

\begin{verbatim}
do j=1,numvar
   linreg(entries=6*numvar+1) vars(j)
   # constant vars(1){0 to lags(1)}  vars(2){0 to lags(2)} $
   vars(3){0 to lags(3)}  vars(4){0 to lags(4)} $
   vars(5){0 to lags(5)}  vars(6){0 to lags(6)}
end do j
\end{verbatim}
Chapter 6
Hypothesis and Specification Testing

RATS has a large and flexible set of testing instructions. These allow you to test virtually any form of linear hypothesis. With the proper steps, you can implement even fairly complicated procedures such as a Hausman specification test.

This chapter describes and demonstrates tests for heteroscedasticity (Section 6.4), serial correlation (6.5), exogeneity or causality (6.6), structural stability (6.7), functional form (6.8), misspecification (6.9), unit roots (6.10) and cointegration (6.11).

RATS is also capable of doing “computationally intensive” testing procedures, such as those which use Monte Carlo methods, or randomized orderings. See Chapter 13 for more on that subject.

There are also many procedures available for a wide variety of hypothesis tests. Some are included with RATS, while many others—written by the RATS user community—are available for downloading from the Estima web site (www.estima.com).

Please note that RATS, unlike some other software, does not produce standard “batteries” of hypothesis tests after a regression. It may provide some comfort to see your model pass a whole set of “specification” tests, but because these are applied regardless of whether they are actually of interest in a specific situation, they may end up missing areas where your model might be failing.

Technical Details
Heteroscedasticity
Serial Correlation
Causality
Structural Stability
Specification
Unit Roots and Cointegration
Chapter 6: Hypothesis Testing

6.1 Introduction

Overview of Relevant RATS Instructions

RESTRICT This instruction can test any set of linear restrictions and can also be used for doing restricted regressions. This is the most general testing instruction, so it is the hardest to use. TEST, EXCLUDE and SUMMARIZE are more specialized, and easier to use.

MRESTRICT is similar to RESTRICT, but uses matrices rather than supplementary cards for the specification of restrictions.

EXCLUDE tests exclusion restrictions: a restriction that every one of a set of coefficients is zero.

TEST is a generalized version of EXCLUDE which allows testing restrictions of the form \( \beta_i = \text{constant} \). You can also use it to test for equality between two coefficient vectors.

SUMMARIZE computes a linear combination of coefficients and its standard error or a non-linear function of the coefficients with its linearized standard error.

RATIO tests restrictions on systems of several equations by forming a likelihood ratio statistic from the covariance matrices. Use it when the restricted and unrestricted maximum likelihood estimates of the system can be computed using single equation methods.

CDF provides a set of “tables” for the Normal, \( F \), \( t \) and \( \chi^2 \) distributions. You provide a computed test statistic and choose the distribution (and degrees of freedom, if necessary) and CDF prints the marginal significance level of the statistic. Use CDF when the test statistic must be computed in a manner not covered by the other instructions.

In addition to these instructions, you can also report your own test statistics in a more flexible fashion by using the DISPLAY or REPORT instructions. The functions \%ZTEST, \%TTEST, \%FTEST and \%CHISQR compute significance levels for the standard testing distributions.

The TITLE option

All of the instructions listed above have a TITLE option which allows you to include a descriptive title for your test. For example,

```plaintext
exclude(title="Test of Cobb-Douglas as a restriction on translog")
# logl2 logk2 loglk
```
Interpreting the Output

The testing instructions print out their result in a form similar to:

\[ F(4,133) = 2.350870 \text{ with Significance Level 0.05737263} \]

This reports the distribution, the degrees of freedom (if applicable), the test statistic, and the marginal significance level (or P-value). This is the probability that a random variable with the appropriate distribution will exceed the computed value (in absolute value for Normal and t distributions). A P-value close to zero leads you to reject the null hypothesis. For instance, a marginal significance level of .02 would cause you to reject a hypothesis at the .05 level, but not at the .01 level. So, the test statistic above would not quite be significant at the .05 level, because the marginal significance level is .057.

Suppose you are regressing a variable Y (with 100 data points) on a constant term and an explanatory variable X, and you want to test the hypothesis that the slope coefficient is one. One way to do this would be:

```
linreg y
  # constant x
  test(title="Test of Unit Slope")
  # 2
  # 1.0
```

The null hypothesis being tested is that the second coefficient is 1.0. Output such as:

```
Test of Unit Slope
F(1,98)= 0.52842 with Significance Level 0.46900386
```

would be strong evidence in support of the null hypothesis, suggesting that the coefficient on X is indeed one. However, a result such as:

```
F(1,98)= 18.10459 with Significance Level 0.00004789
```

would represent a strong rejection of the null, suggesting that the coefficient is not equal to one.

If you want to do a one-tailed test for a Normal or t-distributed statistic, divide the reported marginal significance level by two.

Non-Standard Test Distributions

While many procedures can be shown to generate test statistics which are, at least asymptotically, distributed as one of the “standard” distributions, it’s becoming increasingly common for tests to have a non-standard distribution. The Durbin-Watson test is a long-standing example of this, but unit root and cointegration tests, and general structural stability tests also in general have distributions for which no known algorithm can generate p-values. For these, there are two main options: you can use a lookup table of key critical values or you can use a bootstrapping or other computationally intensive procedure to approximate a p-value. Most of the RATS
Chapter 6: Hypothesis Testing

procedures for these tests will include critical values in the output. The use of bootstrapping is discussed in Chapter 13.

Note, by the way, that exact $p$-values for several of these tests can, in fact, be computed under the assumption of Normal errors using the functions `%QFORMPDF` or `%QFORMDPDF`. If a test statistic takes the form

$$\varepsilon'\Delta \varepsilon / \varepsilon'B\varepsilon$$

then

$$P(\varepsilon'\Delta \varepsilon / \varepsilon'B\varepsilon \leq x) = P(\varepsilon'(A-\lambda B)\varepsilon \leq 0)$$

This can be computed in RATS by `%QFORMPDF((A-xB),0)`. Examples of the use of this are the Durbin-Watson (exact $p$-values can be computed) and the Dickey-Fuller $\rho$ tests (but not $t$-tests). Note that for fairly large data sets, the calculation of these $p$-values can take a noticeable amount of time. Whether it’s worth it to have an exact finite sample $p$-value under the assumption of Normality rather than an asymptotic one that applies more generally is up to you.

Applicability of the Testing Instructions

The five instructions `RESTRICT`, `MRESTRICT`, `EXCLUDE`, `TEST` and `SUMMARIZE` are all based upon “Wald” test calculations as described in the next section. They can be applied to the results of any of the following “regression” instructions:

`LINREG`, `AR1`, `SUR`, `ITERATE`, `PRBIT`, `LGT`, `DDV`, `LDV`, `RREG`, `PREG`, `RLS`.

You can use any but `EXCLUDE` after `BOXJENK`, `CVMODEL`, `DLM`, `FIND`, `MAXIMIZE`, `NLLS`, `NLSYSTEM`, or `GARCH`. You cannot do tests based directly upon an `ESTIMATE` or `KALMAN`.

For `SUR`, `DLM`, `MAXIMIZE`, `NLSYSTEM`, `CVMODEL`, `PRBIT`, `LGT`, `DDV`, `LDV`, `GARCH`, or for any others where you have used the `ROBUSTERRORS` option, the test statistics are reported as Chi-squared rather than $F$.

Wizards

There are wizards for `EXCLUDE`, `TEST` and `RESTRICT`. All are reached by choosing `Regression Tests` on the `Statistics` menu.

Variables Defined

All testing instructions set values for the following variables:

- `%CDSTAT` the computed test statistic (real)
- `%SIGNIF` the marginal significance level (real).

Which distribution is used depends upon the instruction and the situation.
6.2 Technical Information (Wald Tests)

This section describes the computations used by the five regression-based instructions: `EXCLUDE`, `SUMMARIZE`, `TEST`, `RESTRICT` and `MRESTRICT`. All of these compute the “Wald” test, though many “LM” tests are computed by applying these to auxiliary regressions.

A set of $Q$ linear restrictions on the coefficient matrix $\beta$ can be written

$$R_{Q \times K} \beta = r_{K \times 1}$$

We list below the formulas for the test statistic (used by all five instructions) and the restricted coefficient vector and covariance matrix (for `RESTRICT` and `MRESTRICT` with the `CREATE` option). The first form is for single equation regressions and the second for most non-linear estimation procedures, systems of equations and regressions with corrected covariance matrices.

In these formulas, $(X'X)^{-1}$ may not precisely be that matrix, but will be its appropriate analogue. $\Sigma_x$ is the estimated covariance matrix of coefficients. Either matrix is stored by RATS under the name `\%XX`.

Test statistic

$$F(Q,t-K) = \frac{\left( r - R\hat{\beta} \right)' \left[ R(X'X)^{-1}R' \right]^{-1} \left( r - R\hat{\beta} \right)}{\left( Q\hat{\sigma}^2 \right)}$$

$$\chi^2(Q) = \left( r - R\hat{\beta} \right)' \left[ R\Sigma_x R' \right]^{-1} \left( r - R\hat{\beta} \right)$$

Restricted coefficient vector

$$\hat{\beta}_R = \hat{\beta} + (X'X)^{-1}R \left[ R(X'X)^{-1}R' \right]^{-1} \left( r - R\hat{\beta} \right)$$

$$\hat{\beta}_R = \hat{\beta} + \Sigma_x R' \left[ R\Sigma_x R' \right]^{-1} \left( r - R\hat{\beta} \right)$$

Restricted covariance matrix

$$\text{Var}(\hat{\beta}_R) = \hat{\sigma}^2 \left\{ (X'X)^{-1} - (X'X)^{-1}R \left[ R(X'X)^{-1}R' \right]^{-1} R(X'X)^{-1} \right\}$$

$$\text{Var}(\hat{\beta}_R) = \Sigma_x - \Sigma_x R' \left[ R\Sigma_x R' \right]^{-1} R\Sigma_x$$

If the matrix in [...] is not invertible, RATS will, in effect, drop any redundant restrictions and reduce the degrees of freedom appropriately. If RATS makes such an adjustment, you will see a message like

Redundant Restrictions. Using 4 Degrees, not 8
Chapter 6: Hypothesis Testing

6.3 Technical Information (Lagrange Multiplier Tests)

The Testing “Trinity”

The three main types of tests are the Wald, Likelihood Ratio and Lagrange multiplier, known as the testing trinity. This is covered in just about any modern graduate econometrics text, such as Hayashi (2000), Greene (2003) or Wooldridge (2002). The formulas used in the Wald test were described in the previous section.

Lagrange multiplier tests start by estimating a model under the null hypothesis. The gradient of the (log) likelihood or analogous optimand will be zero in the restricted model itself, and if the null is true, it would be expected to be close to zero in the alternative (expanded) model. The LM tests are often much easier to compute than the other two tests, because they require only that you be able to compute the restricted model, and the gradient (and some approximation to the information matrix) of the expanded model evaluated at the restricted model. Since, in practice, the extra parameters are typically zero under the null, the gradient often simplifies quite a bit when computed under the null.

While the LM tests have a general structure which unites them, they can take a wide variety of forms once you simplify the gradient calculation in a specific case. However, there are two main forms which appear repeatedly: the auxiliary regression and the outer product of the gradient (OPG). Note, by the way, that these two can be different ways of doing the same calculation.

Auxiliary Regression LM Tests

An auxiliary regression is a secondary regression which includes some variables generated from your primary regression, typically a function of the residuals. The actual test is usually either for the regression having a zero $R^2$, or for some regressor or regressors having zero coefficients. The latter type of test can be done by just using an EXCLUDE on the coefficients in question. The $R^2$ tests may be based upon either the centered $R^2$ or the uncentered one. The variable $\%\text{TRSQ}$ returns the number of observations times the uncentered $R^2$, while $\%\text{TRSQUARED}$ will give the number of observations times the centered $R^2$.

For instance, the following is part of example 6.3 in Wooldridge (2002). It does an overidentification test, determining if there is correlation between the residuals from an instrumental variables regression and the list of instruments.

```
instruments constant exper expersq motheduc fatheduc huseduc
linreg(instruments) lwage / resids
# constant exper expersq educ
linreg resids
# constant exper expersq motheduc fatheduc huseduc
cdf(title="Overidentification Test by Auxiliary Regression") $ chisqr %trsq 2
```
Chapter 6: Hypothesis Testing

OPG Tests

The outer product of the gradient form of the test can be used when the gradient can be written as \( \sum g_i \) with sufficient regularity conditions that \( \sum g_i \left( \sum g_i g' \right)^{-1} \sum g_i \) is asymptotically distributed as a \( \chi^2 \).

You will often see this described as being done by a regression of a vector of 1’s on the gradient components. The test statistic is then the sum of squared fitted values from that regression, which is the same (in this case) as the number of observations minus the sum of squared residuals. RATS offers a more convenient way to compute this in most cases. In order to run this (literally) as a regression of 1’s on the gradient components, you need to create series for those components. However, in a wide range of cases, the gradient components will just be the product of residuals (or some other series) with a set of already existing variables. If this is the case, you can compute the test statistic using \texttt{MCOV} with the \texttt{OPGSTAT} option.

\texttt{mcov(opgstat=computed value) / resids}
# gradient variables

The “gradient variables” are the variables which are gradient components when they are multiplied by the \texttt{resids} series. Typically, these are the original regressors together with one or more generated series which are supposed to have a zero correlation with the residuals under the null. Remember that if the original regressors \textit{are} included in this list, the degrees of freedom of the test will be reduced, as those will typically have gradient components which are forced to sum to zero under the null hypothesis.

This is from the same example as on the last page. It does a “robust” overidentification test, determining if there is a correlation between the residuals from an instrumental variables regression and the residuals from regressions of the two potentially endogenous variable on their instruments.

\texttt{linreg motheduc / rmoth}
# constant exper expersq educat
\texttt{linreg fatheduc / rfath}
# constant exper expersq educat
*
\texttt{mcov(opgstat=lmrobust) / resids}
# rmoth rfath
\texttt{cdf(title="Robust Overidentification Test") chisqr lmrobust 2}

Note that you can also use the \texttt{RobustLMTest} procedure for this.
Chapter 6: Hypothesis Testing

6.4 Testing for Heteroscedasticity

The Null Hypothesis

The null hypothesis is

\[ y = X\beta + u \quad E[u] = 0, \quad E[u'u'] = \sigma^2 I \]

There are many alternatives ranging from simple (different variances over different periods) to elaborate (ARCH: autoregressive conditional heteroscedasticity). A number of the testing procedures are regression-based LM tests. The examples here are based upon Verbeek (2004).

Note that the procedure @RegWhiteTest, applied immediately after a regression, can be used to do the Breusch-Pagan or White tests described here.

Breusch-Pagan Test

Breusch and Pagan (1979) describe a Lagrange Multiplier test against the very general alternative

\[ \sigma_i^2 = h(z'_i\alpha) \]

where \( z_i \) is some set of variables, such as regressors. The function \( h \) washes out of the test statistic, so the single test works simultaneously against all alternatives based upon \( z \). We show here a slight modification to Breusch and Pagan, due to Koenker (1981), which does not require an assumption of Normal residuals:

```
linreg labor
# constant wage output capital
set usq = %resids**2
linreg usq
# constant wage output capital
cdf(title="Breusch-Pagan Test") chisqr %trsquared 3
```

Harvey's Test

This is the same idea as Breusch and Pagan, but with the \( h \) function specified to be \( \exp \). The \( \exp \) function is a reasonable functional form in practice, because it produces positive values and, if the “\( z \)” variables are in log form, gives a power function for the variance.

```
set logusq = log(%resids**2)
set logq = log(q)
set logw = log(w)
set logk = log(capital)
linreg logusq
# constant logw logq logk
cdf(title="Harvey Test") chisqr %trsquared 3
```
White’s Test

White (1980) describes a test which has power against alternatives which affect the consistency of the least squares covariance matrix. It is a variation of the Breusch-Pagan test, where $z_t$ consists of the regressors, their squares and products. Because there can be duplication among the regressors and products of regressors, you may need to drop some of those. However, if you do end up with collinearity among the regressors, RATS will simply zero out the redundant variables, and reset the degrees of freedom of the regression. The calculation for the degrees of freedom in the CDF instruction below will give the correct test value if such an adjustment is made.

```r
set logw2 = logw**2
set logq2 = logq**2
set logk2 = logk**2
set logwq = logw*logq
set logwk = logw*logk
set logqk = logq*logk
linreg usq
# constant logw logq logk logw2 logq2 logk2 logwq logwk logqk
cdf(title="White Heteroscedasticity Test") chisqr %trsquared %nobs-%ndf-1
```

Because of the complexity of the setup for this test, we would recommend that you use the procedure `@RegWhiteTest`.

ARCH Test

ARCH (for AutoRegressive Conditional Heteroscedasticity) has been proposed by Engle (1982) as a way to explain why large residuals tend to clump together. The model (for first order ARCH) is

$$ u_i \sim N\left(0, \sigma^2(1 + \alpha u_{i-1}^2)\right) $$

The test is to regress the squared residual series on its lag(s). This should have an $R^2$ of zero under the null hypothesis. This uses a different data set than the first three examples, as testing for ARCH rarely makes sense in a cross section data set like that used before.

```r
diff(center) dlogdm / resids
set usq = resids**2
linreg usq
# constant usq(1)
cdf(title="Test for ARCH(1)") chisqr %trsquared 1
```

Testing for higher order ARCH just requires adding extra lags of USQ and increasing the degrees of freedom on the CDF instruction.

A procedure for doing this test is available on our website, in the file `ARCHTEST.SRC`. 
Chapter 6: Hypothesis Testing

Goldfeld-Quandt Test

The Goldfeld-Quandt test has as its alternative hypothesis that one (identifiable) segment of the sample has a higher variance than another. The test statistic is computed by running the regression over the two subsamples and testing the ratio of the estimated variances.

In a cross section data set, the segments are usually determined by the values of one of the variables. While it's often recommended that you sort the data set on that variable to simplify the process of locating the proper subsamples, using RATS it’s easier to use ORDER with the RANKS option to generate a series of ranks, and to generate SMPL series based upon those.

If you suspect that the variance is related to a continuous variable, breaking the sample into two parts means that each subsample will have some observations close to the break value. This will reduce the power of the test. In this situation, the usual advice is to have a third subsample in the middle which isn’t included in the test.

The following is from Greene (2003, example 11.3). The variable of interest is income, so the ranks for that are generated for the data points which are included in the regressions (those with positive values for expenditure). The sample is split between the first 36 and last 36 data points.

```r
order(ranks=incranks,smpl=posexp) income
linreg(smpl=incranks<=36) avgexp
  # constant age income incomesq ownrent
compute rss1=%rss,ndf1=%ndf
linreg(smpl=incranks>36) avgexp
  # constant age income incomesq ownrent
compute rss2=%rss,ndf2=%ndf
*
cdf(title="Goldfeld-Quandt Test") ftest $
  (rss2/ndf2)/(rss1/ndf1)  ndf2  ndf1
```

Note that if the alternative hypothesis is that the variances are simply different, not that a specific half is greater than the other, then either a very small or a very large $F$ will lead to rejection of the hypothesis. For a 5% two-tailed $F$, reject if the significance level is either less than .025 or greater than .975.

A procedure for doing the Goldfeld-Quandt test, called SWITCH, is available on our website, in the file SWITCH.SRC.
Example 6.1 Testing for Heteroscedasticity

This example tests for heteroscedasticity related to population using several of the techniques in the previous section. The Goldfeld-Quandt test is done using the lowest and highest 22 observations, leaving out the eight in the middle. This is example file HETTEST.PRG.

```plaintext
open data states.wks
data(org=obs,format=wks) 1 50 expend pcaid pop pcinc
set pcexp = expend/pop
linreg pcexp / resids
  # constant pcaid pcinc

  Goldfeld-Quandt Test

order(ranks=popr) pop
linreg(smpl=popr<=22) pcexp
  # constant pcaid pcinc
compute rss1=%rss,ndf1=%ndf
linreg(smpl=popr>=29) pcexp
  # constant pcaid pcinc
compute rss2=%rss,ndf2=%ndf
compute fstat=(rss2/ndf2)/(rss1/ndf1)
cdf(title="Goldfeld-Quandt Test") ftest fstat ndf2 ndf1

  Breusch-Pagan Tests, with POP and log(POP)

set ressqr = resids**2
set logpop = log(pop)
linreg(noprint) ressqr
  # constant pop
cdf(title="Breusch-Pagan Test (Levels)") chisqr %trsquared 1
linreg(noprint) ressqr
  # constant logpop
cdf(title="Breusch-Pagan Test (Logs)") chisqr %trsquared 1

  Harvey test

set logusq = log(ressqr)
linreg(noprint) logusq
  # constant logpop
cdf(title="Harvey Test") chisqr %trsquared 1
```
Chapter 6: Hypothesis Testing

6.5 Serial Correlation

The Hypotheses
In all cases considered, the null hypothesis is absence of serial correlation:

\[ y = X\beta + u \quad E[u] = 0, \quad E[uu'] = D \text{ (diagonal)} \]

D is usually required to be scalar \((\sigma^2 I)\) as well.

Alternatives range from first order autoregressive errors to high order moving average errors.

Standard Tests
RATS instructions such as LINREG and BOXJENK incorporate two tests for serial correlation:

- Durbin-Watson, for first order serial correlation
- Ljung-Box \(Q\), for higher order serial correlation

The Durbin-Watson is reported whether or not it is appropriate for a given model. It is biased towards acceptance of the null when there are lagged dependent variables in the regression, as well as for ARMA models.

This section describes and demonstrates several alternative tests.

Durbin’s \(h\)
The Durbin-Watson test is biased towards a finding of no serial correlation when the model contains a lagged dependent variable. Durbin (1970) has proposed an alternative test for such situations, using the “\(h\)-statistic.” While you can compute \(h\) using RATS, we would strongly urge you to use the regression test shown below (under “General LM Tests”), since it is simpler and more robust.

```
linreg logc
# constant logc{1} logy
compute h=sqrt(%nobs)*%rho/sqrt(1-%nobs*%stderrs(2)**2)
cdf(title="Durbin h test") normal h
```

General LM Tests
Godfrey (1978) and Breusch (1978) have proposed a test against an alternative of serial correlation of order \(N\), in either autoregressive or moving average form. It is a generalization of Durbin’s alternative to the \(h\) test for first order serial correlation.

```
linreg %resids
# constant logc{1} logy %resids{1}
cdf(title="Breusch-Godfrey SC Test") chisqr %trsq 1
```
The Q Test

The Ljung–Box $Q$ (1978) tests against high-order serial correlation. It has low power against specific alternatives, such as first order serial correlation, but can detect the more subtle problems which arise because of a misspecified ARMA model.

**BOXJENK** includes $Q$ tests for a fixed number of lags in the summary statistics. You can also use the **QSTATS** option on **CORRELATE** or **CROSS** to do $Q$ tests on autocorrelations or cross-correlations. For instance, the **CORRELATE** here computes $Q$ tests for lags 1 through 8, 1 through 16, and 1 through 24.

```plaintext
boxjenk(ar=2,ma=1,diffs=1) y / resids
correlate(qstats,number=24,span=8,dfc=3) resids
```

The $Q$ test has a known asymptotic distribution only when applied to the residuals from an ARMA model, where it is asymptotically $\chi^2$ with degrees of freedom equal to the number of correlations minus the number of estimated ARMA parameters. RATS does not correct the degrees of freedom in any other situation. You can include your own correction using the **DFC** option as is done above.

The procedure **REGCORRS** is also available to analyze the autocorrelations of residuals. This can produce either a graph or a table (or both) showing the correlations and $Q$ tests. The syntax of this procedure is

```plaintext
@regcorrs(options) series of residuals
```

where the main options are

- **[graph]/nograph**
- **report/[noreport]**
- **method=yule/[burg]**
- **number=number of autocorrelations to compute**
- **dfc=degrees of freedom correction for the $Q$ statistic**

Cumulated Periodogram Test

This competes with the $Q$ test as a test for general serial correlation. Because it uses frequency domain techniques, the technical details are covered in the description of **CACCUMULATE** in the Reference Manual. However, all you need to do is invoke the procedure **CUMPDGM** to use the test.

```plaintext
boxjenk(ar=2,ma=1,diffs=1) y / resids
@cumpdgm resids
```

**CUMPDGM** reports a Kolmogorov–Smirnov statistic. The K–S test is a general test for whether an empirical distribution comes from a hypothesized distribution. **CUMPDGM** prints (large-sample) critical values for the test.
6.6 Granger-Sims Causality/Exogeneity Tests

Background
Granger (1969) proposed a concept of “causality” based upon prediction error: \( X \) is said to Granger-cause \( Y \) if \( Y \) can be forecast better using past \( Y \) and past \( X \) than just past \( Y \). Sims (1972) proved the following: in the distributed lag regression

\[
Y_t = \sum_{j=-\infty}^{\infty} b_j X_{t-j} + u_t, \quad b_j = 0 \text{ for all } j<0 \text{ if and only if } Y \text{ fails to Granger-cause } X.
\]

“Cause” is a loaded term, and many articles have been written about whether this concept is a proper definition of causality. Regardless, the test is important for several reasons:

- Certain theories predict absence of Granger causality from one variable to another. Such a theory can be rejected if causality is found.
- It is a specification test in distributed lag models such as (1). If the coefficients on future \( X \) are non-zero, then a one-sided \( Y \) on \( X \) regression is a poorly specified dynamic relationship. Also, if the test fails, attempts to correct for serial correlation in estimating a one-sided distributed lag are likely to produce inconsistent estimates.
- Its relationship to prediction is important in building good, small forecasting models.

These tests can be extended to larger collections of variables as restrictions on vector autoregressions. Section 10.4 describes these.

The Procedures
The two basic procedures for causality testing are:

- The “Granger test” regresses \( Y \) on lagged \( Y \) and lagged \( X \) and tests lags of \( X \).
- The “Sims test” regresses \( X \) on past, present and future \( Y \), and tests leads of \( Y \).

While the two testing procedures are equivalent theoretically, they are different in practice, because they must be estimated using finite parameterizations of the autoregression (for Granger) and distributed lag (for Sims), which do not directly correspond.

Geweke, Meese and Dent (1982) examined several forms of causality tests and found that the Sims test was sensitive to failure to correct for serially correlated residuals. They proposed as an alternative a test using a two-sided distributed lag augmented with lagged dependent variables. Although the lag distribution on \( X \) is changed completely by the addition of the lagged dependent variables, the \( X \) coefficients are still one-sided under the null. The example on the next page shows these three tests in action.
Chapter 6: Hypothesis Testing

Example 6.2 Granger-Sims Causality Tests

This example tests whether \( M1 \) is exogenous in a bivariate relationship with \( GDP \). It demonstrates the three testing procedures described in the previous section:

- Sims’ procedure from the 1972 paper: prefilter the variables, compute a two-sided distributed lag of \( GDP \) on \( M1 \) and test the leads of \( M1 \).
- Geweke, Meese and Dent’s recommendation: compute a two-sided distributed lag of \( GDP \) on \( M1 \), including lagged \( GDP \). Test the leads of \( M1 \).
- Granger’s procedure: regress \( M1 \) on lags of \( GDP \) and \( M1 \). Test the lags of \( GDP \).

This is example CAUSAL.PRG

```
cal(m) 1959:1
allocate 2006:4
open data haversample.rat
data(format=rats) / gdph fm1
log gdph
log fm1

Sims’ test

set filtml = fm1 - 1.50*fm1{1} + .5625*fm1{2}
set filtgnp = gdph - 1.50*gdph{1} + .5625*gdph{2}
linreg filtgnp
# constant filtml{-4 to 8}
exclude(title="Sims Causality Test")
# filtml{-4 to -1}

Geweke-Meese-Dent variation

linreg gdph
# constant fm1{-4 to 8} gdph{1 to 8}
exclude(title="Geweke-Meese-Dent Causality Test")
# fm1{-4 to -1}

Granger test

linreg fm1
# constant fm1{1 to 8} gdph{1 to 8}
exclude(title="Granger Causality Test")
# gdph{1 to 8}
```
Chapter 6: Hypothesis Testing

6.7 Structural Stability Tests or Constancy Tests

The Hypotheses

The null hypothesis here is one of structural stability: coefficients (and, typically, the variance) are the same over the sample. The alternatives can range from the precise, with a break at a known point in the sample, to the broad, where the coefficients aren’t the same, but nothing more specific is assumed. If an estimated model fails to show stable coefficients, inference using it, or forecasts generated from it, may be suspect. In general, the precise alternatives will generate standard types of tests, while the broad alternatives generate tests with non-standard distributions. And, if there is, indeed, a rather sharp break, the more precise alternatives will offer higher power. But if there is a general misspecification error which leads to a more gradual change in the coefficients, a test of a hard break might not fare as well.

Tests with a known change point are known generically as “Chow tests”. There are several forms of this, depending upon the assumptions made. These are usually not very hard to implement since they usually require only estimating the model over two or three samples. See the discussion beginning on the next page and Examples 6.3 and 6.4.

The tests with unknown change points usually require a fair bit of number crunching, since every possible break in a range of entries must be examined. In most cases, you should rely upon a RATS procedure (such as STABTEST, for the Hansen stability test) to do these. Some “tests” are actually not formal tests, but rather, are graphical displays of the behavior of some statistic, which are examined informally for evidence of a change in the model. For instance, while the individual tests in a series of sequential “$F$-tests” might have (approximately) $F$ distributions, the maximum of them won’t. These are useful mainly if the resulting statistics leave little room for doubt that the parameters aren’t constant. Even if the asymptotic distribution for the most extreme $F$ isn’t known, if the statistic has a nominal $p$-value of .00001, it’s probably a safe bet that the model has a break.

Tests based upon recursive residuals (see Section 5.14) offer an even more general alternative than the unknown change point. Many tests can be constructed from the recursive residuals because they have the property that, under the null hypothesis, they are independent with constant variance. The behavior of such a process is easy to analyze, so a deviation from that will lead to a rejection of constancy.

Example 6.5 demonstrates some of the ways to test for constancy with a less specific alternative than the known change point.
Chow Tests

The term “Chow test” is typically applied to the test of structural breaks at known locations. Formally, in the model

\[ y_t = X_t \beta_1 + u_t \quad t \in T_1 \]
\[ y_t = X_t \beta_2 + u_t \quad t \in T_2 \]
\[
... \\
\[ y_t = X_t \beta_n + u_t \quad t \in T_n \]

the null is \( \beta_1 = \beta_2 = \ldots = \beta_n \). The general alternative is that this is not true, though it’s possible for the alternative to allow for some of the coefficients to be fixed across subsamples.

There are two ways to compute the test statistic for this:

1. Run regressions over each of the subsamples and over the full sample. The subsample regressions, in total, are the unrestricted “regression”, and the full sample regression is the restricted regression. Compute a standard \( F \)-statistic from the regression summary statistics. With more than two categories, it will probably be simplest to use the `SWEEP` instruction to do the calculations.

2. Run the regression over the full sample, using dummies times regressors for subsamples 2 to \( n \). Test an exclusion restriction on all the dummies.

The first procedure is usually simpler, especially if there are quite a few regressors. However, it is only applicable if the model is estimated appropriately by ordinary least squares. You must use the second method if

- you need to correct the estimates for heteroscedasticity or autocorrelation (by using the `ROBUSTERRORS` option), or,
- you are using some form of instrumental variables, or
- you are allowing some coefficients to be constant across subsamples.

Both of these require that you have enough data points in each partition of the data set to run a regression. An alternative, known as the Chow predictive test, can be used when a subsample (usually at the end of the data set) is too short. In effect, this estimates the model holding back part of the sample, then compares that with the fit when the remainder of the sample is added in.

Examples 6.3 and 6.4 tests a regression for differences between large and small states. Large states are those with a population above 5,000 (in thousands). With this data set (and most cross section data sets), use the option `SMPL` to do the subsample regressions.
Chapter 6: Hypothesis Testing

Example 6.3 Chow Test Using Subsample Regressions

The \texttt{SMPL} option is used to indicate the split between the two subsamples. With time series data sets, you can handle most sample splits with different \texttt{start} and \texttt{end} parameters on \texttt{LINREG}.

This is example file \texttt{CHOW1.PRG}. We use the variables \%RSS and \%NDF to calculate the test statistic. \texttt{RSSLARGE}, \texttt{RSSSMALL} and \texttt{RSSPOOL} are the residual sums of squares of the large state, small state and pooled regressions. Similar variables are defined for the degrees of freedom. The value for the unrestricted regression is the sum for the split samples.

```plaintext
open data states.wks
data(org=col,format=wks) 1 50 expend pcaid pop pcinc
set pcexp = expend/pop
*  
linreg(smpl=pop<5000) pcexp
# constant pcaid pcinc
compute rsssmall=%rss , ndfsmall=%ndf
linreg(smpl=pop>=5000) pcexp
# constant pcaid pcinc
compute rsslarge=%rss , ndflarge=%ndf

Full sample regression

linreg pcexp
# constant pcaid pcinc
compute rsspool=%rss
*
compute rssunr=rsssmall+rsslarge , ndfunr=ndfsmall+ndflarge
compute fstat = ( (rsspool-rssunr)/3 ) / (rssunr/ndfunr)
cdf ftest fstat 3 ndfunr

Using \texttt{SWEEP}. Create a series with different values for each category in the sample split.

set sizes = %if(pop<5000,1,2)

Use \texttt{SWEEP} with the option \texttt{GROUP}=$\text{category series}$. This will do a separate regression on each category. Save the covariance matrix of the residuals (in this case a 1x1 matrix, since there is only the one “target” variable). The sum of squared residuals will be \%NOBS $\times$ the 1,1 element of that matrix. The total number of regressors is in \%NREGSYSTEM.

sweep(group=sizes,cvout=cv)
# pcexp
# constant pcaid pcinc
compute rssunr=cv(1,1)*%nobs
compute ndfunr=%nobs-%nregsystem
compute fstat = ( (rsspool-rssunr)/3 ) / (rssunr/ndfunr)
cdf ftest fstat 3 ndfunr
```

236 RATS User’s Guide
Example 6.4 Chow Test Using Dummy Variables

This method is more generally applicable than the subsample regression method on the previous example, but the cost is a more complex procedure, especially if the number of regressors or number of subsamples is large. This is because a separate dummy must be constructed for each regressor in each subsample beyond the first. Thus, there are $(n-1) \times k$ dummies. This is not much of a problem here, since $n = 2$ and $k = 3$, but had we split the sample four ways, we would need nine SETs. For the more complicated cases, you would probably want to create a VECTOR or RECTANGULAR of SERIES to handle the various interaction terms.

Once the dummies (or more accurately, subsample dummies times regressors) are set up, the procedure is straightforward: estimate the model over the whole sample, including regressors and dummies, and test the joint significance of the dummies.

Because this example uses the ROBUSTERRORS option to correct the covariance matrix for heteroscedasticity, the test statistic will be reported as a $\chi^2$ with three degrees of freedom. Without that option, it will give identical results to the previous example.

This is example file CHOW2.PRG.

```plaintext
open data states.wks
data(org=obs,format=wks) 1 50 expend pcaid pop pcinc
set pcexp = expend/pop
set large = pop>=5000 Will be one for large states

Set up dummies for PCAID and PCINC

set dpcaid = pcaid*large
set dpcinc = pcinc*large

Compute regression with dummies

linreg(robusterrors) pcexp
# constant pcaid pcinc large dpcaid dpcinc

Test dummies

exclude(title="Sample Split Test-Robust Standard Errors")
# large dpcaid dpcinc
```
Example 6.5 General Constancy Tests

The first test here is Bruce Hansen’s (1991) test for general parameter stability, which is a special case of Nyblom’s (1989) stability test. This is based upon the behavior of partial sums of the regression’s normal equations for the parameter and variance. The STABTEST procedure generates test statistics for the overall regression (testing the joint constancy of the coefficients and the variance), as well as testing each coefficient and the variance individually. It also supplies approximate p-values.

The next test is a Chow predictive test. The regression is run over the sample through 1971:3, then again through 1973:3. The difference between the sums of squares divided by the number of added data points (8) forms (under the null) an estimate of the variance of the regression that’s independent of the one formed from the first subsample, and thus it generates an F. Note that, in this case, there is enough data to do a separate regression on the second subsample. However, it’s a very short subsample, and the standard Chow test would likely have relatively little power as a result.

The remainder of the example works off statistics generated from recursive estimation. The first graph is that of the recursive residuals with the (recursively estimated) standard error bands. This doesn’t form a formal test; however, if there is a break, it’s likely that the residuals will, for a time, lie outside the bands until the coefficients or variance estimates adjust. The second is a more formal CUSUM test (Brown, Durbin and Evans (1975)). Under the null, the cumulated sums of the recursive residuals should act like a random walk. If there is a structural break, they will tend to drift above the bounding lines, which here are set for the .05 level.

The final graph is a set of one-step Chow predictive F-tests. This is basically the same information as the recursive residuals graph with a different presentation. Again, this is an informal test. This graph is designed to present not the sequential F’s themselves, but the F’s scaled by the .05 critical value. At the start of the sample, the F’s are based upon very few denominator degrees of freedom, so F’s that are quite large may very well be insignificant. Anything above the “1” line (shown on the graph with the help of the VGRID option) is, individually, statistically significant at the .05 level.

This is example file CONSTANT.PRG. It’s based upon an example from Johnston and DiNardo (1997).

```
open data auto1.asc
cal(q) 1959:1
data(format=prn.org=columns) 1959:1 1973:3 x2 x3 y

The StabTest procedure does the Hansen test for parameter instability

@stabtest y 1959:1 1971:3
# constant x2 x3
```
Chapter 6: Hypothesis Testing

Chow predictive test over the next two years after 1971:3

```
compute rss1=%rss,ndf1=%ndf
linreg(noprint) y 1959:1 1973:3
  # constant x2 x3
compute f=((%rss-rss1)/8)/(rss1/ndf1)
cdf(title="Chow Predictive Test") ftest f 8 ndf1
```

Tests and graphs based upon recursive estimation

```
rls(sehist=sehist,cohist=cohist,sighist=sighist,$
  csum=cusum,csquared=cusumsq) y 1959:1 1973:3 rresids
  # constant x2 x3

RSTART is the first observation without a perfect fit (In general, you could get this by %REGSTART()+%NREG).

compute rstart=1959:4

This graphs the recursive residuals with the upper and lower two (recursively generated) standard error bands.

```
set lower = -2*sighist
set upper = 2*sighist
graph(header="Recursive Residuals and Standard Error Bands") 3
  # rresids
  # lower
  # upper / 2
```

CUSUM test with upper and lower bounds

```
set cusum = cusum/sqrt(%seesq)
set upper5 rstart 1973:3 = .948*sqrt(%ndf)*$
  (1+2.0*(t-rstart+1)/%ndf)
set lower5 rstart 1973:3 = -upper5
graph(header="CUSUM test") 3
  # cusum
  # upper5
  # lower5 / 2
```

Sequential F-Tests. These are generated quite easily using the cumulated sum of squares from RLS.

```
set seqf = (t-rstart)*(cusumsq-cusumsq{1})/cusumsq{1}
set seqfcval rstart+1 * = seqf/%invftest(.05,1,t-rstart)
graph(vgrid=||1.0||,$
  header="Sequential F-Tests as Ratio to .05 Critical Value")
  # seqfcval
```
6.8 Functional Form Tests

Introduction
This section describes tests which are designed to determine if the functional form for a regression equation is adequate. For instance, should an equation be estimated in logs or levels? Should it be linear in the variables, or should a higher power, like a quadratic, be used? Note that some of these procedures are “non-nested”, and so can result in rejecting both of the possibilities, or accepting both.

RESET Tests
RESET stands for Regression Equation Specification Error Test. These were proposed by Ramsey (1969). If the regression equation

\[ y_t = X_t \beta + u_t \]

is specified correctly, then adding non-linear functions (and, in particular, powers) of the fitted values \( \hat{X}_t \beta \) should not improve the fit significantly. This is designed to test whether in

\[ y_t = f(\hat{X}_t \beta) + u_t \]

the “\( f \)” function is just \( f(x) = x \). However, it can pick up other types of failures.

You can get the fitted values using the instruction \texttt{PRJ}. The test is carried out by generating the needed powers of the fitted values, and running an auxiliary regression with the original specification plus the powers. Test an exclusion on the higher powers. The following (from an example in Verbeek (2004)), does a cubic RESET.

\begin{verbatim}
linreg lprice
# constant llot bedrooms bathrms airco
prj fitted
set fit2 = fitted**2
set fit3 = fitted**3
linreg lprice
# constant llot bedrooms bathrms airco fit2 fit3
exclude(title="RESET test with quadratic and cubic")
# fit2 fit3
\end{verbatim}

In practice, you’ll find it easier to use the \texttt{@RegRESET(h=power)} procedure. Use this immediately after the regression that you want to test:

\begin{verbatim}
linreg lprice
# constant llot bedrooms bathrms airco
@RegRESET(h=3)
\end{verbatim}
MacKinnon-White-Davidson Test

Davidson and MacKinnon (1981) and MacKinnon, White and Davidson (1983) proposed a method for testing the functional form for the dependent variable. The most common use of this would be for determining whether levels or log is more appropriate. This is a non-nested test, so it’s possible to reject each in favor of the other, or to fail to reject either. The test procedure is to add the difference between the levels fit and the exponentiated log fit to the log equation, and to add the difference between the log fit and the log of the levels fit to the levels equation. The added coefficient should be insignificant if the original equation was properly specified. You can just look at the $t$-statistic on it.

The following is taken from an example in Greene (2003):

```
linreg m1  # constant tbilrate realgdp
prj linearfit
set logml = log(m1)
set logy  = log(realgdp)
set logr  = log(tbilrate)
linreg logml  # constant logr logy
prj logfit *
set loggap = logfit-log(linearfit)
set lingap = linearfit-exp(logfit)
linreg logml  # constant logr logy lingap
linreg m1  # constant tbilrate realgdp loggap
```
6.9 Specification Tests

Introduction

*Specification test* is a very general term which really can cover many of the tests described in previous sections. The one absolute requirement for performing a specification test is this:

*The model must incorporate more assumptions than are required to estimate its free coefficients.*

For instance, if our model, in total, consists of

\[
 y_i = X_i \beta + u_i \quad E(X_i u_i) = 0, \quad \beta \text{ unknown}
\]

we have nothing to test, because the least squares residuals will be, by construction, uncorrelated with the $X$'s. We get specification tests when we go beyond these assumptions:

- If we assume, in addition, that the $u$'s are homoscedastic or serially uncorrelated, we can test those additional assumptions, because we don’t use them in estimating $\beta$ by least squares (though we may rely on them for computing the covariance matrix).

- If we assume, in addition, that $E(u_i | X_i) = 0$, we can test whether various other functions of X are uncorrelated with the residuals. The RESET test is an example of this.

- If we assume, in addition, that $E(X_i u_i) = 0$ for all $t,s$ (strict econometric exogeneity), we get, in the distributed lag case, Sims’ exogeneity test, which tests whether or not $u_i$ is uncorrelated with $X_{t+j}$.

Two basic strategies are available:

- Compare estimates of $\beta$ computed with and without the additional assumptions. If the additional assumptions are true, the estimates should be similar. This gives rise to “Hausman tests” (Hausman (1978)).

- If the assumptions can be written as orthogonality conditions ($u_i$ uncorrelated with a set of variables), seeing whether or not these hold in sample is a test of the overidentifying restrictions. The general result for this style of test is given by Hansen (1982).
Hausman Tests

Hausman tests operate by comparing two estimates of $\beta$, one computed making use of all the assumptions, another using more limited information. If the model is correctly specified, $\hat{\beta}_1 - \hat{\beta}_2$ should be close to zero. The difficult part of this, in general, is that the covariance matrix of $\hat{\beta}_1 - \hat{\beta}_2$ is not the covariance matrix of either one alone, it is

$$
(1) \quad \text{Var}(\hat{\beta}_1) + \text{Var}(\hat{\beta}_2) - 2 \text{Cov}(\hat{\beta}_1, \hat{\beta}_2)
$$

and computing the covariance term is quite unappealing. However, in Hausman’s settings (1978), the covariance matrix of the difference simplifies to

$$
(2) \quad \text{Var}(\hat{\beta}_1) - \text{Var}(\hat{\beta}_2)
$$

where $\hat{\beta}_1$ is the less efficient estimator. This result is valid only when $\hat{\beta}_2$ is efficient and $\hat{\beta}_1$ is based upon the same or a smaller information set.

Procedure

To perform a Hausman test by the direct comparison of estimators, follow these steps:

1. Estimate the model using one of the two estimators. Save its coefficient vector (%BETA) and covariance matrix (%XX) into other matrices.
2. Estimate the model using the second estimator.
3. Use TEST with the options ALL and VECTOR=saved coefficients and COVMAT=difference in covariance. Replace %XX with the difference between the less efficient covariance matrix and the more efficient one. This tests for equality between the two estimated coefficient vectors. The covariance matrix of the difference will probably not be full rank. TEST will determine the proper degrees of freedom.

You may have to do a bit of work to come up with truly comparable estimates of the covariance matrix. For instance, you want to scale the covariance matrices by the same estimate of $\sigma^2$. 
Example

To look at a simple example, suppose

\[
(3) \quad c_i = \alpha_0 + \alpha_1 y_i + u_i
\]

\[
(4) \quad E(u_i) = E(c_{i-1} u_i) = E(y_{i-1} u_i) = 0
\]

\[
(5) \quad E(y_i u_i) = 0
\]

If we take (3) and (4) as maintained hypotheses, we can estimate the equation consistently by instrumental variables. If (5) is also valid, OLS will be efficient. The Hausman test is the following:

```r
linreg realcons 1950:2 *
# constant realgdp
compute xxols=%xx, betaols=%beta, seesqols=%sigmasq
instruments constant realgdp{1} realcons{1}
linreg(inst) realcons 1950:2 *
# constant realgdp
*
compute cvdiff=seesqols*(%xx-xxols)
test(title="Hausman Test", all, form=chisquared, $
   vector=betaols, covmat=cvdiff)
```

In many cases (such as this one), the Hausman test can be computed more easily with an auxiliary regression.

```r
linreg realgdp
# constant realgdp{1} realcons{1}
prj yfit
linreg realcons
# constant realgdp yfit
exclude(title="Wu test")
# yfit
```

For this specific application (testing whether the explanatory variables in a regression are exogenous), you can use the `@RegWuTest` procedure immediately after running the regression. (It uses the current instruments list in constructing the alternative).

```r
instruments constant realgdp{1} realcons{1}
linreg realcons 1950:2 *
# constant realgdp
@RegWuTest
```
“Hansen” Tests

By limiting ourselves to a notation appropriate for those models which can be estimated using single-equation methods, we can demonstrate more easily how these tests work. Assume

\[ y_t = f(X_t, \beta) + u_t \]

\[ E(Z_t' u_t) = 0 \]

with some required regularity conditions on differentiability and moments of the processes. \( f \) will just be \( X_t \beta \) for a linear regression. The \( Z \)'s are the instruments.

If \( Z \) is the same dimension as \( \beta \), the model is just identified, and we can test nothing. Hansen’s key testing result is that

\[ u'ZWZ'u \sim \chi^2 \]

when the weighting matrix \( W \) is chosen “optimally.” The degrees of freedom is the difference between the number of orthogonality conditions in (7) and the number of parameters in the \( \beta \) vector.

This test is automatically included in the output from LINREG, AR1, NLLS, NLSYSTEM and SUR as the J-Specification. This generates output as shown below, with the first line showing the degrees of freedom (the difference between the number of conditions in (7) and the number of estimated coefficients) and the test statistic, and the second line showing the marginal significance level:

<table>
<thead>
<tr>
<th>J-Specification(4)</th>
<th>7.100744</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance Level of J</td>
<td>0.13065919</td>
</tr>
</tbody>
</table>

The computed test statistic can be obtained as %JSTAT, its significance level as %JSIGNIF and the degrees of freedom as %JDF. \( u'ZWZ'u \) is available as %UZWZU.

If the weighting matrix used in estimation isn’t the optimal choice, there are two ways to “robustify” the test. The first is described in Hansen (1982)—it computes an adjusted covariance matrix \( A \) for \( Z'u \) so that \( u'ZA^{-1}Z'u \sim \chi^2 \). The other is described in Jagannathan and Wang (1996). It uses \( u'ZWZ'u \) as the test statistic, but they show that it has an asymptotic distribution which is that of a more general quadratic form in independent Normals. You can choose between these with the option JROBUST=STATISTIC (for the first) or JROBUST=DISTRIBUTION (for the second).
Chapter 6: Hypothesis Testing

Example 6.6 Hansen’s Specification Test

This does two Hansen tests for the consumption equation in Klein’s model I. The first is the simplified version for two-stage least squares. The second uses the GMM optimal weights (see Section 7.8) allowing for one lag of autocorrelation. This is example file HANSEN.PRG.

cal(a) 1920
allocate 1941:1
open data klein.prn
data(org=columns,format=prn) / cons $
  profit privwage invst klagged1 prod govtwage govtexp taxes
set wagebill = privwage+govtwage
set trend = t-1931:1
set capital = klagged1+invst
smpl 1921:1 1941:1

instruments constant trend govtwage govtexp taxes $
  profit{1} capital{1}  prod{1}

Simple 2SLS. The test statistic will be included in the output.

linreg(inst) cons / resids
# constant profit{0 1} wagebill

Doing the test by regressing residuals on the full instrument set.

linreg resids
# constant trend govtwage govtexp taxes $
  profit{1} capital{1}  prod{1}
cdf chisqr %trsq 4

The two test statistics are slightly different because the J-statistic produced by the first LINREG uses a degrees of freedom corrected estimate of the residual variance. To get the identical result, you need to multiply %TRSQ by (%NOBS–4.0)/%NOBS

Test with the weight matrix adjusted for serial correlation and heteroscedasticity. The test statistic here can’t easily be computed except by using some set of options for LINREG:

linreg(inst,optimal,lags=1,lwindow=newey) cons
# constant profit{0 1} wagebill

cdf chisq %uzwzu 4
Example 6.7 Hausman’s Specification Test

This example (HAUSMAN.PRG) applies a Hausman test to Klein’s Model I. SUR generates 3SLS and 2SLS estimates (the latter using CV=identity matrix).

The covariance matrix of the 2SLS estimates computed using SUR is incorrect because the variances do not appear in the calculation. The true covariance matrix that allows for the correlations among residuals is:

\[
\left( X' \left( I \otimes Z(Z'Z)^{-1}Z' \right) X \right)^{-1} \left( X' \left( S \otimes Z(Z'Z)^{-1}Z' \right) X \right) \left( X' \left( I \otimes Z(Z'Z)^{-1}Z' \right) X \right)^{-1}
\]

In this formula, \( X \) is the (stacked) matrix of explanatory variables and \( Z \) the matrix of instruments. The matrix on each end is the \( %XX \) produced by \texttt{SUR(INST)} with \( CV=\text{identity matrix} \). The center matrix can be computed using an application of \texttt{SUR(INST)} with \( CV=S^{-1} \). The coefficients from this last \texttt{SUR} are meaningless; we use it only for the version of \( %XX \) that it computes.

Note that you could also use

\texttt{sur(model=klein,inst,cv=\%identity(3),robusterrors)}

to get a corrected estimate of the covariance matrix for the system 2SLS estimator, skipping the extra step described above, but that won’t match up with the 3SLS the way that is needed for a (small-sample) Hausman test. It’s only when both 3SLS and the 2SLS covariance matrix are computed using a common covariance matrix for the residuals that you’re guaranteed that the difference in the covariance matrices will be positive semi-definite.
Chapter 6: Hypothesis Testing

cal(a) 1920
allocate 1941:1
open data klein.dat
data(org=obs,format="(7x,9f8.3)") / consumption $
   profit privwage invest klagged production govtwage govtexp taxes
set wagebill = privwage+govtwage
set trend    = t-1931:1
set capital  = klagged+invest
smpl 1921:1 1941:1
instruments constant trend govtwage govtexp taxes profit{1} $
   capital{1}  production{1}
*
equation   conseq consumption
# constant  profit{0 1}   wagebill
equation   invseq invest
# constant  profit{0 1}   capital{1}
equation   wageeq privwage
# constant  production{0 1}  trend
*
group klein conseq invseq wageeq

Compute 2SLS estimates using SUR. Save the covariance matrix of residuals in
VCV2SLS, the coefficients and %XX matrix in BETA2SLS and COV2SLS.
sur(model=klein,inst,cv=%identity(3),cvout=vcv2sls)
compute [symmetric] cov2sls=%xx
compute [vector] beta2sls=%beta

    Compute covariance matrix of 2SLS. Apply SUR with CV=INV(VCV2SLS), then
    compute the matrix expression from the preceding page.
sur(model=klein,inst,nosigma,cv=inv(vcv2sls),noprint)
compute cov2sls=%mqform(inv(%xx),cov2sls)

    Compute 3SLS estimates.
sur(model=klein,inst)
*
test(all,vector=beta2sls,covmat=cov2sls-%xx)
6.10 Unit Root Tests

Much of applied macroeconometrics in the past twenty years has been devoted to an analysis of the implications of “unit root” and related behaviors in time series data. The econometrics governing the tests is technically difficult, with a large body of published papers discussing the proper methods. Because of this, we have chosen not to code specific tests directly into RATS. Instead, we rely upon procedures, which can more easily be adapted to changes in current practice. We will discuss here procedures to do the basic forms of the tests. On the Estima web site (www.estima.com), there are many procedures for unit root testing which are either based upon the basic tests, but include more options, or which provide different testing methods.

Classical Tests

There are two principal “classical” testing methodologies: Dickey– Fuller (from Fuller (1976) and Dickey and Fuller (1979)) and Phillips–Perron (from Phillips (1987) and Phillips and Perron (1988)). The difference between them is their treatment of any “nuisance” serial correlation aside from that generated by the hypothesized unit root. Each tends to exhibit rather poor behavior in the presence of certain types of serial correlation. See the Monte Carlo analysis in Schwert (1989).

We have written procedures called DFUNIT and PPUNIT which do the Dickey– Fuller and Phillips–Perron tests, respectively, and are supplied on the files DFUNIT.SRC and PPUNIT.SRC. They have similar syntax:

```
@dfunit( options ) series start end
@ppunit( options ) series start end
```

- `series` Series to be tested for unit roots.
- `start end` Range to use in testing. By default, the full range of the series.

- `det=none/[constant]/trend`
  This chooses the deterministic parts to include.

- `ttest]/nottest`
  Each testing procedure has two variants, one based upon the regression t statistic, the other on \( T(\rho - 1) \). The former is the one most commonly used, and is the default.

- `lags=number of additional lags [0] (DFUNIT)`
  With a positive value for LAGS, DFUNIT performs an “augmented Dickey– Fuller test”: testing for a unit root in an AR(1+LAGS) representation. It has the same asymptotic distribution under the null as the standard Dickey– Fuller test does in the AR(1) case. Note that some authors describe the number of lags in a Dickey–
Chapter 6: Hypothesis Testing

Fuller test as the number of lags in the AR, not (as is done here) by the number of additional lags. If that’s the case, their "3" lag test would be done using LAGS=2.

\( \text{lags=number of covariance lags \[4\]} \) (PPUNIT)

In PPUNIT, LAGS is the number of lags to include in the calculation of \( \text{var } Y \) (in Phillips' notation).

We have implemented the Phillips–Perron test as suggested by Phillips, using the weighted covariance estimator to compute \( \sigma^2 \) (in his notation). This is calculated using an MCOV instruction.

Examples

One of the examples worked out by Hamilton (1994) is for the U.S. 3-month Treasury bill rate. The following computes several of the tests used in Hamilton’s examples. The test statistics all have a non-standard distribution, and their distribution depends upon the set of other explanatory variables included (constant, trend, none) and upon whether you choose the \( T(\rho - 1) \) or t-test form. If you choose the t-test form, the procedures include approximate critical values, computed using the formulas from MacKinnon (1991). If you use the \( T(\rho - 1) \) form, make sure you choose the correct table to look up the critical values. The output produced by the first two tests is provided below.

```
cal(q) 1947
open data gnptbill.txt
data(format=prn,org=obs) 1947:1 1989:1
@dfunit tbill
@ppunit(lags=4) tbill
@dfunit(lags=4,nottest) tbill

Dickey-Fuller Unit Root Test, Series TBILL
Regression Run From 1947:02 to 1989:01
Observations 169
With intercept with 0 lags on the differences
T-test statistic      -1.72945
Critical values: 1%= -3.470 5%= -2.879 10%= -2.576

Phillips-Perron Unit Root Test, Series TBILL
Regression Run From 1947:02 to 1989:01
Observations 169
With intercept , with 4 lags in the error process
T-test statistic      -1.80563
Critical values: 1%= -3.470 5%= -2.879 10%= -2.576
```
Bayesian Tests

A third procedure implements the Bayesian odds ratio test proposed by Sims (1988), with a slight correction: Sims’ formula is missing a \( \log 2\pi \) term. This is the procedure \texttt{BAYESTST}.

\[
\text{@bayestst ( options ) series start end}
\]

- \texttt{series} Series to be tested for unit roots.
- \texttt{start end} Range to use in testing. By default, the full range of the series.

\[\text{limit=lower limit for stationary prior} [.5]\]

The stationary part of the prior for \( \rho \) is flat on the interval \((\text{lower limit}, 1.0)\).

\[\text{alpha=prior probability on the stationary values of} \; \rho [.8]\]

This is the probability on the stationary part of the prior. The remaining probability is concentrated on \( \rho=1 \).

The following is an example from the output from \texttt{BAYESTST}. The first is the \( t^2 \) which is used as the test statistic. The “Schwarz Limit” and “Small Sample Limit” are the asymptotic and small sample Bayesian “critical values” for the test statistic. The Schwarz Limit washes out any terms which become negligible asymptotically. The Small Sample value depends upon your choices for \texttt{ALPHA} and \texttt{LIMIT}. The “Marginal Alpha” is the value for \texttt{ALPHA} at which the posterior odds for and against the unit root are even. A small value indicates that the data evidence is strongly against the unit-root hypothesis, as you would have to put most of the prior probability on the unit root just to make the posterior odds even.

In this example, the test statistic falls between the Schwarz limit and the small sample limit. Thus the test fails to reject the unit–root hypothesis using a large sample approach, but does reject the hypothesis for the precise values of \texttt{ALPHA} and \texttt{LIMIT} which were chosen.

<table>
<thead>
<tr>
<th>Squared t</th>
<th>Schwarz Limit</th>
<th>Small Sample Limit</th>
<th>Marginal Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.921</td>
<td>12.658</td>
<td>6.661</td>
<td>0.6806</td>
</tr>
</tbody>
</table>
Cointegration Tests

If we take two series, each of which has a unit root, then, in general, any linear combination of them will also have a unit root. However, it is possible for there to exist a linear combination which is stationary instead. Such series are said to be cointegrated (Engle and Granger (1987)). The textbooks by Hamilton (1994) and Enders (2003) are good references on the subject. Of these, Hamilton is the more theoretical; Enders the more applied.

Cointegration is a property of two or more series. Some examples where cointegration might possibly be seen are

1. $\log Y_t - \log C_t$ \hspace{1cm} consumption proportional to income
2. $\log P_t - \log S_t - \log P^*$ \hspace{1cm} purchasing power parity ($P$'s are prices, $S$ exchange rate)

In neither case would we expect these linear combinations to be exactly constant. However, if the variables are cointegrated, we would expect these residuals to stay close to a fixed value if observed over a long span of time. If there were still a unit root in the residuals, we would expect to see them wandering farther away as the sample size increased.

Note that before a set of series can be cointegrated, they must be (individually) integrated (I(1)). This rather important point has been missed by people trying to get results quickly, but not carefully. If one of the input series is stationary, rather than I(1), you will almost certainly find “cointegration”, because $1 \times$ the stationary series + $0 \times$ the others is a stationary linear combination. Note that there is nothing wrong with including both integrated and stationary variables in an analysis, you just have to understand how they interact when you look at cointegrating rank statistics.

Just as with unit root tests, there are quite a few cointegration testing procedures available on the Estima web site (www.estima.com). As with unit root tests, we are somewhat skeptical of the value of much of the empirical work done on cointegration. For analyzing joint behavior of a set of time series in greater detail, we would recommend the program CATS by Dennis, Hansen and Juselius (available separately from Estima). CATS adopts a structured approach to cointegration, emphasizing the testing of specific restrictions.

Most of the testing procedures are based upon a two-step procedure of estimating a cointegrating vector and then performing a stationarity test on the residuals. The result underlying this is that if two variables $X_{1t}$ and $X_{2t}$ are cointegrated, then if the regression

\[ X_{1t} = \alpha X_{2t} + u_t \]

is run, the estimate of $\alpha$ is not only consistent (even though it disregards possible simultaneous equations bias), but is “superconsistent,” converging to the true value at a faster rate than for a typical regression coefficient (Stock (1987)). But what happens if they aren’t cointegrated? Then the estimated coefficient in (3) is likely to be garbage; this is the “spurious regression” of Granger and Newbold (1974). It’s one
thing to test a restriction like (1) and (2) that is rooted in economic theory. It’s quite another to blindly estimate a “cointegrating vector” and to rely upon asymptotic distribution theory to save us from an incorrect inference. If, for instance, we were to estimate the coefficients in (2) and came up with

\[(4) \quad \log P_t-.05 \log S_t-.50 \log P_t^*\]

and we were able to reject non-cointegration, would we seriously believe that (4) represents some type of long-term equilibrium condition? We would be on much safer ground reaching the conclusion that this was due to sampling error. After all, we are attempting to infer very long-run behavior of a set of series from a finite sample.

**Testing Procedures**

The simplest testing procedure for a known cointegrating vector is simply a standard unit root test applied to the residual from the hypothesized linear combination of the variables. The unit root test will have the same asymptotic distribution that it has when applied to any other observed series.

```
set ppp = uscpi-exrat-italcpi
@dfuncit(lags=12) ppp
```

The simplest testing procedure for cointegration with an unknown cointegrating vector is to apply a unit root test to the residuals from a regression involving the variables. This is the Engle-Granger test. Since the sum of squares of a non-stationary linear combination should be quite a bit higher than those for a stationary linear combination, we would expect that least squares would zero in on a stationary linear combination if it exists. For that reason, it’s even more important in this case to make sure the input variables are themselves I(1). Because the coefficients are now estimated, the critical values for the unit root test are different and get more negative the more variables we include in the cointegrating regression. There are two procedures for doing this: `@EGTestResids` takes as input the residuals from the cointegrating regression. `@EGTest` takes the set of variables and does the preliminary regression itself.

```
@egtest(lags=12)
# uscpi italcpi exrat
```

An alternative to the regression-based tests is the likelihood ratio approach, which is the basis for the CATS software. The likelihood approach allows for testing sequentially for the rank of cointegration from 0 (no cointegration, \(N\) separate stochastic trends) up to \(N\), which would mean no unit roots. See the discussion in Hamilton.

The procedure `@JOHMLE` does the basic Johansen likelihood ratio test.

```
@johmle(lags=6, det=constant)
# uscpi italcpi exrat
```
Chapter 6: Hypothesis Testing

Example 6.8 Cointegration Testing Procedures

The following example is taken from Hamilton, and can be found on the file COINTTST.PRG. Hamilton tests whether the Purchasing Power Parity (PPP) restriction (2) is a cointegrating vector for Italian and U.S. data.

After testing the individual series for unit roots (all pass), the proposed “equilibrium” condition is then tested. If the variables are cointegrated and we have used the correct cointegrating vector, then this series should fail a unit root test. Because the final Dickey-Fuller test accepts the unit root, we reject the null hypothesis of cointegration—the deviations from PPP are not stationary. The graph of deviations (below) would tend to confirm that.

The example also includes an Engle-Granger and Johansen Likelihood Ratio test.

```plaintext
cal(m) 1973:1
open data exdata.rat
data(format=rats) 1973:1 1989:10 pc6it pzunew exritl
set italcpi = 100*log(pc6it.pc6it(1973:1))
set uscpi = 100*log(pzunew.pzunew(1973:1))
set exrat = -100*log(exritl.exritl(1973:1))
@dfunit(lags=12,trend) uscpi
@dfunit(lags=12,trend) italcpi
@dfunit(lags=12,trend) exrat
set ppp = uscpi-exrat-italcpi
graph(header="Figure 19.3 The real dollar-lira exchange rate") # ppp
@dfunit(lags=12) ppp *
@egtest(lags=12) # uscpi italcpi exrat *
@johmle(lags=6,det=constant) # uscpi italcpi exrat
```

![Graph showing deviations from PPP over time.](image)
Chapter 7
Non-Linear Estimation

RATS offers a range of instructions for estimating non-linear models. Which should be used depends upon the underlying assumptions. The first half of the chapter describes the elements that the estimation instructions have in common: the optimization techniques and the methods of describing the model. The second half demonstrates the instructions themselves.

See also Chapter 12, which describes several model types which require the techniques described here, such as Markov switching models.
Machine Precision

There are certain problems with which all non-linear estimation techniques in RATS must deal. One difficulty is that, on a computer, floating point numbers are represented only approximately. Most systems running RATS show fourteen to fifteen significant digits. This does not mean, however, that you can get answers from estimation problems correct to fourteen digits. Intermediate calculations can often push the significance limit long before that. For instance, \( f(x) = 50 + (x - 1)^2 \) will give identical values for \( x = 1 \) and \( x = 1.00000001 \). Because the computer can't distinguish between the function value for these two (or anything in between), we can't get greater accuracy than eight digits in minimizing \( f \).

A related problem is that floating point numbers have a huge, but limited, range, typically with around \( 10^{300} \) as the largest number and \( 10^{-300} \) as the smallest positive number. This can cause problems if a calculation is done in such a way that intermediate calculations overflow the range.

If, for instance, you attempt to calculate the binomial coefficient \( \binom{200}{100} \) using the expression \( \%\text{FACTORIAL}(200)/\%\text{FACTORIAL}(100)^2 \), RATS will produce a missing value, as \( \%\text{FACTORIAL}(200) \) overflows. As these difficulties typically arise where a logarithm of the big number is needed, the way to work around this is to make sure you take logs early on and add or subtract them as needed, rather than risking overflow by multiplying and dividing huge numbers. Here the log of the binomial coefficient can be computed by \( \%\text{LNGAMMA}(201) - 2.0 \times \%\text{LNGAMMA}(101) \). For normal densities, use \( \%\text{LOGDENSITY} \) rather than \( \%\text{DENSITY} \) if you'll eventually be taking logs anyway.

Convergence

A non-linear optimization is considered “converged” when the change from one iteration to the next is “small.” RATS defines the change in a vector of parameters by taking the maximum across its elements of

\[
\min(|\beta - \beta_0| / |\beta_0|, |\beta - \beta_0|)
\]

where \( \beta_0 \) is the parameter’s value before this iteration. This uses the relative change for parameters which, on the previous iteration, had absolute values of 1 or larger, and absolute changes for smaller values. This is compared with the convergence criterion that you set (typically by the CVCRIT option). The default value of this is 0.00001 for most methods. A value much smaller than that (say \( 10^{-8} \)) probably is unattainable given the precision of floating point calculations on most machines, and even if the estimation succeeds in meeting the tighter value, you are unlikely to see much effect on the final coefficients.

However, you need to be careful how you parameterize your function. In particular, if you have a parameter which has a natural scale which is quite small (say .0001),
then a change of .000009 might have a substantial effect on the function value, but would be small enough to pass the convergence test. Rescaling the parameter (replacing it wherever it appears with .001 times itself, or scaling a variable it multiplies by .001) can help here. This also helps with the accuracy of numerical derivatives if RATS needs to take them.

RATS does allow you to set the optimizers so that they consider convergence achieved if the change in the function value is small. This is done using the instruction \texttt{NLPAR} with the \texttt{CRITERION=VALUE} option. Convergence on function value is usually quite a bit easier to achieve than convergence on coefficients. You should probably choose this only when just the optimized function value matters, and the coefficients themselves are of minor importance.

Local Optima and Initial Guesses

With the exception of the “genetic” algorithm, all optimization methods used in RATS will generally end up converging to the local optimum which is “uphill” from your initial guesses. For most functions, there is no guarantee that this will be the global optimum. If you have any doubts about whether your function might have multiple local optima, it’s a good idea to try re-estimating from several extra sets of initial guesses, to see if you converge just to the one set of estimates.

The genetic algorithm (described in Section 7.3) scans more broadly. However, in a large problem the cost, in compute time, can be enormous.

Tracing Progress

Most non-linear estimation instructions include an option \texttt{TRACE}, which produces intermediate output showing the progress of the iterations. The form of this varies depending upon the estimation technique being used, but this is fairly common:

\texttt{Non-Linear Optimization, Iteration 6. Function Calls 39.}
\texttt{Cosine of Angle between Direction and Gradient 0.0116839. Alpha used was 0.00}
\texttt{Adjusted squared norm of gradient 0.534343}
\texttt{Diagnostic measure (0=perfect) 1.4860}
\texttt{Subiterations 1. Distance scale 1.000000000}
\texttt{Old Function = -250.152278 New Function = -250.105608}
\texttt{New Coefficients:}
\texttt{1.876483 14.467947 -0.175269 0.604792}

\texttt{TRACE} can be helpful in seeing what is going on with your estimation. If the function value seems to stabilize, but the squared norm of the gradient doesn’t go to zero, and the diagnostic measure stays high (above 2), the model may be only barely identified, with a “ridge” rather than a peak in the surface. Very slow progress at the start is often a sign of poor initial guess values, possibly very near a zone where the function is either explosive or simply uncomputable (due to things like logs of negative numbers). This is a good sign that you need to try several different sets of guess values, as there wasn’t an obvious “best” direction from your original ones.
Chapter 7: Non-Linear Estimation

7.2 Newton–Raphson and Related Methods

The ideal function for optimization is quadratic. If

\[ F(x) = a + c'x + \frac{1}{2}x'Qx \]

and \(c\) and \(Q\) are known, then the maximum of \(F\) (if \(Q\) is negative definite) can be found in one step by \(x = -Q^{-1}c\). If, for instance, you feed NLLS a function which is linear in the parameters (and thus the sum of squares is quadratic), the algorithm will usually report convergence in two iterations: the first finds the minimum and the second makes no further progress and thus passes the convergence test.

While we usually don’t have quadratic functions when we do non-linear optimization, there is a large set of tools which can be applied to functions which are twice continuously differentiable, and thus have a second order Taylor expansion. If \(x_k\) is the current estimate, then, for \(x\) in a neighborhood of \(x_k\),

\[ F(x) \approx F(x_k) + g'(x - x_k) + \frac{1}{2}(x - x_k)'H(x - x_k) \]

where \(g\) is the gradient and \(H\) the Hessian (matrix of second derivatives) evaluated at \(x_k\).

Newton–Raphson takes the updated \(x\) to be the optimizer of (2), which is

\[ x = x_k - H^{-1}g \]

There are two main drawbacks to the use of Newton–Raphson as an optimizing algorithm:

1. It requires calculation of the second derivatives of \(H\). If there are \(n\) parameters, this means \(n(n+1)/2\) second derivatives. Computing these can be very time consuming, and, if it has to be done numerically, can be subject to considerable approximation error.

2. Until the estimates are near the optimum, it is possible that \(H\) won’t show the correct curvature. Because Newton–Raphson is really searching only for a zero in the gradient, it could head for a minimum when you want a maximum, or vice versa.

RATS uses Newton–Raphson only for logit, probit and related models (estimated via the DDV and LDV instructions) which have fairly simple second derivatives. And even then, it uses the Newton–Raphson prediction only to give the direction of movement, not the distance.
Chapter 7: Non-Linear Estimation

Hill-Climbing Methods

These are a very general collection of techniques for maximization of a twice-continuously differentiable function. A single iteration takes the following steps:

1. Compute the gradient ($g$)
2. Select a direction by premultiplying the gradient by a matrix, that is $d = Gg$. Make sure the directional derivative is positive, so that (at least locally) we are going “uphill” in that direction. If not, modify the direction.
3. Select a distance to travel in that direction: $x_{k+1} = x_k + \lambda d$

When RATS needs to make an adjustment in the direction in step 2, it does it by adding in a multiple of the gradient. RATS is looking for a direction in which

$$\frac{d \cdot g}{\|d\| \|g\|} \geq \alpha$$

where $\alpha$ is a control parameter. On most iterations, $\alpha$ is zero. However, if RATS detects that the estimation isn’t proceeding smoothly, it will switch to .00001 (by default), which you can reset using the ALPHA option on NLPAR. If the initial direction fails the test, RATS replaces it with $(d + \gamma g)$ where this is solved for the value of $\gamma$ which just meets (4). While you might expect that making $\alpha$ higher would speed up the process by picking a direction in which the function is “steeper,” this is not the case. In fact, taking $d = g$, which maximizes the left side of (4), gives us the infamous “steepest descent” (really “steepest ascent” for maximization problems) which is well known for producing very slow progress towards the maximum because of constant severe direction changes from iteration to iteration.

The process of selecting $\lambda$ is called line search. RATS refers to the attempts to find a suitable $\lambda$ as subiteration. There are two distinct philosophies which can be used in the line search stage. One is called exact (or perfect) line search, which maximizes the function over $\lambda$. In inexact line search, we look for a value of $\lambda$ at which a condition is satisfied which guarantees progress towards the maximum. The choice between these two methods is controlled by the EXACTLINESEARCH option on NLPAR. The default is NOEXACT. For this, RATS tests

$$\delta \leq \frac{F(x_{k+1}) - F(x_k)}{\lambda \, d \cdot g} \leq 1 - \delta$$

where $\delta$ (a number between 0 and .5) is controlled by the DELTA option on the instruction NLPAR. The middle part is the ratio between the directional arc derivative with respect to $\lambda$ and the directional derivative itself. This condition is described in, for instance, Berndt, Hall, Hall and Hausman (1974). The choice of subiteration method rarely affects whether you are able to achieve convergence. It can affect how quickly you achieve it. The default (NOEXACT) is almost always faster if you have fewer than thirty parameters, because it reduces the number of function evaluations.
RATS applies the hill climbing technique with four different methods of computing the multiplying matrix $G$. Some of these are specialized to particular functions and are thus not available in all situations. The four methods (Newton–Raphson, Gauss–Newton, BFGS, and BHHH) are described below.

### Newton–Raphson

Used by **DDV** and **LDV**, Newton–Raphson takes $G = -H^{-1}$. It converges very quickly once it is close to the optimum.

### Gauss–Newton

Gauss–Newton is the primary method for the instructions **NLLS** and **BOXJENK** and is an option on **DLM**. An extension to multiple equations is used by **NLSYSTEM**. This method is specific to non-linear least squares estimation. It is an algorithm for solving problems like:

$$
\begin{align*}
\min_x \sum_{t=1}^{T} u_t^2 & \quad \text{where } u_t = f(y_t, x) \\
\end{align*}
$$

Gauss–Newton is based upon a linearization of the residuals around $x_k$.

$$
(6) \quad u_t(x_k) = -\left(\frac{\partial u_t}{\partial x}\right)(x - x_k) + u_t(x)
$$

This takes the form of a least squares regression of $u_t(x_k)$ on the partial derivatives and is the same as a hill-climbing step (after converting to a maximization) with

$$
(7) \quad G = -\sum_{t=1}^{T} (\partial u_t/\partial x)'(\partial u_t/\partial x)^{-1}
$$

The objective function for instrumental variables (with $Z$ as the instruments) is

$$
(9) \quad \left(\sum_{t=1}^{T} u_t Z_t\right) W \left(\sum_{t=1}^{T} Z_i' u_t\right)
$$

where $W$ is a weighting matrix, typically $W = (\sum Z_i' Z_i)^{-1}$. The same type of linearization gives

$$
(10) \quad G = -\left(\sum_{t=1}^{T} (\partial u_t/\partial x)' Z_t\right) W \left(\sum_{t=1}^{T} Z_i' (\partial u_t/\partial x)\right)^{-1}
$$
BFGS (Broyden, Fletcher, Goldfarb, Shanno)

The BFGS method (described in Press, et. al. (1988)) is used by the instructions BOXJENK, CVMODEL, DLM, FIND, GARCH, MAXIMIZE, and NLSYSTEM with certain options.

G generally starts with a diagonal matrix, the form of which varies from instruction to instruction. At each iteration, it is updated based upon the change in parameters and in the gradient in an attempt to match the curvature of the function. The basic theoretical result governing this is that if the function is truly quadratic, and if exact line searches are used, then in \( n \) iterations, \( G \) will be equal to \(-H^{-1}\). If the function isn’t quadratic, \( G \) will be an approximation to \(-H^{-1}\).

Because the estimate of \( G \) produced by BFGS is used in estimating the covariance matrix and standard errors of coefficients, you need to be careful not to apply BFGS to a model which has already been estimated to a fairly high level of precision. If BFGS uses fewer iterations than the number of parameters being estimated, the \( G \) will be poorly estimated and, hence, the standard errors derived from them will be incorrect. All of the instructions which allow BFGS will let you use the HESSIAN option to set an initial guess for \( G \). If you have a model which is nearly converged, and use an initial \( G \) which is close to the correct matrix as well (for instance, using \%XX from a previous estimation), you can have a bit more confidence in the resulting covariance matrix. If you don’t have a better initial \( G \), move your guess values for the parameters away from the optimum to force a longer estimation sequence.

BHHH (Berndt, Hall, Hall and Hausman)

BHHH implements the proposal for choosing \( G \) recommended in Berndt, Hall, Hall and Hausman (1974). Because it can only be applied to specific types of optimization problems, the only RATS instructions which can use it are MAXIMIZE and GARCH. The function being maximized has the form

\[
F(x) = \sum_{i=1}^{T} f(y_i, x)
\]

where \( F \) must be the actual log likelihood function (possibly omitting additive constants). It chooses \( G \) as \( J^{-1} \) where

\[
J = \sum_{i=1}^{T} \left[ \frac{\partial f}{\partial x} (y_i, x) \right] \left[ \frac{\partial f}{\partial x} (y_i, x) \right]'
\]

Under fairly general conditions, \(-J\) will have the same asymptotic limit (when divided by \( T \)) as the Hessian \( H \). The information equality is used in deriving this, which is why \( F \) is required to be the log likelihood.

If the function is not the log likelihood, the algorithm will still usually end up at the correct maximum, as a good choice of \( G \) generally affects only how quickly the hill is climbed. Convergence, however, is likely to be slow, and the standard errors and covariance matrix will be incorrect.
7.3 Derivative-Free Methods

RATS also offers several methods which do not require a differentiable function. In general, continuity is the only requirement for these.

Simplex Method

The simplex method is a search procedure which requires only function evaluations, not derivatives. It starts by selecting $K+1$ points in $K$-space, where $K$ is the number of parameters. The geometrical object formed by connecting these points is called a simplex. (Note that the only relationship between this and the simplex method of linear programming is that each method utilizes a simplex). One of these vertices represents the initial values. RATS makes each of the other points equal to the initial point plus a perturbation to one and only one of the parameters. Notice that, unlike a general grid search, this initial set of points does not need to enclose the optimum.

The basic procedure at each step is to take the worst of the $K+1$ vertices, and replace it with its reflection through the face opposite. Where hill-climbing methods look for a direction where the function increases, the simplex method moves uphill by eliminating the directions where the function decreases. As a result, the simplex method is more robust to initial conditions and the behavior of the function, but is much slower to converge than hill-climbing techniques when the latter are appropriate.

There is a tendency for the simplex method to get stuck in situations where the points are too much in a line to provide information about the shape. To prevent this, RATS perturbs the vertices every 30 function evaluations. (How often this is done is controlled by the JIGGLE option on NLPAR).

For more information, see, for instance, Press, et. al. (1988).

RATS uses the simplex method (exclusively) to estimate the parameters in ESMOOTH. While ESMOOTH generates models which can be estimated by non-linear least squares models, the Gauss–Newton algorithm that applies to such problems does not deal well with the unstable regions in the parameter space.

You can choose to use the simplex method for estimation for FIND (where it’s the default method), BOXJENK, CVMODEL, DLM, GARCH, MAXIMIZE, NLLS and NLSYSTEM (where it’s an option). FIND is the “catch-all” optimization instruction. If you can’t do it any other way, you can probably do it with FIND. See more in Section 7.12.

For the other seven instructions, the most important use of SIMPLEX is to refine initial estimates before using one of the derivative-based methods, which are more sensitive to the choice of initial estimates. This is usually done by using the PMETHOD (and PITERs) options to choose a “preliminary” method. For instance, you may find that

```
maximize(pmethod=simplex,piters=5,method=bfgs)
```

works better than the BFGS (or BHHH) alone. Note, by the way, that RATS counts iterations for the simplex minimization in a non-standard way: each “iteration”
includes $2K$ changes to the parameter set. This roughly equalizes the number of function evaluations per iteration with the other methods.

**Genetic algorithm**

Genetic algorithms are designed to handle difficult optimization problems by applying lots of compute power and a stylized model of “evolution,” with random “mutations.” RATS applies a variation of this called *differential evolution*. Given a “population” of parameter vectors, at each iteration each vector is compared with a possible successor. Whichever is the “fitter” of the two (the one with the better function value) is retained, the other discarded. There are a number of schemes for generating successors, all of which require that at least one parameter be “mutated,” that is, have some random number added to it. In differential evolution, the size of a mutation in a parameter is determined from the difference between that parameter in two randomly selected elements of the population. If a particular parameter takes values which are still fairly widely dispersed throughout the population, then mutations will be large. If, on the other hand, it becomes clear that an individual parameter is fairly well determined, so that its values are tightly bunched, future changes in it will be small.

`METHOD=GENETIC` is available as an option on the instructions `BOXJENK`, `CVMODEL`, `DLM`, `FIND`, `GARCH`, `MAXIMIZE`, `NLLS` and `NLSYSTEM`.

There are four options on `NLPAR` which govern the genetic algorithm:

- `populate=scale for population size [6]`
- `mutate=[simple]/shrink`
- `crossover=probability of taking mutated parameter [.5]`
- `scalefactor=scale factor for mutations [.7]`

The **POPULATE** option is the simplest of these to explain: it just indicates the factor by which the number of parameters is multiplied to get the population size. Note, however, that the population size is never allowed to be below 20. As mentioned above, at each iteration, each of the parameter vectors in the population is compared against a trial vector, to be replaced if the trial vector is better. The other three options govern how this trial vector is generated. One existing parameter vector is chosen as the base for this. Two other parameter vectors are chosen at random and their difference is taken. In the **SIMPLE** form of mutation, this difference is multiplied by **SCALEFACTOR** and added to the base. A binomial trial is performed for each component. With probability equal to the **CROSSOVER**, the trial value is chosen; otherwise the original value is retained. A large value of **SCALEFACTOR** (bigger than one) is useful if you need to search a wide area. The **SHRINK** form for mutation is similar but, instead of using a randomly selected base vector, it uses a linear combination of the original vector and the best vector from the previous iteration.

It’s likely that the ideal set of choices for this will vary depending upon the function being optimized. If you don’t seem to be making good progress, try different values.
Chapter 7: Non-Linear Estimation

7.4 Constrained Optimization

The previous sections have described optimization problems where the free parameters are allowed to take any values. Now some functions have implicit bounds on the parameters. Variance parameters, for instance, usually are forced to be positive by the presence of a log(\(v\)) term in the function. There is no need to do anything special to see that this is enforced. The function value will be NA if \(v\) goes non-positive, and so RATS will be forced to take a smaller step into positive territory.

However, there are situations in which the parameters are constrained, not by the nature of the function, but by the nature of the problem. If a parameter represents a physical amount of something, a negative value may be nonsensical but an unconstrained optimization might well give us a negative estimate.

If you have a single parameter subject to this type of imposed bound, constrained optimization is simple: estimate the model unconstrained. If the parameter has a permitted value, you’re done. Otherwise, re-estimate with the bound imposed.

However, if you have several constraints, it becomes much more complicated to work through the possible combinations. Instead, you can let RATS do the work for you.

Types of Constraints

There are three basic types of constraints that you can impose on your non-linear parameters. These are all imposed using the instruction NONLIN (note that you must define the parameters before they can appear in a constraint expression—see the examples later in this section):

1. Substitution constraints. These give one parameter as a function of others, for instance, \(B3 = B1 * B2\). This type of constraint is handled directly during the estimation process, so RATS doesn’t need any special techniques.

2. Equality constraints. These are similar to substitution constraints, except that the equation is not explicitly solved for one of the variables. An example is \(B1 * B2 == B3 * B4\). (Note that you must use the == operator). However, if you can solve out for one variable, do it. For instance, here, unless you have a strong possibility that \(B2\) will be forced to zero, using \(B1 = B3 * B4 / B2\) will be more efficient.

3. Inequality constraints. These cannot be strict inequalities. For instance, you can’t restrict a parameter to be positive; it must be constrained to be non-negative. Examples are \(B1 >= 0\), \(B1 + B2 + B3 <= 1.0\).


Algorithm

On each iteration, RATS first determines which of the inequality constraints are to be considered “active.” This includes all constraints which are violated, and any others which have non-zero Lagrange multipliers from the previous iteration. Let the active
constraints (including any equality constraints) be represented as \( c(\beta) = 0 \). The Lagrangean for maximizing \( f(\beta) \) subject to these is

\[
L = f(\beta) - \mu' c(\beta)
\]

where \( \mu \) is the vector of Lagrange multipliers. RATS maintains an estimate of

\[
G = -\left( \frac{\partial^2 L}{\partial \beta \partial \beta'} \right)^{-1}
\]

It computes \( g = \partial f / \partial \beta \) and \( N = \partial c / \partial \beta \). Using these, a new set of Lagrange multipliers is computed using the formula:

\[
\mu = (N'GN)^{-1}(N'GN - c)
\]

These are then used to compute the direction vector

\[
d = G(N\mu + g)
\]

This direction vector would, in a perfect world, take us to a zero gradient point for the Lagrangean.

With the direction chosen, the question then is how far to move in that direction. RATS searches over the step size \( \lambda \) using the following penalty function

\[
f(\beta + \lambda d) - \frac{1}{n} \| c(\beta + \lambda d) \|^2
\]

The scale factor \( r \) changes from iteration to iteration, becoming smaller as the estimates come closer and closer to meeting the constraints. With a new set of parameters in hand, RATS then updates the estimate of \( G \) using the BFGS formula and moves on to the next iteration. Any constrained estimation done with a hill-climbing method will use this variant of BFGS, even if you selected the BHHH or Gauss–Newton methods. *Constraints are ignored when using the derivative-free simplex or genetic methods.*

Constrained optimization is usually quite a bit slower than a similar problem with no constraints. This is not due to the extra calculations done in each iteration. Instead, it simply takes more iterations, often many more. Even if the set of active constraints stays the same throughout the estimation process, as the Lagrange multipliers change in the early going, the Lagrangean itself changes. Thus, we're trying to hit a moving target. Once the Lagrange multiplier estimates settle down, the estimates will start converging more smoothly.

When setting up your constraints, be aware of the sensitivity to scale. If, for instance, you were to put in a constraint such as \( 1000000 * B1 >= 0 \), a fairly modest negative value of \( B1 \) would cause a large value in this constraint’s component of \( c \). This would likely so overwhelm the other constraints that all the early iterations would be devoted to forcing this single constraint to hold.
Chapter 7: Non-Linear Estimation

7.5 Covariance Matrices

The main (and in some cases the only) point of the optimization methods is to answer the question: for what parameters is this function maximized (minimized)? However, most of the interesting optimizations depend upon sample data. Assuming that there is some randomness in the observed data, the point estimates will vary depending upon the sample. This means that providing the point estimates alone will not fully describe the available information; it is also desirable to report some measure of the sensitivity of the estimates to the sample.

The only optimization techniques available in RATS that can provide estimates of the standard errors and covariance matrix are the hill-climbing methods described in 7.2. The derivative-free methods from 7.3 can produce point estimates only.

The Basic Framework

With most of the non-linear estimation instructions in RATS, the function being optimized takes the form

\[ F(\beta) = \sum_{t=1}^{T} f(y_t, X_t, \beta) \]  

The exceptions to this are FIND, where you provide \( F(\beta) \) directly, and CVMODEL, where you provide sufficient statistics for such a sum.

If \( \beta_T \) is the estimate using data through \( T \) and \( \beta_o \) is the true parameter, then a first-order Taylor expansion of the gradient is

\[ \frac{\partial F}{\partial \beta}(\beta_T) = \frac{\partial F}{\partial \beta}(\beta_o) + \frac{\partial^2 F}{\partial \beta \partial \beta'}(\beta_T - \beta_o) \]

The left-hand side of this is zero, so we can solve to get

\[ \sqrt{T}(\beta_T - \beta_o) \approx -\left( \frac{1}{T} \frac{\partial^2 F}{\partial \beta \partial \beta'} \right)^{-1} \frac{1}{\sqrt{T}} \frac{\partial F}{\partial \beta}(\beta_o) \]

Using arguments similar to those in Section 5.3, the second factor on the right can be written

\[ \frac{1}{\sqrt{T}} \sum_{t=1}^{T} \frac{\partial f}{\partial \beta}(y_t, X_t, \beta) \]

where the summands have expected value zero. Again, under the proper conditions (see, for instance, Hayashi (2000), section 7.3) this will have an asymptotic Normal distribution with mean zero and a covariance matrix that can be estimated consistently by

\[ B_T = \frac{1}{T} \left\{ \sum_{k=-L}^{L} \sum_{t} \frac{\partial f}{\partial \beta}(y_t, X_t, \beta) \frac{\partial f}{\partial \beta}(y_{t-k}, X_{t-k}, \beta) \right\} \]
Combined with

\[
\mathbf{A}_T = \frac{1}{T} \sum_t \frac{\partial^2 f}{\partial \beta \partial \beta'}(y_t, \mathbf{X}_t, \beta)
\]

we get the familiar expression

\[
\sqrt{T} (\beta_T - \beta_o) \sim N(0, \mathbf{A}_T^{-1} \mathbf{B}_T \mathbf{A}_T^{-1})
\]

\(\mathbf{B}_T\) can be calculated from the derivatives computed during each iteration. It’s \(\mathbf{A}_T\) that we don’t necessarily have. The only hill-climbing method that computes it is Newton–Raphson, which is used only by \texttt{DDV} and \texttt{LDV}. However, BFGS produces an approximation to the (inverse) Hessian. As stated in Section 7.2, BFGS exactly produces the Hessian in \(K\) iterations (\(K=\text{number of parameters}\)) if the function \(F(\beta)\) is quadratic, and exact line searches are used. If a function is only locally quadratic (which is a required assumption to get the covariance matrix using this type of analysis), then BFGS produces an approximation. But be careful: if you start BFGS very close to the final estimates, it won’t have a chance to “map out” the curvature, as the gradient will already be quite close to zero. To give BFGS a decent chance to develop the curvature estimates, you may need to start with initial conditions pulled away from the optimum.

**Simplifying Assumptions**

In many cases, the calculation of the covariance matrix can be simplified. In some (for instance, with \texttt{CVMODEL} or \texttt{FIND}), it has to be, as the information to compute \(\mathbf{B}_T\) isn’t available. All of these start by assuming the \(L\) in (5) is zero. A non-zero \(L\) means that the partial derivatives are correlated across time. If the observations are independent, or the model is assumed to exhaust any correlation, this should be a safe assumption.

If \(F(\beta)\) is the log-likelihood function, then the information equality applies (see, for instance, Hamilton (1994), section 14.4). \(\mathbf{A}_T\) and \(\mathbf{B}_T\) have the same expected values; they aren’t, however, equal. Whichever one we can compute most easily can substitute for the other. In the case of \texttt{METHOD=BHHH}, we are computing \(\mathbf{B}_T\) (with \(L=0\)) at every iteration to get the \(\mathbf{G}\) matrix in the hill-climbing procedure. For \texttt{METHOD=BFGS}, it’s \(\mathbf{A}_T\) (more accurately, its inverse) that we’re computing (approximately).
Chapter 7: Non-Linear Estimation

Cautions

Different methods of estimation will generally produce point estimates which agree
to the precision that you request. The covariance matrices, however, can differ
substantially. It’s not uncommon for standard errors to disagree at the second or
third significant digit. This is because they are all based upon the asymptotic equiva-
rence of matrices which not identical in sample.

By default, **MAXIMIZE** assumes that the information equality applies. If $F(\beta)$ is not
the log likelihood function (other than missing some additive constants), the simpli-
fying assumptions above are wrong, and the covariance matrix is likely to be quite
wrong as well. For instance, if you use **MAXIMIZE** to fit a least squares model by
maximizing $\sum(-u_i^2)$, the point estimates will be correct, but the standard errors
will be off by a factor of $\sqrt{2\sigma^2}$. This is because the actual log likelihood element is
$$-5\left(\frac{u_i^2}{\sigma^2}\right),$$
not just $-u_i^2$.

Because **FIND** can be used for optimization problems which are not based upon
sample data, it doesn’t automatically compute and display standard errors. You need
to include the **STDERRS** option if you want it to do so and you must be using
**METHOD=BFGS**. The calculation of these assumes the information equality holds.

**NLLS and NLSYSTEM**

The covariance matrix calculations for **NLLS** and **NLSYSTEM** are similar to those
above but exploit the special structure of the problem. For **NLLS**, for instance,

(8) \[ F(\beta) = \sum_{i=1}^{T} (y_i - f(X_i, \beta))^2 \]

(9) \[ \frac{\partial F}{\partial \beta} = -2\sum_{i=1}^{T} (y_i - f(X_i, \beta)) \frac{\partial f}{\partial \beta}(X_i, \beta) \]

(10) \[ \frac{\partial^2 F}{\partial \beta \partial \beta'} = 2\sum_{i=1}^{T} \frac{\partial f}{\partial \beta}(X_i, \beta) \frac{\partial^2 f}{\partial \beta \partial \beta'}(X_i, \beta) - 2\sum_{i=1}^{T} (y_i - f(X_i, \beta)) \frac{\partial^2 f}{\partial \beta \partial \beta'}(X_i, \beta) \]

The second sum in (10) is the sum of the product of residuals times a function of the
exogenous variables only. When divided by $T$, it isn’t unreasonable to assume that
this will become negligible in large samples. Thus, this term is ignored in computing
$A_T$, which eliminates the need to compute second derivatives. The convenient
simplification here is to assume that

(11) \[ E(u_i^2|X_i) = \sigma^2 \]

which, when applied to (9) gives us

(12) \[ E(B_T) = \sigma^2 E(A_T) \]
The ROBUSTERRORS option is available on the instructions MAXIMIZE, NLSYSTEM, NLLS, LDV, DDV, and GARCH to do the full covariance matrix calculation described by (5), (6) and (7). This can only be used with METHOD=BFGS on MAXIMIZE and METHOD=GAUSS (the default) for NLLS and NLSYSTEM.

The LAGS option, when used together with ROBUSTERRORS, gives a non-zero value to \( L \) in (5). This is subject to the same caution as for linear models (Section 5.3): if you don’t use a choice for LWINDOW like NEWEYWEST, the covariance matrix computed could fail to be positive definite.

ROBUSTERRORS with CLUSTER=SERIES with category values allows for arbitrary patterns of correlation in the gradient within the categories given by the values of the series you provide. You must have at least as many categories as parameters to estimate, and generally should have many more.
7.6 Setting Up Your Model: NONLIN and FRML

While some of the instructions covered in this chapter are special purpose instructions which handle only one type of model, many of them allow you to, and in fact require you to, provide the function to be optimized. In addition, you have to indicate the set of parameters which are to be estimated. This second task is done with the instruction NONLIN. The first is usually done with the help of the instruction FRML.

PARMSETs

A PARMSET is a specialized data type which consists of a list of parameters (a “parameter set”) for non-linear estimation and constraints on them. PARMSETs are constructed and maintained using the instruction NONLIN, and are used by instructions CVMODEL, DLM, FIND, GARCH, MAXIMIZE, NLLS, and NLSYSTEM. PARMSETs can be combined using the standard “+” operator. You can set up vectors or matrices of them or pass them as procedure parameters.

The Instruction NONLIN

NONLIN sets up or edits the list of free parameters in the model. In its simplest form, it simply defines a list of free parameters to be estimated by a subsequent instruction such as NLLS:

\[
\text{nonlin alpha gamma delta} \\
\text{nonlin g b0 b1 b2}
\]

NONLIN can accept either single REAL variables as parameters or a complete VECTOR or other real array, or an array of real-valued arrays, such as a VECTOR[VECTOR]. For instance,

\[
\text{declare vector b(5)} \\
\text{nonlin b rho}
\]

will create a parameter set which includes the five elements of B plus the single value RHO. A vector used in this way doesn’t have to be dimensioned at the time of the NONLIN instruction, but must be dimensioned before being used for estimation.

These above instructions have created an internal PARMSET, which is the one used by the estimation instructions unless you provide a different one. To create a different PARMSET, use the PARMSET option on the NONLIN.

\[
\text{nonlin(parmset=boxcoxparms) sigma lambda} \\
\text{nonlin(parmset=regparms) b}
\]

The first of these creates a PARMSET named BOXCOXPARMS which includes the two variables SIGMA and LAMBDA. The second creates REGPARMS. You can combine one PARMSET with another using the “+” operation, either in a COMPUTE instruction or directly on the estimation instruction. For instance,

\[
\text{compute fullparm=boxcoxparms+regparms or} \\
\text{maximize(parmset=boxcoxparms+regparms) ...}
\]
Chapter 7: Non-Linear Estimation

The functions \%PARMSPEEK and \%PARMSPOKE can be used (with a named PARMSET) to move values into and out of a parameter set. For instance, in the last example, \%PARMSPEEK(BASE) will return a vector with elements G, B0, B1 and B2 in order, and \%PARMSPOKE(BASE, V) would make G=V(1), B0=V(2), etc. This can be handy when you wish to parameterize a function using single variable names (which makes it easier to change the function, and read the estimation output), but you also need to be able to move information into and out of the parameter set easily. For instance,

\[
\text{nonlin(parmset=simszha) a12 a21 a23 a24 a31 a36 a41 $ a43 a46 a51 a53 a54 a56}
\]

\[
\text{... compute axbase=\%parmspeek(simszha)}
\]

\[
\text{... compute \%parmspoke(simszha,axbase+saxx*au)}
\]

In the application from which this is taken, it’s necessary to reset the parameters using a matrix operation. However, parameterizing the function as A(13) would have been clumsy, since adding or deleting a parameter would have required a complete rewrite of six lines defining a matrix. By using \%PARMSPEEK and \%PARMSPOKE, you can set up the PARMSET in the most convenient fashion and still be able to set the elements using matrix operations.

**Using NONLIN to Impose Constraints**

NONLIN is also used to add constraints on the parameters. See Section 7.4 for more on constrained optimization.

\[
\text{nonlin(parmset=base) g b0 b1 b2}
\]

\[
\text{nonlin(parmset=constraint) b0>=0.0 b1>=0.0 b2>=0.0}
\]

\[
\text{nlls(parmset=base) ... nlls(parmset=base+constraint)}
\]

This does NLLS on a model, the first time without constraints imposed, the second with non-negativity constraints on the three “b” parameters. Note that constraints are only obeyed for certain types of non-linear optimization methods. In particular, the simplex and genetic algorithms won’t impose inequality constraints.

Constraints can also apply to all elements of a vector or matrix. For instance, the following creates a PARMSET with the single real variable A and a vector with NLAGS elements B, and constrains the B’s to be positive.

\[
\text{dec vect b(nlags)}
\]

\[
\text{nonlin a b b>=0}
\]
Chapter 7: Non-Linear Estimation

**FRMLs**

A FRML is a specialized data type which describes a function of the entry number T. Usually this function includes references to current and lagged data series. For non-linear estimation, it will also use the parameters that are to be estimated.

You can define vectors and arrays of FRMLs, although you must take some special care in defining the elements of these in a loop. FRMLs can also be passed as procedure parameters.

FRMLs which are used in non-linear estimation need to be created using the instruction **FRML**. FRML’s can also be created by **LINREG** and some other instructions from the results of regressions. Those, however, have the estimated coefficients coded into them, so they are of no use for further estimation.

FRML’s usually produce a real value, but you can also create formulas which produce matrices. These are used, for instance, by the instructions **CVMODEL** and **DLM**. You need to do a **DECLARE** instruction as the first step in creating such a formula. For instance, to make A a formula which returns a RECTANGULAR, do

```
DECLARE FRML[RECT] A
```

**The Instruction FRML**

The **FRML** instruction is used to define the formula or formulas that you want to estimate. **FRML** will usually take the form:

```
frml  formula name  dependent variable = function(T)
```

The **dependent variable** is often unnecessary. Some simple examples follow:

```
nonlin k a b
frml logistic = 1.0/(k+a*b**t)
```

is a formula for a logistic trend. The parameters K, A and B will be estimated later.

```
nonlin b0 b1 b2 gamma sigmasq
frml olsresid = pcexp-b0-b1*pcaid-b2*pcinc
frml varfunc  = sigmasq*(pop**gamma)
```

translates the following into FRMLs: $PCEXP_t - \beta_0 - \beta_1 PCAID_t - \beta_2 PCINC_t$ and $\sigma^2 POP_t^\gamma$ with $\beta_0$, $\beta_1$, $\beta_2$, $\gamma$ and $\sigma^2$ representing the unknown parameters.

```
nonlin i0 i1 i2 i3 i4
frml investnl invest = $
  i0+i1*invest{1}+i2*ydiff{1}+i3*gnp+i4*rate{4}$
```

translates an investment equation which depends upon current GNP and lags of INVEST, YDIFF and RATE. You could also do this (more flexibly) with:

```
frml(regressors,vector=invparms) investnl invest
  # constant invest{1} ydiff{1} gnp rate{4}
nonlin invparms
```
Recursive FRMLs

You can create an FRML to compute a function which depends upon its own value at the previous data point. Such recursively defined functions are not uncommon in time series work. For instance, in a GARCH model, this period’s variance depends upon last period’s. In a model with a moving average term, this period’s residual depends upon last period’s. We will demonstrate how to handle this for a geometric distributed lag. (This is for purpose of illustration. You can estimate this more easily with BOXJENK).

\[ y_t = \beta_0 + \beta_1 \left\{ \sum_{s=0}^{\infty} \lambda^s X_{t-s} \right\} \]

We can generate the part in braces (call it \( Z_i \)) recursively by:

\[ Z_i = X_i + \lambda Z_{i-1} \]

The parameter \( \text{THETA} \) represents the unobservable

\[ Z_0 = \sum_{s=0}^{\infty} \lambda^s X_{-s} \]

At each \( T \), the formula below sets \( XLAGS \) equal to the new value of \( Z \), and then uses this value to compute the next entry. \( \% \text{IF} \) is used to distinguish the first observation from the others, as it needs to use \( \text{THETA} \) for the pre-sample value of \( Z \).

\[ \text{declare real xlags} \]
\[ \text{nonlin beta0 beta1 theta lambda} \]
\[ \text{frml geomdlag = (xlags = x + lambda*%if(t==1949:1,theta,xlags))},\]
\[ \beta0 + \beta1 * xlags \]

Notice that the formula includes two separate calculations: the first computes \( XLAGS \), and the second uses this to create the actual value for the formula. This ability to split the calculation into manageable parts is a great help in writing formulas which are easy to read and write. Just follow each of these preliminary calculations with a comma. The final value produced by the formula (the expression after the last comma) is the one that is used.

The handling of the initial conditions is often the trickiest part of both setting up and estimating a recursive function. The example above shows one way to do this: estimate it as a free parameter. Another technique is to set it as a “typical” value. For instance, if \( X \) were a series with zero mean, zero would not be an unreasonable choice. The simplest way to set this up is to make \( XLAGS \) a data series, rather than a single real value.

\[ \text{set xlags = 0.0} \]
\[ \text{nonlin beta0 beta1 lambda} \]
\[ \text{frml geomdlag = (xlags=x+lambda*xlags{1}),beta0+beta1*xlags} \]

When the formula needs an initial lagged value of \( XLAGS \), it pulls in the zero.
Chapter 7: Non-Linear Estimation

The estimation instructions which depend upon FRMLs (DLM, MAXIMIZE, NLLS, and NLSYSTEM) all have a STARTUP option, which allows you to define a FRML which is executed before each function or derivative calculation. This allows you to write your main formula without having to worry about the special case of the first observation. For instance, our original coding for the formula can be rewritten as:

\begin{verbatim}
declare real xlags
nonlin beta0 beta1 theta lambda
frml initgeom = xlags=theta
frml geomdlag = (xlags=x+lambda*xlags),beta0 + beta1 * xlags
\end{verbatim}

Using STARTUP=INITGEOM on the instruction which estimates the model will cause XLAGS to be initialized to the current value of THETA at the start of every function evaluation.

Using Sub-FRMLs

Many of the models which you will be estimating will have two or more distinct parts. For instance, in a maximum likelihood estimation of a single equation, there is usually a model for the mean and a model for the variance. While both parts are needed for the complete model, there is no direct interaction between the two. You might very well want to alter one without changing the other.

This can be handled within RATS by defining separate FRMLs for each part, and then combining them into a final FRML. Redefining one of the components has no effect upon the other. For instance,

\begin{verbatim}
nonlin b0 b1 b2 gamma sigmasq
frml olsresid = pcexp-b0-b1*pcaid-b2*pcinc
frml varfunc = sigmasq*(pop**gamma)
frml likely = %logdensity(varfunc(t),olsresid(t))
\end{verbatim}

defines the log likelihood function for a model with Normally distributed errors whose variance is proportional to a power of the series POP. The variance model and regression residuals model are represented by separate formulas. The LIKELY formula doesn’t need to know anything about the two formulas which it references.

There is one minor difficulty with the way the model above was coded: the single NONLIN instruction declares the parameters for both parts. This is where PARMSETS can come in handy. If we rewrite this as

\begin{verbatim}
nonlin(parmset=olsparms) b0 b1 b2
frml olsresid = pcexp-b0-b1*pcaid-b2*pcinc
nonlin(parmset=varparms) gamma sigmasq
frml varfunc = sigmasq*(pop**gamma)
frml likely = %logdensity(varfunc(t),olsresid(t))
\end{verbatim}

then the PARMSET for the complete model is OLSPARMS+VARPARMS.
Creating a FRML from a Regression

In the last example, the OLSRESID formula was fairly typical of the “mean” model in many cases: it’s linear in the parameters, with no constraints. This could be estimated by LINREG if it weren’t for the non-standard model of the variance.

This is a relatively small model, but it is possible to have a model for the mean which has many more explanatory variables than this one. Coding these up as formulas can be a bit tedious. Fortunately, FRML provides several ways to simplify this process. We demonstrated the REGRESSORS option earlier—it takes the right side of the formula from a list of regressors. For example, we can set up this model with the following:

```r
frml(regressors, parmset=olsparms, vector=b) olsmodel
#   constant pcaid pcinc
nonlin(parmset=varparms) gamma sigmasq
frml varfunc = sigmasq*(pop**gamma)
frml likely = %logdensity(varfunc(t),pcexp-olsmodel(t))
```

The first FRML instruction does the following:

1. Creates OLSMODEL as the formula \( B(1) + B(2) \times PCAID + B(3) \times PCINC \).
2. Puts the 3-element vector \( B \) into the PARMSET named OLSPARMS.

The final function has to be altered slightly because OLSMODEL gives the explained part of the model, not the residual. Notice that, with this way of setting up the model, you can change the mean model by just changing the list of explanatory variables on the supplementary card.

You can also create formulas following a LINREG statement by using FRML with the LASTREG option, or by using the EQUATION option to convert an estimated equation.

Defining FRMLs in a Loop

You can create VECTORS or other arrays of FRMLs. This can be very handy when you have a large number of FRMLs with a similar form. 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.

```r
dec vector b(n)
dec vect[frml] blackf(n)
nonlin gamma b
do i=1,n
   frml blackf(i) s(i) = (1-b(&i))*gamma+b(&i)*market
end do i
```

The &i’s are needed in the formula because \((1-B(i)) \times GAMMA + B(i) \times MARKET\) is a perfectly good formula, which would be calculated using the value of \(i\) at the time the formula is used, not at the time it was defined.
Chapter 7: Non-Linear Estimation

7.7 Non-Linear Least Squares/Two-Stage Least Squares

Background

RATS can estimate, by non-linear least squares or non-linear two-stage least squares, models of the form

\[ y_i = f(X_i, \beta) + u_i \]

This is done using the instruction \texttt{NLLS} (the instruction \texttt{NLSYSTEM} (see Section 7.9) performs non-linear systems estimation and multivariate GMM, while \texttt{MAXIMIZE} (7.10) is designed to handle more general problems).

The objective function for non-linear least squares is

\[ \sum_{t=1}^{T} u_t^2 \]

and for non-linear two-stage least squares it is

\[ \left( \sum_{i=1}^{T} u_i Z_i \right) W \left( \sum_{i=1}^{T} Z_i' u_i \right) \]

where \( W = (\sum Z_i' Z_i)^{-1} \)

Section 7.8 describes how to create and use a different weighting matrix \( W \).

The Instruction NLLS

The syntax for the \texttt{NLLS} instruction, which does the actual estimation by least squares or two-stage least squares, is:

\texttt{nlls(formula, other options) depvar start end resids}

Before you can run \texttt{NLLS} you need to:

- Set the parameter list using \texttt{NONLIN}.
- Set up the function using \texttt{FRML}.
- Set initial values for the parameters, using, for instance, \texttt{COMPUTE}.

Technically, \texttt{NLLS} is set up for models strictly of the form (1). However, you can use * in place of \texttt{depvar}. \texttt{NLLS} interprets this as a series of zeros. Since \( R^2 \) and related statistics are meaningless in such cases, they are omitted.

By default, \texttt{NLLS} estimates using the Gauss–Newton algorithm, which is described in Section 7.2. The covariance matrix is calculated as detailed in 7.5. However, it also provides the derivative-free methods \texttt{SIMPLEX} and \texttt{GENETIC} (Section 7.3). These can be applied if the function is continuous but not differentiable, and they can also be used to refine initial guesses before using the standard method, or to make a broader search of the parameter space.
Chapter 7: Non-Linear Estimation

Instrumental Variables

As with LINREG, you do instrumental variables by setting the instruments list using INSTRUMENT and including the INST option when estimating

\[ \text{nonlin discount riskaver} \]
\[ \text{frml } h = \text{discount} \times \text{realret} \times \text{consgrow}^{\times \text{riskaver}-1} \]
\[ \text{compute discount} = .99, \text{riskaver} = -.95 \]
\[ * \]
\[ \text{instruments constant consgrow{1 to 4} realret{1 to 4}} \]
\[ \text{nlls(frml=h,inst,noprint)} * \]

Constrained Estimation

NLLS can handle parameter sets with constraints imposed. However, it doesn’t use the Gauss–Newton algorithm. Instead, it uses constrained BFGS as documented in Section 7.4.

Hypothesis tests

You can test hypotheses on the coefficients using TEST, RESTRICT and MRESTRICT. Coefficients are numbered by their positions in the parameter vector. You cannot use EXCLUDE since it requires a variable list rather than coefficients.

The hypothesis tests are based upon a quadratic approximation to the sum of squares surface at the final estimates. This is the “Wald” test. If you have used the option ROBUSTERRORS, they will report a chi-squared test. If not, they will report an F. The F is not strictly valid, as the numerator in the F is only asymptotically chi-squared; however, it most likely has better small sample properties than the asymptotic distribution, since it allows for the fact that the variance of the residuals is estimated and not a known constant.

Example

The following estimates the parameters for a CES production function

\[ \log Q_i = \log \gamma - \frac{\nu}{\rho} \log(\delta K_i^{-\rho} + (1 - \delta) L_i^{-\rho}) + u_i \]

\[ \text{nonlin lgamma delta nu rho} \]
\[ \text{frml ces = lgamma-nu/rho* } \]
\[ \log( \delta \times k**(-rho) + (1.-delta) \times l**(-rho) ) \]
\[ * \]
\[ \text{compute lgamma=1.0} \]
\[ \text{compute delta=0.4} \]
\[ \text{compute nu=0.8} \]
\[ \text{compute rho=0.6} \]
\[ * \]
\[ \text{nlls(frml=ces,trace) logq} \]
Example 7.1 Non-Linear Least Squares

This analyzes a consumption function of the form $C = \alpha_0 + \alpha_1 YD^{\alpha_2}$. This is from Examples 10.1 and 10.3 of Pindyck and Rubinfeld (1998). The program is supplied on the example file NLLS.PRG.

```
calendar(q) 1947:1
open data nllsmpc.xls
data(format=xls,org=obs) 1947:1 1995:3

For convenience, rename the dataset variables to match the P&R names.

set c   = gcq
set yd  = gydq

Estimate the non-linear model using NLLS

nonlin a0 a1 a2
frml consfrml c = a0 + a1*yd**a2
compute a0=a1=a2=1.0
nlls(frml=consfrml) c
compute rssunr=%rss

Do a Wald test of $a_2=1$. $a_2$ is coefficient #3

test
# 3
# 1.0

Impose the restriction that $a_2$ is 1.

nonlin a0 a1 a2=1.0
nlls(frml=consfrml, iters=50) c / resids

Do a likelihood ratio test

cdf(title="Likelihood Ratio Test") chisqr $
%\text{nobs}(\log(%\text{rss}) - \log(rssunr))$ 1

Do an LM test. Here this will be a test of the significance of a regression of the residuals on the derivatives of the residuals with respect to three parameters, evaluated at the restricted estimates. The derivatives with respect to $a_0$ and $a_1$ are just CONSTANT and $YD$, so we only have to create a new series for the derivative with respect to $a_2$.

set a2deriv = a1*yd*log(yd)
linreg resids
# constant yd a2deriv
cdf(title="Lagrange Multiplier Test") chisqr %\text{trsq} 1
```
Chapter 7: Non-Linear Estimation

7.8 Method of Moments Estimators (Univariate)

Background Information

RATS can produce generalized method of moments estimators (see Hansen (1982)) for models whose orthogonality conditions can be expressed as

\[ E[Z_i f(y_i, X_i, \beta)] = 0 \]

for a closed form function \( f \) and a list of instruments \( Z \) which does not depend upon \( \beta \). This is a very broad class, which includes two-stage least squares and other instrumental variables estimators. It can be done with \texttt{LINREG} (if \( f \) is linear) or \texttt{NLLS}. The extension to multiple equations is treated in Section 7.9.

We will describe here techniques for implementing some of the less standard results from Hansen’s paper.

The OPTIMALWEIGHTS and WMATRIX Options

Ordinarily, RATS computes an instrumental variables estimator by minimizing

\[ u'Z(Z'Z)^{-1}Z'u \]

which is non-linear two-stage least squares (\( Z \) and \( u \) are obtained by stacking observations into a matrix). The matrix \( (Z'Z)^{-1} \) serves to weight the orthogonality conditions. Hansen shows that a more efficient estimator can be obtained by replacing \( (Z'Z)^{-1} \) by the inverse of

\[ mcov(Z, u) = \sum_{k-L}^{L} \sum_{t} Z_t' u_t u_{t-k} Z_{t-k} \]

This matrix plays a key role in the covariance matrix calculations described in Section 5.3, and some of its numerical properties are discussed there.

RATS provides an option called \texttt{OPTIMALWEIGHTS} on \texttt{LINREG} and \texttt{NLLS} for doing this estimation directly. For example:

\begin{verbatim}
instruments constant z1{1 to 6}
linreg(inst,optimalweights,lags=2,lwindow=newey) y1
# constant x1 x2 x3
\end{verbatim}

For \texttt{LINREG}, \texttt{OPTIMALWEIGHTS} tells RATS to compute the matrix shown in (3) and takes its inverse as the weighting matrix. \texttt{NLLS} is similar, but it resets the weight matrix with each iteration. With either instruction, you can retrieve the last weight matrix used as the \texttt{SYMMETRIC} array \%WMATRIX.

If you wish to provide your own weighting matrix, you can use the \texttt{WMATRIX} option on \texttt{LINREG} or \texttt{NLLS}. These steps would be used to compute your own weight matrix:

1. Estimate the model in the standard way, saving the residuals.
Chapter 7: Non-Linear Estimation

2. Use **MCOV** to compute the \( \text{mcov}(Z, u) \) matrix and use **COMPUTE** to invert it. Note that, to get the proper scale for the covariance matrix, the weight matrix needs to be the inverse of a matrix which is \( O(T) \).

3. Reestimate the model with the options **WMATRIX=new weighting matrix**. If this is an (intentionally) sub-optimal weight matrix, you can use **ROBUST-ERRORS**, **LAGS** and **LWINDOW** to correct the covariance matrix.

**J-Tests and the %UZWZU Variable**

Another result from Hansen is that if the optimal weights are used, the value of \( u'ZWZ'u \) is asymptotically distributed as \( \chi^2 \) with degrees of freedom equal to the number of overidentifying restrictions. He proposes this as a test of these assumptions. This is sometimes known as the \( J \)-test. **LINREG** and **NLLS** compute and print the \( J \)-statistic and its significance level using the weight matrix that you supply, or the optimal weight matrix if you choose the **OPTIMALWEIGHTS** option. This is included in the regression output, and is available afterwards in the variables %JSTAT, %JSIGNIF and %JDF (test statistic, significance level, and degrees of freedom respectively).

If the weight matrix used in estimation isn’t the “optimal” one, there are two ways to adjust the specification test. One uses the alternative form for the test statistic (Hansen’s Lemma 4.1). The other, proposed by Jagannathan and Wang (1996), uses the standard \( J \)-statistic but uses its (non-standard distribution). You choose between these with the option **JROBUST=STATISTIC** or **JROBUST=DISTRIBUTION**.

In addition, the **LINREG** and **NLLS** instructions (and the systems estimators **NLSYSTEM** and **SUR**) define the variable \( %UZWZU \). \( %UZWZU \) is the value of \( u'ZWZ'u \) for whatever weight matrix is used. In the case of simple 2SLS, it will be the \( J \)-statistic times the estimated variance. Note that if the model is just identified (that is, if the number of free coefficients is equal to the number of instruments), \( %UZWZU \) is zero, theoretically. It may be very slightly different due to a combination of computer roundoff error and (for non-linear models) the differences between the estimated and true parameters.

**The ZUMEAN option**

The model (1) can be a bit too restrictive in some applications. For instance, in financial econometrics, it’s not uncommon for a model to generate something similar to (1), but with a fixed but non-zero value for the expectation. The calculations done for GMM can easily be adjusted to allow for such a fixed mean for the moment conditions. To do this with RATS, use the **ZUMEAN=vector of expected values**. The vector of expected values should have the same dimensions as the set of instruments.

```
nonlin delta gamma
frml fx = delta*cons**(-gamma)
instruments r1 to r10
compute [vect] zumean=%fill(10,1,1.0)
nlls(frml=fx,inst,zumean=zumean,optimalweights)
```
Example 7.2 Generalized Instrumental Variables

The model (1) in Section 7.8 is general enough to handle situations where \( f \) is not a residual function, and instead, (1) represents first order conditions from a maximization problem. You can implement this in RATS by using an * in place of the \texttt{depvar} parameter.

This example runs through the calculations for the single return models in Hansen and Singleton (1982) on a constructed data set. The example can be found on the file \texttt{GIV.PRG}.

calendar(q) 1960:1
open data hsdale.wks
data(format=wks,org=cols) 1960:1 1987:4 c price div *
set consgrow = c/c{1}
set realret = (price+div)/price{1}
*
nonlin discount riskaver
frml h = discount*realret(t)*consgrow(t)**riskaver-1
compute discount = .99,riskaver = -.95

Estimate the model with six lags in the information set

\[
\text{instruments constant consgrow\{1 to 6\} realret\{1 to 6\}}
\]
\[
\text{nlls(inst,frml=h,optimal) *}
\]

Do specification tests on lag lengths 1, 2, 4 and 6. These are done over a common interval (the range supported by the six lag estimation—\%REGSTART() returns the start entry from the last regression). The NLLS output is suppressed and the tests results are displayed using CDF.

\[
\text{compute start=%regstart()}
\]
\[
\text{do for nlag = 1 2 4 6}
\]
\[
\text{instruments constant consgrow\{1 to nlag\} realret\{1 to nlag\}}
\]
\[
\text{nlls(inst,noprint,frml=h,optimal) * start *}
\]
\[
\text{cdf(title="Specification Test for "+nlag+" lags")}$
\]
\[
\text{chisqr %uzwzu 2*nlag-1}
\]
\[
\text{end do for}
\]

Same specification tests using REPORT to pretty up the output

\[
\text{report(action=define,hlabel=||"Lags","J-Stat","P-Value"||)}
\]
\[
\text{do for nlag = 1 2 4 6}
\]
\[
\text{instruments constant consgrow\{1 to nlag\} realret\{1 to nlag\}}
\]
\[
\text{nlls(inst,noprint,frml=h,optimal) * start *}
\]
\[
\text{report(atcol=1,row=new) nlag %jstat %jsignif}
\]
\[
\text{end do for}
\]
\[
\text{report(action=format,atcol=2,width=8)}
\]
\[
\text{report(action=format,atcol=3,picture="#.###")}
\]
\[
\text{report(action=show)}
\]
Chapter 7: Non-Linear Estimation

7.9 Non-Linear Systems Estimation

NLSYSTEM vs. SUR

The instruction NLSYSTEM estimates a system of equations by non-linear least squares or (for instrumental variables) by the generalized method of moments. The analogous instruction for linear systems is SUR, which is described in Section 5.11. NLSYSTEM uses formulas (FRMLS) rather than the equations used by SUR. For models which can be estimated with both instructions, it is much slower than SUR for two reasons:

- SUR does not have to compute derivatives with each iteration (the derivatives of the linear functions are just the explanatory variables).
- SUR uses every possible means to eliminate duplicate calculations. The cross-products of regressors, dependent variables and instruments can be computed just once.

For a linear model without complicated restrictions, use SUR. For a linear model with restrictions, NLSYSTEM may be simpler to set up. Thus, even if NLSYSTEM takes longer to estimate the model, you get the right answers sooner.

NLSYSTEM vs. NLLS

Like NLLS, NLSYSTEM can do either instrumental variables or (multivariate) least squares. In most ways, NLSYSTEM is just a multivariate extension of NLLS. There are three important differences, though:

- When you use NLLS, you specify the dependent variable on the NLLS instruction itself. With NLSYSTEM, you must include the dependent variable when you define the FRML.
- NLSYSTEM will automatically compute the optimal weighting scheme described in the last section.
- Primarily because of the preceding point, you will rarely use ROBUSTERRORS with NLSYSTEM. ROBUSTERRORS, in fact, is a request to compute a sub-optimal estimator and correct its covariance matrix.

Technical Details

(1)  $u_t = (u_{1,t}, ..., u_{nt})'$ 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) \( Z_t = (z_{t1}, \ldots, z_{tr})' \) is the vector of instruments at \( t \), and

(5) \( G(\beta) = \sum_t u_t \otimes Z_t \)

then Generalized Method of Moments solves

(6) \( \min_{\beta} G(\beta)' [SW] G(\beta) \)

where \( SW \) is the weighting matrix for the orthogonality conditions. 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.

In either case, the estimation process depends upon certain “nuisance parameters”–the estimates will change with \( \Sigma \) and \( SW \). Unless you use options to feed in values for these, \texttt{NLSYSTEM} recomputes them after each iteration. The objective function is thus changing from one iteration to the next. As a result, the function values that are printed if you use the \texttt{TRACE} option will often seem to be moving in the wrong direction, or at least not changing. However, the old versus new values on any single iteration will always be improving.

The continuous recalculation of weight matrices for instrumental variables can sometimes create problems if you use the \texttt{ZUDEP} option. This allows for general dependence between the instruments (“\( Z \)”) and residuals (“\( u \)”). You may not be able to freely estimate the covariance matrix of \( n \times r \) moment conditions with the available data. Even if \( n \times r \) is less than the number of data points (so the matrix is at least invertible), the weight matrices may change so much from iteration to iteration that the estimates never settle down. If this happens, you will need to switch to some type of “sub-optimal” weight matrix, such as that obtained with \texttt{NOZUDEP}, and use the \texttt{ROBUSTERRORS} option to correct the covariance matrix.

**Multivariate Least Squares**

The first order necessary conditions for minimizing (3) are

(7) \( \sum_t \frac{\partial u_t'}{\partial \beta} \Sigma^{-1} u_t = 0 \)

Ignoring the second derivatives of \( u \), a first order expansion of the left side in (7) (which is the gradient \( g \)) at \( \beta_k \) is

(8) \( \sum_t \frac{\partial u_t'}{\partial \beta} \Sigma^{-1} u_t + \left( \sum_t \frac{\partial u_t'}{\partial \beta} \Sigma^{-1} \frac{\partial u_t}{\partial \beta} \right)(\beta - \beta_k) \)

Setting this to zero and solving for \( \beta \) puts this into the general “hill-climbing” framework (if minimization is converted to maximization) from Section 7.3 with

(9) \( G = \left( \sum_t \frac{\partial u_t'}{\partial \beta} \Sigma^{-1} \frac{\partial u_t}{\partial \beta} \right)^{-1} \)
Chapter 7: Non-Linear Estimation

\( G \) ends up being the estimate of the covariance matrix of the estimates. If you use ROBUSTERRORS, the recomputed covariance matrix is

(10) \( G \text{mcov}(v,1)G \), where

(11) \( v_t = \frac{\partial u'}{\partial \beta} \Sigma^{-1} u_t \) (the summands from (7))

GMM Estimation

GMM estimation is done by similar methods to multivariate least squares—the first order conditions of the optimization problem (6) are expanded, ignoring the second derivative of the residuals. The estimated covariance matrix of coefficients is

(12) \( A = \left( \frac{\partial G'}{\partial \beta} [SW] \frac{\partial G}{\partial \beta} \right)^{-1} \)

If you use the ROBUSTERRORS option (with a sub-optimal weight matrix), the covariance matrix becomes \( ABA \), with \( B \) dependent on the options chosen:

With NOZUDEP, ROBUSTERRORS, and LAGS=0:

(13) \( B = \left( \frac{\partial G'}{\partial \beta} [SW] (\Sigma \otimes Z'Z) [SW] \frac{\partial G}{\partial \beta} \right) \)

With ZUDEP, ROBUSTERRORS, and LAGS=0, or NOZUDEP, ROBUSTERRORS, and LAGS>0

(14) \( B = \left( \frac{\partial G'}{\partial \beta} [SW] \text{mcov}(Z \otimes u,1) [SW] \frac{\partial G}{\partial \beta} \right) \)

Example

We can generalize Example 7.2 to handle multiple returns as follows. Suppose that we have series REALRET1, REALRET2 and REALRET3. We set up one FRML for each return. We use the INSTRUMENTS and ZUDEP options on NLSYSTEM. With about 110 data points, this could run into the problem described on the last page when we get to four lags on the instruments (and so 51 moment conditions).

```
nonlin discount riskaver
frml h1 = discount*realret1(t)*consgrow(t)**riskaver-1
frml h2 = discount*realret2(t)*consgrow(t)**riskaver-1
frml h3 = discount*realret3(t)*consgrow(t)**riskaver-1
compute discount=.99,riskaver=-.95
do for nlag = 1 2 4
    instruments constant consgrow{1 to nlag} realret1{1 to nlag} $ 
    realret2{1 to nlag} realret3{1 to nlag}
    nlsystem(instruments,zudep) / h1 h2 h3
    cdf chisqr %uzwzu 12*nlag+1
end dofor
```
Example 7.3 Non-Linear Systems Estimation

This uses `NLSYSTEM` to estimate equations from an expenditure system given in Theil (1971), Section 7.6. It works with four groups of commodities: Food, Vice, Durable Goods and Remainder. Data for each consists of its value share \(x\) (percent) change in quantity \(\left(WQ_{it}\right)\) and the (percent) change in price \(\left(DP_{it}\right)\). For commodity group \(i\), we have the equation

\[
WQ_{it} = u_i DQ_{it} + \sum_j \pi_{ij} DP_{jt} + \epsilon_{it}
\]

This system of equations is redundant, as the sum of the \(\epsilon\) over \(i\) must be zero. We thus estimate only three of the four equations. We are interested in two constraints:

\[
\sum_j \pi_{ij} = 0 \quad \text{(homogeneity)}
\]

\[
\pi_{ij} = \pi_{ji} \quad \text{(symmetry)}
\]

To simplify the former, we parameterize the equations using an “excess” parameter for \(\pi_{i4}\). This parameter has a value of 0 if constraint (2) holds.

We use different `PARMSETS` to handle the different problems. For instance, the `PARMSET “RELAX”`, when added in with the base parameter set relaxes the homogeneity assumptions by allowing the \(\pi_{i4}\) to be estimated. The `PARMSET “SYMMETRY”`, when added to the others, imposes the symmetry constraints. This example is on the file `CONSUMER.PRG`.

cal(a) 1922
all 1962:1
open data consumer.wks
data(format=wks,org=cols)
set dq = wqfood + wqvice + wqdura + wqrem
nonlin(parms=base) mu1 mu2 mu3 p11 p12 p13 p21 p22 p23 p31 p32 p33
nonlin(parms=relax) ep14 ep24 ep34
nonlin(parms=symmetry) p12=p21 p13=p31 p23=p32
* frml ffood wqfood = mu1*dq+p11*(dpfood-dprem)+$
p12*(dpvice-dprem)+p13*(dpdura-dprem)+ep14*dprem
frml fvice wqvice = mu2*dq+p21*(dpfood-dprem)+$
p22*(dpvice-dprem)+p23*(dpdura-dprem)+ep24*dprem
frml fdura wqdura = mu3*dq+p31*(dpfood-dprem)+$
p32*(dpvice-dprem)+p33*(dpdura-dprem)+ep34*dprem
* compute mu1=mu2=mu3=0.0
compute p11=p12=p13=p21=p22=p23=p31=p32=p33=0.0
compute ep14=ep24=ep34=0.0
nlsystem(parmset=base) / ffood fvice fdura
nlsystem(parmset=base+relax) / ffood fvice fdura
compute ep14=ep24=ep34=0.0
nlsystem(parmset=base+symmetry) / ffood fvice fdura
7.10 More General Maximization

The Instruction MAXIMIZE

MAXIMIZE is designed for many of the estimation problems which specialized instructions like LINREG and NLLS cannot handle. Its primary purpose is to estimate (single) equations by maximum likelihood, but it is more general than that.

MAXIMIZE is perhaps most often used for estimating the more esoteric variations of ARCH and GARCH models that the built-in GARCH instruction can’t handle. These models are covered in detail in Chapter 12.

The problems that MAXIMIZE can solve are those of the form

\[
\max_\beta \sum_{t=1}^{T} f(y_t, X_t, \beta)
\]

where \(f\) is a RATS formula (FRML). Things to note about this:

- MAXIMIZE only does maximizations (directly), but can, of course, do minimizations if you put a negative sign in front of the formula when you define it.
- MAXIMIZE does not check differentiability and will behave unpredictably if you use methods BFGS or BHHH with a function that is not twice-differentiable. Differentiability is not an issue with the SIMPLEX and GENETIC methods.

If even this is too narrow for your application, you will need to try the instruction FIND—see Section 7.11.

Setting Up for MAXIMIZE

The steps in preparing to use MAXIMIZE are the same as for NLLS (Section 7.7):

- Set the parameter list using NONLIN.
- Set up the function using FRML.
- Set initial values for the parameters, using COMPUTE or INPUT. In some cases, such as recursive ARCH/GARCH models, you will also need to initialize one or more series used to hold values such as residuals or variances.

Most of the ideas described in Section 7.6 for simplifying the specification of a model through the use of “sub-FRMLs” and separate PARMSETs are most likely to be used when the final FRML is to be estimated using MAXIMIZE.
Example

Example 17.6 in Greene (2003) estimates a generalization of a Cobb-Douglas production function:

\[ \log y_i + \theta y_i = \beta_1 + \beta_2 \log K_i + \beta_3 \log L_i + \varepsilon_i, \varepsilon_i \sim N(0, \sigma^2) \text{i.i.d.} \]

Without the \( \theta y_i \) term, this would be a standard Cobb-Douglas function, and could be estimated by least squares. With the extra term included, the likelihood function will have to include a Jacobian term. Because the Cobb-Douglas model itself is estimated with \( \log y_i \) as the dependent variable, it makes sense to maintain the likelihood in terms of \( \log y_i \) so the Jacobian can be seen to be

\[ \frac{\partial \varepsilon_i}{\partial (\log y_i)} = 1 + \theta y_i \]

To do maximum likelihood, we need to maximize

\[ \sum \log(1 + \theta y_i) + \log f_N(\varepsilon_i | \sigma^2) \]

where \( f_N(\varepsilon_i | \sigma^2) \) is the Normal density given mean zero and variance \( \sigma^2 \). Note that Greene includes an extra term of \( -\log y_i \). That converts this to the log likelihood of \( y \), not \( \log y \). Because the extra term doesn’t interact with the parameters, it has no effect on the estimates—only on the value of the log likelihood.

Equation (3) is in the proper form for MAXIMIZE since it’s a sum across observations. The function \%LOGDENSITY (VARIANCE, X) computes the required Normal density. We need to estimate five parameters: \( \beta_1, \beta_2, \beta_3, \theta, \sigma^2 \). An obvious source for initial guess values is the (log) linear Cobb-Douglas model, which will give us the maximum likelihood values of all the parameters given \( \theta = 0 \). Use COMPUTE to copy values out of the LINREG statistics into the parameters. To make this easier to read, the definition of \( \varepsilon_i \) is done with a sub-FRML. See the file MAXIMIZE.PRG for the complete program.

```
linreg logy
# constant logk logl
*
nonlin b1 b2 b3 theta sigsq
frml resid = logy+theta*y-b1-b2*logk-b3*logl
compute b1=%beta(1),b2=%beta(2),b3=%beta(3),$
   theta=0.0,sigsq=%sigmasq
frml loglike = log(1+theta*y)+%logdensity(sigsq,resid)
```

Note that some simple forms of Maximum Likelihood (including this one) can be done more efficiently using NLLS with the JACOBIAN option. See MAXIMIZE.PRG.
Multivariate Likelihoods

You can use `MAXIMIZE` to estimate multivariate Normal likelihood functions using the `%LOGDENSITY` function. (%LOGDENSITY can be used for both univariate and multivariate Normals). The log likelihood element for an \( n \)-vector at time \( t \) is, in general,

\[
-\frac{n}{2} \log 2\pi - \frac{1}{2} \log |\Sigma_t| - \frac{1}{2} u_t^T \Sigma_t^{-1} u_t
\]

This is computed by the function `%LOGDENSITY(SIGMA,U)` where SIGMA is the covariance matrix and \( U \) the vector of deviations from the means of the components. (Note that this, and all other density functions computed by RATS, include the integrating constants so models estimated using different distributions will give comparable likelihood values).

The biggest problem is creating the FRML which will allow you to compute this as the final step. It is often a good idea to create a FRML [VECTOR] to compute \( U \) and a FRML [SYMMETRIC] for \( \Sigma \). Of course, if \( \Sigma \) doesn’t depend upon time, you don’t need to create a FRML just to compute it. If all parameters of \( \Sigma \) are freely estimated, you can even `DECLARE` and `DIMENSION` it and add it to the nonlinear parameter set as a SYMMETRIC array.

The following adds on to Example 7.3 to estimate by maximum likelihood. Because \( \Sigma \) is unconstrained, this should give identical answers to the first `NLSYSTEM` instruction. As just mentioned, the covariance matrix is put in to the parameter set `SIGMAPARMS` as an unrestricted 3x3 matrix.

```
dec frml[vector] ufrml
frml ufrml = ||wqfood-ffood,wqvice-fvice,wqdura-fdura||
dec symm sigma(3,3)
nonlin(parmset=sigmaparms) sigma
frml mvlikely = %logdensity(sigma,ufrml)
compute sigma=%identity(3)
maximize(parmset=base+sigmaparms,iters=400) mvlikely
```
7.11 Using FUNCTION

While \texttt{FRLMs} are general enough to handle a very wide range of non-linear optimizations through the instructions like \texttt{NLLS} and \texttt{MAXIMIZE}, there are others that fit into one of those general frameworks, but have a function which is too complicated to be computed using a single formula. Perhaps you need a numerical integral, or a less standard transcendental or probability density function which can be computed using a sequence of RATS instructions, but can't be done with just one. In those situations, you can use a \texttt{FUNCTION} to do the required calculation. A \texttt{FUNCTION} is a sequence of RATS instructions which takes a set of parameters (perhaps empty) and produces a return value.

The following, for instance, returns the “digamma” function (or a very good numerical approximation of it) of its argument \( z \). The digamma is the derivative of the log \( \Gamma \) function. Since this example was originally written, we have added a built-in \%DIGAMMA function to RATS for doing this computation. However, we think the techniques demonstrated in this example program may be helpful.

```r
function %DiGamma z

* 
if z<=0 {
    compute %DiGamma=%NA
    return
}
compute zp=z
compute %DiGamma=0.0
while (zp<30) {
    compute zpr=1/zp
    compute %DiGamma=%DiGamma-zpr
    compute zp=zp+1
}
compute zpr=1/zp
compute %DiGamma=%DiGamma+log(zp)+$
    zpr*(-.5+zpr*(-1.0/12.0+zpr**2*(1.0/120.0-zpr**2/252.0)))
end
```

Once you’ve executed these program lines, you can use \%digamma\(\text{(argument)}\) to get the digamma function of the argument. We generally suggest that, if you have a function that you’re likely to use in a number of programs, that you name it with a % as the lead character, just as we do with almost all of the functions defined by RATS. This reduces the chance that you’ll run into a name conflict with variables in later programs.

The \texttt{COMPUTE %DiGamma} instructions are the ones that set the return value. Whatever is the last value assigned to the function name before a \texttt{RETURN} or the \texttt{END} of the function is the value the function returns. For instance, here we return %\texttt{NA} if the argument is out of range (non-positive) right at the start; otherwise, a recursion process is used to push the argument to a level where a simple series expansion can be employed.
If this were intended as a “production” version of the function, there would be a minor change which would be handy. This uses the two variables \(ZP\) and \(ZPR\) in addition to function name \(%\text{DIGAMMA}\) and argument \(Z\). As this function is written, those would go into the regular symbol table, where they might conflict with names used in a main program. A slight improvement would be to add the line \texttt{LOCAL REAL ZP ZPR} right after the \texttt{FUNCTION} line. This makes those two variables “local” to the function, so you don’t have to worry about hitting a name in the caller. \texttt{LOCAL} is like \texttt{DECLARE}, but is used only within a \texttt{FUNCTION} or \texttt{PROCEDURE}. Local variables are described in Section 16.2.

The following uses the function above to estimate the parameters of a two parameter gamma model for the series \(y\):

\begin{verbatim}
set ysq = y**2
set logy = log(y)
set recy = 1/y
*nonlin p lambda
\end{verbatim}

\textit{The following conditions, when summed over the data, should yield zero. Any pair will give an exact solution, though each pair will give a DIFFERENT solution, due to sample variation.}

\begin{verbatim}
frml yfrml = y-p/lambda
frml ysqfrml = y**2-p*(p+1)/lambda**2
frml recyfrml = 1/y-lambda/(p-1)
frml logyfrml = log(y)-%digamma(p)+log(lambda)
*compute p=2,lambda=1.0
nlsystem(instruments,nosigma) / yfrml ysqfrml recyfrml logyfrml
\end{verbatim}

When you write a function, RATS assumes that the function value and the parameters are all real-valued. They can, however, take other types. You indicate this with the \texttt{TYPE} instruction, which needs to be included early in the function code, before any instructions which actually do anything. The following returns the sum of the logs of the diagonal elements of a \texttt{SYMMETRIC} matrix.

\begin{verbatim}
function %sumlogdiag a
type symmetric a
local integer i
compute  %sumlogdiag=0.0
do i=1,%rows(a)
    compute %sumlogdiag=%sumlogdiag+log(a(i,i))
end do i
end
\end{verbatim}

Note, by the way, that this could be done without \texttt{FUNCTION} using built in expressions: \%SUM(\%LOG(\%XDIAG(A)). In general, it’s better to use the built-in functions where possible.
Example 7.4 Using FUNCTION

Suppose that we want to estimate a yield-curve using data from many bonds. For each bond, we have the following information:

- Periods to maturity (series MATURITY).
- The coupon (series COUPON).
- The current market value (series VALUE).

We need to estimate the parameters of a function which discounts the future stream of payments, minimizing (in some fashion) the gaps between the present values predicted by the model and the actual market values. We’d like to use NLLS, but the present value for a bond requires a sum over all the coupons due until maturity. We can’t use annuity formulas for this because the interest rate isn’t fixed. So we create a FUNCTION, which takes as its argument the number of the bond, and uses the data in the three series described above to compute the present value of the bond under whatever are the parameter settings being evaluated.

In order to use NLLS, we still need a FRML which gives the value for each entry. Here, all this FRML needs to do is return the value of the function for entry T.

We will model the function \( \delta(T) \), which gives the present value of $1 at T, by

\[
\delta(T) = \exp\left(-T\left(a_0 + a_1 T + a_2 \max(0, T - C)\right)\right)
\]

The term multiplying \(-T\) is a piecewise linear function with a join point at \(C\).

This is example BONDS.PRG.

```plaintext
compute nbonds=20
all nbonds
open data bonds.xls
data(format=xls,org=cols)

The data set consists of US Treasury bonds, which have semi-annual coupons, but with the coupon stated in annual terms. The maturity series also provides annual information. The coupon is divided by two and the maturity multiplied by 2 to give values for the actual coupon period.

set coupon   = coupon / 2
set maturity = maturity * 2
* nonlin a0 a1 a2
compute a0=.030,a1=a2=0.0
compute cusp=2.0
*```
function BondPV bond
  type real BondPV
  type integer bond
  *
  local real mdate cdate
  *
  compute mdate=maturity(bond)
  compute BondPV=100.0 * $
    \exp(-mdate*(a0+mdate*a1+%max(mdate-cusp,0.0)*a2))

Walk backwards through the coupons.

compute cdate=mdate 
while (cdate > 0.0) {
  compute BondPV=BondPV + coupon(bond) * $
    \exp(-cdate*(a0+cdate*a1+%max(cdate-cusp,0.0)*a2))
  compute cdate=cdate-1
}

Adjust for simple interest payable by the purchaser for the initial coupon. CDATE will be –(fraction of period).

compute BondPV=BondPV+coupon(bond)*cdate
end
*
frml BondPrice value = BondPV(t)
*
nlls(robusterrors,frml=BondPrice) value

Graph the estimated yield curve from maturities of 0->10 periods (5 years). After the present value function is evaluated, the maturity is switched to years and the annual yield computed.

set testm 1 50  = .20*t
set pvalue 1 50 = $
  \exp(-testm*(a0+testm*a1+%max(testm-cusp,0.0)*a2))
set testm 1 50  = testm/2
set yield 1 50  = -log(pvalue)/testm
scatter(style=line,window="Yield Curve")
# testm yield
7.12 Using FIND

Using FIND

\textit{FIND} is the “catch-all” optimization instruction. If you can’t do it any other way, you can probably do it with \textit{FIND}. Its great advantage is that it can optimize an expression which takes several RATS instructions to calculate and which doesn’t fall into one of the forms treated by the existing optimization instructions. For instance, we have used it to estimate “hyperparameters” in Kalman filter problems.

\textit{FIND} works quite differently from the other estimation instructions. The \textit{FIND} instruction itself controls a set of other instructions, which are the ones which actually compute the function value. The syntax is

\texttt{find(options) maximum/minimum/root expression}

statement block: instruction or block of instructions used in computing \texttt{expression}

end find

Like the other non-linear estimation instructions, the options include \texttt{METHOD}, \texttt{ITERS} and \texttt{CVCRIT}. The default method, however, is \texttt{SIMPLEX} (Section 7.3).

Just as with the other estimation instructions, you need to put your free parameters into a \texttt{PARMSET} using \texttt{NONLIN} and give them initial guess values. The main difference, though, is that the function being optimized is supplied by you, through the \texttt{expression} on \texttt{FIND} and an instruction or set of instructions which are used to compute it. Typically, \texttt{expression} will just be a real variable which will be set by those instructions. If so, you also need to \texttt{DECLARE} that variable, since RATS won’t see the instruction which sets it until after the \texttt{FIND}.

For example, in the production function in Section 7.10, given \( \theta \), all the other coefficients can be estimated by a least squares regression. The following uses \texttt{FIND} to search for the maximum likelihood \( \theta \) as an alternative to estimating all parameters simultaneously using \texttt{MAXIMIZE}. The regression part of the likelihood is concentrated into the single value \texttt{%LOGL} which is then adjusted by the sum of the log Jacobians. \texttt{FIND} controls a loop much as \texttt{DO} does. The instructions between \texttt{FIND} and \texttt{END} are executed each time a new function evaluation is needed.

\begin{verbatim}
nonlin theta
find maximum %logl
    set yadjust = log(y)+theta*y
    linreg(noprint) yadjust
    # constant logk logl
    sstats / log(1+theta*y)>>yterms
    compute %logl=yterms+%logl
end find
\end{verbatim}
7.13 Troubleshooting

If all the functions which you were trying to maximize were globally concave and computable over the entire parameter space, there would be little need for this section. Unfortunately, that’s not the real world. If you do any significant amount of work with non-linear optimization, you will run into occasional problems. Make sure that you understand the effects of numerical precision and initial guess values described in 7.1. Of course, almost all problems are due to “bad” initial guess values, in the sense that if you could start out at the optimum, everything would be simple. And many problems can be solved by coming up with better settings for these.

If you haven’t tried using preliminary simplex iterations (Section 7.3) before using BFGS or BHHH, we would suggest that you start with that. If you have, and you still are having problems, read further.

“Missing Values And/Or SMPL Options Leave No Usable Data Points”

RATS will produce this message if it can’t find any data points at which it can compute an objective function. In some cases, there is a simple fix. For instance, if you are maximizing a function which includes a term of log(sigsq) and you haven’t given sigsq an initial guess value, the standard initialization value of zero will leave the function uncomputable. If you did this, you should see a warning like

```markdown
## NL6. NONLIN Parameter SIGSQ Has Not Been Initialized. Trying 0
```

These warnings usually are of little consequence, since zero is, for many parameters, a quite reasonable initial guess value. If, however, you’re having missing value problems, be careful about this.

The other common source for this problem is a failure to set up a recursively defined function properly. If you have an ARCH/GARCH model or one with moving average errors, you need to give a hard value for the start parameter since the recursion has to have a known initial point. See the “Recursive FRML’s” part of Section 7.6 if you need more help.

“Subiterations Limit Exceeded”

This message is issued when the subiteration process described in Section 7.2 fails to produce a step size which meets the criteria described there. This is a very serious problem—the predicted rate of change of the function in the chosen direction is quite wrong. If this occurs on a very early iteration, it’s possible that increasing the subiterations limit from its default value of 30 will fix the problem. However, you’re more likely to have success in this situation by doing some preliminary simplex iterations (PMETHOD=SIMPLEX, PITERS=5 for instance) to improve the initial guess values.

If it occurs later in the iteration process, it is usually the result of one of two problems:

1. Estimates at or near the boundary of the computable zone
2. Collapse to near-singularity of the Hessian
If you have a function value which can’t be computed over some region (a variance
goes negative, a covariance matrix becomes singular, a probability exceeds 1) and the
estimates have moved close to that region, it may be hard for general hill-climbing
methods to work because the function value may be going from computable to
uncomputable over a short distance. If this is what has happened (check the coeffi-
cient values on the TRACE output), you might be able to push the estimates away by
restoring the estimation with simplex iterations. If it still doesn’t move, you’ve
probably found at least a local maximum at the boundary. You’ll need to try the
entire process from a different set of initial guess values to see if there is a higher
global optimum away from the boundary.

The collapse to near-singularity of the Hessian is the more likely culprit if you’re
using METHOD=BFGS. The instruction DISPLAY %CVTOCORR(%XX) (which displays
the last inverse Hessian as a correlation matrix) will likely show you some off-
diagonal values very near 1.0 (such as .9999). In some cases, you may be able to fix
this by just re-executing the instruction, which will reinitialize the BFGS process. If,
however, you have some parameters which are very highly correlated, this will
probably quite quickly reproduce the problem. If that’s the case, you may need to find
a different (equivalent) parameterization. Concentrating out a parameter or param-
eters, can, where possible, help out quite a bit. And substitutions like
\( bx^p = b' \left( x / x_0 \right)^p \)
and \( \alpha + \varphi x_{i-1} = \alpha (1 - \varphi) + \varphi x_{i-1} \), generally produce better behavior in the parameter
pairs involved.

### Slow Convergence (Many Iterations)

The more parameters that you try to estimate, the more iterations you’re likely to
need. If you have a function which seems to be making very slow progress, check the
TRACE output, paying particular attention to the distance scale on the subiterations:

\[ \text{Subiterations 2. Distance scale } 0.500000000 \]

If you’re consistently seeing distance scales of 1.0 or 0.5, you may just need to be
patient. Numbers like those mean that the predictions for the behavior of the func-
tion value seem to be accurate. If, on the other hand, you’re seeing short steps like

\[ \text{Subiterations 5. Distance scale } 0.062500000 \]

you may need to rethink the parameterization of the model as described above.

You can sometimes improve the behavior of the estimates by doing small sets of
iterations on subsets of the parameters. Something like

\[
\begin{align*}
\text{maximize}(\text{iters}=20,\text{parmset}=\text{pset1},\text{noprint}) & \text{ maxfrml} \\
\text{maximize}(\text{iters}=20,\text{parmset}=\text{pset2},\text{noprint}) & \text{ maxfrml} \\
\text{maximize}(\text{parmset}=\text{pset1+}\text{pset2},\text{noprint}) & \text{ maxfrml}
\end{align*}
\]

The first MAXIMIZE will do 20 iterations of the parameters in PSET1 given the initial
guess values in PSET2. The second will do 20 iterations on PSET2 given the refined
values of \texttt{PSET1}. Then the final \texttt{MAXIMIZE} will estimate the full set. Because the subiterations apply the same step size multiple to the entire parameter set, the overall process may be slowed down by a few poorly estimated parameters.

**Slow Convergence (Time)**

If each iteration takes a long time, there are two possible reasons: the function evaluations themselves are slow, or there are a large number of parameters. You could, of course, be dealing with both. If you need to speed up your function evaluations, look for situations where you do redundant calculations.

- If your \texttt{FRML} or \texttt{FUNCTION} includes a calculation which doesn’t depend upon the parameters, do it once and save the results.
- If you have a calculation which doesn’t depend upon time, but does depend upon the parameters, use the \texttt{START} option on the estimation instruction and compute it in that.
- Within a \texttt{FRML}, avoid making multiple references to the same sub-\texttt{FRML}. For instance, if \texttt{RESIDA} is a \texttt{FRML}, the following are equivalent, but the second computes \texttt{RESIDA} just once:

\begin{verbatim}
frml frontier = log(theta) - .5*resida**2 + $
log(\%cdf(-lambda*resida))

frml frontier = alpha=resida, $
log(theta) - .5*alpha**2 + log(\%cdf(-lambda*alpha))
\end{verbatim}

If the size of the parameter set is what is slowing things down, you might want to think about the suggestion above for using subsets of the parameters. This will give you the biggest improvement in speed if you can optimize over a less complicated formula for different subsets.

**Zero Standard Errors**

If the model seems to converge reasonably, but your output shows that some coefficients have (true) zero standard errors, this will almost always be because that coefficient doesn’t actually affect the function value. Most commonly, this is just a simple error of including a coefficient doesn’t appear in the formula(s) being estimated. More subtly, it can be a parameter which multiplies a series or subcalculation which takes a zero value.
Chapter 8
Introduction to Forecasting

This chapter provides an introduction to forecasting with RATS, including the various types of forecasting models and the instructions used to generate forecasts. It also discusses ways to store and graph forecasted values, produce forecasts using rolling regressions, compute forecast errors, generate Theil U statistics, and how to troubleshoot forecasting problems.

Chapter 9 provides additional information on univariate forecasting methods, including Box–Jenkins ARIMA models, exponential smoothing techniques, and spectral methods. See Chapter 10 for more information on forecasting VAR models. Chapter 11 discusses issues related to simultaneous equations and model solving.
8.1 Introduction

Forecasting With RATS

This chapter provides an overview of the techniques and models available in RATS, introduces the instructions you can use to produce forecasts, and discusses some important issues related to producing good forecasts. Most of the information in this chapter applies to all the methods available in RATS, and is recommended reading for anyone doing forecasts with any type of model. See Chapter 9 for additional details on ARIMA, exponential smoothing, and spectral techniques, Chapter 10 for Vector Autoregressions (VAR’s) and Chapter 11 for Simultaneous Equations.

The main forecasting instructions are `FORECAST` and `UFORECAST`. `UFORECAST` (Univariate FORECAST) is the simplest of these—as the name implies, it is used only for forecasting (linear) univariate models. `FORECAST` can handle models with many equations, and with nonlinearities. You can also use `PRJ` for forecasting certain simple types of models, though its main use is analyzing the in-sample properties of a fitted model.

Regression Models

Consider the basic regression model:

\[ y_t = X_t \beta + u_t \]

Such a model is generally not a self-contained forecasting model unless it is a simple regression on deterministic trend variables and/or lags of the dependent variable. Otherwise, to forecast the future values of \( y \), we either need to know the future values of \( X \), or the forecasts must be made conditional upon some assumed values for them. It’s also possible to “close” the model by constructing additional equations which will forecast the future \( X \)’s.

As a simple example, suppose that we are going to forecast a series of orders using a regression on a linear trend. We have data through 2007:6, but want forecasts out to 2008:12. In order to do this, we need to define the trend series out to 2008:12.

```plaintext
data (format=xls, org=columns) 1986:1 2007:6 orders
set trend 1986:1 2008:12 = t
linreg orders
# constant trend
prj forecast 2007:7 2008:12
```
Models With Lagged Dependent Variables

If the right hand side of your model includes lags of the dependent variable (that is, if $X$ in equation (1) contains lagged values of $y$), the model is referred to as a *dynamic* model. One example would be the simple autoregressive model:

$$y_t = \alpha + \beta y_{t-1} + u_t$$

For such models, you have the choice between computing *static* forecasts or *dynamic* forecasts. (These terms are also discussed in Section 1.1.9).

*Static forecasts* are computed as a series of one-step-ahead forecasts, using only actual values for lagged dependent variable terms. You can only compute static forecasts in-sample (or up to one period beyond the end of the sample), because you must have actual data available for the lagged dependent variable terms. Static forecasting will always produce the same forecasted values for a given time period $T$, regardless of the point in time at which you start computing forecasts.

*Dynamic forecasts* are multi–step forecasts, where forecasts computed at earlier horizons are used for the lagged dependent variable terms at later horizons. For example, the forecasted value computed for time $T$ will be used as the first–period lag value for computing the forecast at time $T+1$, and so on.

For dynamic forecasts, use `UFORECAST` or `FORECAST`, which do dynamic forecasting by default. If you want static forecasts instead, you can use the `STATIC` option on `UFORECAST` or `FORECAST` or, for simple linear models, use the `PRJ` instruction. Note that if you are only forecasting one step ahead, the two methods will produce the same result.

If your model is itself static, that is, it has no lagged dependent variables, you will get identical results with any of the methods just described.

Time Series Models

In “pure” time series models, future values of a series (or group of series) depend *only* on its (their) past values. This category includes Box–Jenkins (ARIMA), exponential smoothing, and spectral models, as well as VAR systems that do not include other exogenous variables. For all except exponential smoothing and spectral methods, you can use `FORECAST` to produce dynamic forecasts or, with the `STATIC` option, one-step-ahead static forecasts (in–sample only). (`UFORECAST` can be used for single equations).

An ARIMA model with moving average terms also requires lags of the residuals for forecasting. For instance, if

$$y_t = \alpha + \phi y_{t-1} + u_t + \theta u_{t-1}$$

in order to forecast period $T+1$, a value is needed for the $u_T$ ($u_{T+1}$ is a post-sample shock and is set to zero). When you use `BOXJENK` to estimate an ARIMA model, it will automatically save these.
8.2 Producing Forecasts

Static Forecasting with PRJ

The instruction `PRJ` can be used to compute fitted values or forecasts for a linear model after you estimate the model with `LINREG`, `STWISE`, `BOXJENK` or `AR1`. They are calculated using the most recently estimated regression. `PRJ` does static forecasts only, and so can only be used out-of-sample (for more than one period) if your model itself is static. An example of such a use of `PRJ` is provided on page 298.

Using UFORECAST and FORECAST

For more general forecasting of linear and non-linear models, use `UFORECAST` or, for multi-equation models or additional flexibility, `FORECAST`. As noted earlier, these produce dynamic forecasts by default—use the `STATIC` option to get static forecasts. Remember, for models that do not have a dynamic structure (that is, they have no lagged dependent variables), either method will produce the same results. The next paragraphs discuss the use of the instructions in various situations.

Single-Equation Linear Models

If you are estimating your model with `LINREG`, use the `DEFINE` option to save the estimated equation, which can then be forecasted:

```
set trend 1986:1 2008:12 = t
linreg(define=ordereq) orders
# constant trend
```

You can use the Single-Equation Forecasts wizard on the Data menu to generate the forecasting instruction, or you can type the command directly. For example:

```
uforecast(equation=ordereq) fore_orders 2007:7 2008:12
```

or

```
forecast(steps=18) 1
# ordereq  fore_orders
```

The parameters for the `UFORECAST` are the series for the forecasts (`FORE_ORDERS`), and the range. The option `EQUATION=ORDEREQ` gives the equation to use.

The basic syntax of `FORECAST` is shared by several other instructions (`IMPULSE`, `ERRORS`, `SIMULATE` and `HISTORY`). The `STEPS` option indicates that we want to forecast for 18 steps. We could also use the options `FROM=2007:7`, `TO=2008:12` to set the forecast range. The supplementary card supplies the equation to forecast (`ORDERREQ`), and the series into which the forecasts are saved (`FORE_ORDERS`).

You can also use the `SMPL` instruction to specify the forecast range. Note that this changes the default range for any subsequent instructions, so be careful using it.

```
smpl 2007:7 2008:12
forecast 1
# ordereq  fore_orders
```
AR1 Models

The **DEFINE** option on the **AR1** instruction produces a linearized equation from the original non–linear model:

```r
open data commfore.xls
data(format=xls,org=cols) 1986:1 2006:12 tbill3mo ipsteel
ar1(define=areq) orders
# constant  tbill3mo{0 1}  ipsteel
uforecast(equation=areq) forecast 2007:1 2008:12
```

This reads actuals plus forecasts for series **TBILL3MO** and **IPSTEEL**, fits an AR1 model, and forecasts the model for 24 steps starting in 2007:1.

Multiple Equation Systems

**FORECAST** can also be used to produce forecasts for systems of equations. The equations are usually defined using either **EQUATION** instructions or the **SYSTEM** instruction and then estimated using **ESTIMATE** or **SUR**. However, your system might consist of equations estimated using several **LINREG** instructions, or it might include identity equations (see the next page for details) or other equations whose coefficients you have set yourself using **EQUATION** or **ASSOCIATE**.

The forecasting instructions don’t “care” how the equations in the model were defined or estimated. As long as the necessary data are available and the equation coefficients have been estimated or set, you can produce forecasts.

To forecast a system of equations, you can use the **VAR (Forecast/Analyze)** wizard on the **Statistics** menu to generate the appropriate instruction, or type in the **FORECAST** instruction directly. If using **FORECAST** directly, you can either:

- list each of the equations in the system on a separate supplementary card, or
- group the equations into a **MODEL** and use the **MODEL=name** option to forecast the model. If you create the equations using a **SYSTEM** instruction, you can use the **MODEL** option on **SYSTEM** to automatically group those equations into a **MODEL**. Otherwise, use the **GROUP** instruction to construct the **MODEL** from a list of existing equations.

For example:

```r
forecast(from=2007:8,to=2008:12) 3
# saleseq fsales
# orderseq forders
# indexeq findex
```

or

```r
group ourmodel saleseq>>fsales orderseq>>forders indexeq>>findex
forecast(model=ourmodel,from=2007:8,to=2008:12)
```

Note that a **MODEL** can include non–linear formulas as well as equations.
Non-Linear Models

Non-linear models are constructed using formulas (FRMLs). These are typically defined using the instruction FRML, although they can also be created using LINREG or EQUATION. You can estimate them using NLLS, NLSYSTEM, or MAXIMIZE.

To generate forecasts for a non-linear model, you must group them into a model and use the MODEL option on FORECAST or STEPS—you cannot list FRMLs on supplementary cards. This is true even when forecasting a single formula.

Using Identities

In some cases, you may need or want to include identities in your forecasting model. You might have accounting identities in a simultaneous equations model (see also Chapter 11), or, if you are forecasting a model estimated in logs, you might want to include an identity FRML to generate levels from the logs.

Identity equations can be created using the EQUATION instructions, while identity formulas are created using FRML. Identities should be listed after all estimated equations on the supplementary cards, or on the list in a GROUP instruction.

These two instructions define the same identity, one (RSUMID) as a FRML and one (RSUMEQ) as an EQUATION.

```
frml(identity) rsumid rsum = rate+rate{1}
equation(identity,coeffs=||1.0,1.0||) rsumeq rsum
# rate{0 1}
```
8.3 Generating and Using Results

Saving Output from UFORECAST and FORECAST

UFORECAST and FORECAST provide tremendous flexibility for handling output. You can:

- save your forecasts directly into series, which you can then print, graph, write to a data file, or use for further analysis.
- use the PRINT option to display the forecasts in the output window.
- use the WINDOW option to display the forecasts in a spreadsheet-style window. This provides convenient viewing of the forecasts, and also allows you to Copy and Paste the forecasts into spreadsheet programs or other applications, or export the data directly to a file with the File-Export operation.

Saving Forecasts in Series

This is straightforward for UFORECAST. For FORECAST, you can save your forecasts into series in three ways:

- If you are using supplementary cards to list the equations (rather than using the MODEL option), you can simply provide series names for the forecasts as the second parameter on each supplementary card. This example forecasts the equation IRATEEQ and stores the forecasts in a series called OLSFORE:

  ```
  smpl 2007:7 2008:6
  forecast
  # irateeq olsfore
  ```

  If there are several equations, you don’t have to save forecasts for each of the equations. Just leave off the series name parameter on a supplementary card if you don’t need that equation’s forecasts.

- When using the GROUP instruction to create a model, use the “>>series” notation on GROUP to provide the names for the forecast series. When you forecast the model (using the MODEL option), the forecasts will be stored in those series (see the example on page 301).

- You can use the RESULTS option on FORECAST to save the forecasts in a VECTOR of SERIES.

You can save your forecasts into existing series (including the actual dependent variables), or you can have FORECAST create new series (see “Saving Series in a Loop” for an exception). Note that saving your forecasts back into the dependent variable(s) has no effect on the actual forecasts produced, because the forecasts are computed and stored internally before being saved to the target series.
This example computes forecasts for an interest rate series and stores them in a new series called OLSFORE:

```
linreg(define=irateeq) rate 1960:1 1995:8
# constant ip grm2 grppi{1}
smpl 1995:1 1996:2
uforecast(equation=irateeq) olsfore
```

The forecast range intentionally overlaps the estimation range.

And this example forecasts a MODEL, and uses the RESULTS option to store the forecasts in a VECTOR of SERIES called FORECSTS:

```
forecast(model=gdpmod,results=forecsts,steps=24)
```

### Saving Series in a Loop

If you want to use FORECAST inside a loop (or other compiled section, such as a procedure) and want to save the forecasts into new series, you must either use the RESULTS option to store the forecasts, or else create the new series prior to the beginning of the compiled section (using, for instance, DECLARE, SET, or CLEAR).

### The WINDOW and PRINT Options

The WINDOW option allows you to display the forecasts in a spreadsheet–style window. This provides a convenient way to view the results, and it also allows you to Copy and Paste the data directly into a spreadsheet program or other application, or export the data to a file using the File-Export menu operation. The argument of the WINDOW option provides the title for the output window. For example, the following forecasts twenty-four steps of a six variable model, putting the forecasts into a window named “Forecasts.”

```
forecast(model=sixvar,window="Forecasts",steps=24)
```

The PRINT option on FORECAST causes the forecasted values to be displayed in the output window. Both PRINT and WINDOW can be used with any of the other methods of saving or displaying output.

```
forecast(model=gdpmod,results=forecsts,print,steps=24)
```

### Graphing Forecasts

If you save your forecasts into series, you can graph them using the GRAPH or SCATTER instructions. This example forecasts an interest rate model, saves the forecasts in a series called RDIFF_FORE and then graphs both the forecasts and the original series (RDIFF).

```
smpl 1994:3 1996:2
uforecast(equation=arimaeq) rdiff_fore
graph(key=below,header="Monthly Changes in Rate") 2
# rdiff
# rdiff_fore
```
Saving Forecasts to Files

To save your forecasts to a file, you can either:

- store the forecasts into a series and output the series using **COPY** (or **DEDIT**, **STORE**, and **SAVE**), or
- use the **WINDOW** option to display your forecasts in window. You can export the data in one of several formats by doing **File-Export**, or you can use **Edit-Copy** to copy the data to the clipboard for pasting into another application.

For example, this creates a file named **OLSFORE.XLS** which holds the forecasts.

\[
\text{smpl 2007:8 2008:7}
\]

\[
\text{uforecast(equation=irateeq) olsfore}
\]

\[
\text{open copy olsfore.xls}
\]

\[
\text{copy(format=xls,org=cols,dates) / olsfore}
\]

\[
\text{close copy}
\]

Forecasting with Rolling Regressions

Computing forecasts based on a series of rolling regressions (that is, a series of regressions performed over changing sample periods) is a common practice, and an easy one to implement in RATS.

Suppose we have monthly data for actual stock market returns, predicted returns and a yield series for long bonds. The model of interest is

\[
R_t = \alpha + \beta P_{t-1} + \gamma Y_{t-1}
\]

that is, this period's actual return can be predicted with the help of a linear combination of last period's predicted return and the long bond yield. To test whether this actually would work in practice, we want to generate a series of out-of-sample forecasted returns from rolling regressions. Here, we first run the regression through 1999:12, and compute a forecast for 2000:1. Then we estimate through 2000:1 and forecast 2000:2, and so on. Note that we have used the **NOPRINT** option on **LINREG**. We really aren’t interested in seeing almost 90 regressions—we want the forecasts.

\[
\text{compute regstart = 1970:1}
\]

\[
\text{clear rhat}
\]

\[
\text{do regend=1999:12,2007:5}
\]

\[
\text{linreg(noprint) r regstart regend}
\]

\[
\text{# constant pr[1] by[1]}
\]

\[
\text{uforecast rhat regend+1 regend+1}
\]

\[
\text{end do}
\]

At the end of this, the series **RHAT** will have a series of one-step out-of-sample forecasts over the period 2000:1 to 2007:6.
Chapter 8: Forecasting

The last example used the same starting date for all regressions. Suppose you instead want to estimate over a “moving window” of, say, five years. There are various ways to handle the loop indexes to accomplish this—here’s one of them (this program also saves the estimated coefficients from each regression):

```
clear rhat coef1 coef2 coef3
do regend=1999:12,2007:5
   linreg(noprint) r regend-59 regend
      # constant pr{1} by{1}
      compute coef1(regend) = %beta(1)
      compute coef2(regend) = %beta(2)
      compute coef3(regend) = %beta(3)
      uforecast rhat regend+1 regend+1
end do
```

Both these examples compute only one-step-ahead forecasts, so storing the forecasts from each step in the rolling regression is easy. Suppose you instead want to compute twelve-step-ahead forecasts, and save only the twelfth forecast step from each rolling regression. If you simply did:

```
uforecast(equation=foreeq) yhat regend+1 regend+12
```

then forecast step 12 from the first trip through the loop would be overwritten by forecast step 11 from the second trip through the loop, and so on. Thus an additional instruction is needed to copy the forecasts at later horizons to another series.

Here’s one way:

```
clear rhat rhat12
do regend=1999:12,2007:5
   linreg(define=foreeq,noprint) r regend-59 regend
      # constant pr{1} by{1}
   uforecast rhat regend+1 regend+12
   compute rhat12(regend+12)=rhat(regend+12)
end do
```

Another way is to use **FORECAST** with the **SKIPSAVE** option, which is designed for just this sort of situation. For example:

```
clear rhat12
do regend=1999:12,2007:5
   linreg(define=foreeq,noprint) r regend-59 regend
      # constant pr{1} by{1}
   forecast(skip=11,steps=12)
      # foreeq rhat12
end do
```

See “Using the SKIPSAVE Option” in the **FORECAST** section of the Reference Manual for more information.
8.4 Forecast Performance

**Standard Errors of Forecast**

If you use `PRJ` to compute forecasts, you can use the `STDERRS` option to produce a series of the standard errors of projection.

```plaintext
prj(stderrs=sx) fitted
set upper = fitted+sx*1.96
set lower = fitted-sx*1.96
```

These are given by the formula

\[ s \sqrt{1 + x_i (XX)^{-1} x'_i} \]

where \( s \) is the standard error of the regression and \( XX \) is its cross-product matrix. This is a special formula which only works in static forecasting situations. The variance of projection that it computes consists of two parts: one due to the error in the regression equation itself (which would be present even if the coefficients were known with certainty) and one due to the sampling error of the coefficients. In a dynamic forecast, neither of these has such a simple form. Because forecasts are fed forward, the forecast errors from early steps accumulate. And the forecasts at higher steps depend upon the estimated coefficients in a highly non-linear way. How these are computed for a dynamic model is discussed more fully in the Reference Manual in the description of the instruction `ERRORS`.

`UFORECAST` and `FORECAST` also have `STDERRS` options, but these only include the part of the standard error which is produced by the regression equation, so they are computed as if the coefficients were known exactly.

Because `FORECAST` is designed to handle more than one equation, its `STDERRS` option returns a `VECTOR[SERIES]`. This pulls out the forecasts and standard errors for the second equation in the model:

```plaintext
forecast(model=tmodel,steps=5,results=forecasts,stderrs=stderrs)
set lower 37 41 = forecasts(2)+stderrs(2)*%invnormal(.025)
set upper 37 41 = forecasts(2)+stderrs(2)*%invnormal(.975)
```

**One-Step Forecast Errors**

By using the `STATIC` and `ERRORS` options on `FORECAST` or `UFORECAST`, you can easily compute and store a series of forecast errors (the differences between the forecasted value and the actual value) for each period in the forecast range. For a dynamic model, remember that these are one-step forecasts. As with `RESULTS` and `STDERRS`, the `ERRORS` option for `FORECAST` returns a `VECTOR[SERIES]`.
Chapter 8: Forecasting

UFOREERRORS Procedure

This is a procedure for analyzing forecast errors for a single variable. You provide it with the series of actual values and forecasts and it computes a variety of statistics on the errors.

@UForeErrors( options ) actual forecast start end

Parameters

actual series of actual values
forecast series of forecasts
start end range of forecasts to analyze (by default, the range of the series forecast)

Options:

[print]/nopr

title="...descriptive title for forecasts..."
window="...title of window..." (if you want a separate report window)

This computes and defines the following statistics: the mean forecast error (%FERRMEAN), mean absolute error (%FERRMAE), root mean square error (%FERRMSE), mean percentage error (%FERRMPE), mean absolute percentage error (%FERRMAPE) and root mean square percentage error (%FERRMSPE).

The last three are defined only if the actual values are positive throughout the test range. They are also all defined as decimals, not true percentages, that is, they’ll report a value of 0.11 for an 11% error.

THEIL and the Theil U and other Performance Statistics

For larger models, or for multiple step forecasts, you can use the instruction THEIL to produce forecast performance statistics. You use it to compute a series of forecasts within the data range. The forecasts are compared with the actual values and a variety of statistics are compiled from the forecast errors: the mean error, the mean absolute error, root mean square error and a Theil’s U statistic. The last is a ratio of the root mean square error for the model to the root mean square error for a “no change” forecast. This is a convenient measure because it is independent of the scale of the variable. The statistics are compiled separately for each forecast horizon, that is, if you do 12 forecast steps, you’ll get separate information for each variable and for each forecast step from 1 to 12.

THEIL is usually applied to ARIMA models (Chapter 9) or VAR’s (Chapter 10). Its use is a standard part of the methodology for forecasting with VAR’s.
8.5 Comparing Forecasts

Forecasts can be compared informally by using the performance statistics described in the previous section. However, these are subject to sampling error, and so, while one forecast procedure may produce a better RMSE than another in a particular sample, it may not be clear whether that difference is really significant.

Among the procedures which have been proposed for an actual test is that of Diebold and Mariano (1995). This is based upon the use of a loss function, which could be the squared errors, but also could be a loss function tailored to the situation, like an error in the sign, if calling a direction right is more important than the actual forecast value.

The calculation is simple to perform in RATS once you’ve generated the sets of forecasts. Compute the series of the differences between the loss functions for the two forecasts. Regress this on a CONSTANT, requesting robust standard errors with the appropriate number of lags. (If you’re comparing $K$ step forecasts, this would generally be $K-1$). The null is that the intercept is zero. Now, typically the alternative is that one set of forecasts is better than the other (not just that they’re different), so a one-tailed test is appropriate. Note that the series of forecasts should all be for the same forecasting horizon. Don’t mix 1 to $N$ step forecasts in a single test.

For instance, consider the example of the rolling regressions in Section 8.3. That generated one set of one-step forecasts for stock returns (into the series RHAT). Suppose the alternative is the “naive” forecast of the previous period’s return value, and that our loss function is the squared error. The Diebold-Mariano test can be implemented by something like:

```r
set naive 2000:1 2007:5 = r{1}
set d = (r-naive)**2-(r-rhat)**2
linreg(robusterrors) d
# constant
```

If the regression coefficient is positive, then, in this sample, the naive forecast did worse. The correct adjustment from a two-tailed test to a one-tailed test (of equal accuracy vs. naive better than the model) is to double the significance level on the regression coefficient. If the regression coefficient is negative, subtract twice the listed value from 1 to get the one-tailed significance level.

We also have provided a procedure which directly handles this test for the two most important loss functions: squared errors and absolute errors. This is on DMARIANO.SRC. Note that the default on the LWINDOW option for the procedure is TRUNCATED, which follows the recommendation of the authors. However this could produce a non-positive standard error; unlikely in practice, but possible.
8.6 Troubleshooting

Why Don’t I Get Forecasted Values?

At some point, you may discover that you are getting NA’s (missing values) for one or more of the periods for which you expected to get forecasts. Almost without exception, this will be because required right-hand-side values were not available, either due to missing values in the data set, a mistake in specifying the forecast range, or (for simultaneous equation systems) a mistake in the model specification itself.

Consider a simple forecast model where series $y$ at time $t$ depends only on series $x$ at time $t$ plus a constant term, a model you could estimate using:

```
linreg(define=yeq) y
# constant x
```

Remember, if you are forecasting this as a single-equation model (rather than as part of a system), you cannot possibly compute a forecasted value for $Y(t)$ unless the series $X$ already contains a valid number at time $t$. Suppose these are quarterly data, and you have data for $X$ through 2007:2. If you do:

```
uforecast(equation=yeq,print) yhat 2007:3 2007:4
```

the forecasts will be reported as “NA” because $X$ (upon which $Y$ depends) is itself undefined (NA) for these periods. To get out-of-sample forecasts for $Y$, you will need to supply values for $X$ yourself over the forecast range (using `SET` or `COMPUTE` or by reading them in from a data file), or you will need to define and estimate a model for $X$ and compute forecasts, either independently, or as part of a system which also includes the equation for $Y$.

Finding the Problem

First, turn on the `PRINT` option on your forecasting instruction so you can see the forecasts as RATS is calculating them. If these look correct, but the series in which you save the forecasts have NA’s, then you have a problem with the way you are saving or using the forecasts. Make sure that you’re using variable names in the correct positions.

Otherwise, there’s a problem with either your data or the model (or both). Check the following:

1. If you are forecasting a multiple-equation system, make sure your equations have been defined and estimated properly, and that you’ve remembered to include all the required equations on supplementary cards or in the `MODEL`. In particular, make sure that you’ve included any required identity equations.
2. Check your data! Many users have wasted a great deal of time searching for programming mistakes when the real problem was simply missing data in one or more right-hand-side variables. If you are having trouble getting forecast values, do yourself a favor and check the right-hand-side data series carefully, by using the PRINT instruction or double-clicking the series in the series window list to display the actual series used in the model. A common mistake is verifying that original series are OK, but forgetting that transformations (such as log of a negative number) can produce NA’s. Check the data that are actually being used in the model!

3. Check the range for which you are trying to compute forecasts—make sure you aren’t starting your forecasts too far beyond the end of your available data. Suppose you have a simple ARIMA model:

   boxjenk(ar=1,define=areq) y

and you have data for Y through 2006:12. You could produce forecasts out to any horizon if you start forecasting at entry 2007:1 (or earlier). However, if you tried to start at 2007:2 (or later), you would get NA’s at every forecast horizon. That’s because Y at 2007:2 depends on Y at 2007:1 and you don’t have data for that observation.
Chapter 9
Univariate Forecasting

This chapter focuses on forecasting a single variable using time series methods, such as ARIMA modeling, exponential smoothing, and spectral methods. For a general introduction to basic forecasting concepts and the instructions used to produce forecasts, please see Chapter 8. That chapter also provides information on regression-based forecasting models. See Chapter 10 for details on Vector Autoregression models and Chapter 11 for details on simultaneous equations.
Chapter 9: Univariate Forecasting

9.1 Time Series Methods

Background

Time series modeling techniques forecast the future by

1. modeling the correlation between a series (or group of series) and its (their) past,
2. assuming that the relationship between current and past will continue into the future, and
3. computing forecasts on the basis of those assumptions.

They are extrapolative techniques which incorporate no information other than that available in the past data.

Methods

RATS supports three methods for univariate time series forecasting:

**Exponential smoothing**

is a small collection of models which are often adequate for forecasting. Unlike the other methods, it does not really attempt to model the autocorrelation. You choose a model based upon the two most important aspects of time series behavior: trend and seasonality.

**Box-Jenkins**

offers a broad collection of models for fitting the serial correlation pattern. Because of the number of choices permitted, it can handle many types of time series. Choosing an appropriate model, however, is something of an art.

**Spectral forecasting**

uses frequency domain techniques to fit a “generic” Box-Jenkins model. It produces very acceptable forecasts relative to Box-Jenkins models, but is less robust to errors in preparing the data for forecasts. It also requires more data than Box-Jenkins.

You can also forecast using state-space models (done in RATS using the instruction DLM—see Section 12.2), and do multivariate time series forecasting with vector autoregressive models, described in Chapter 10. And you can produce forecasts using a variety of standard regression models. See Chapter 8 for details.

In the remainder of this chapter, we'll examine some issues that are common to all three of these modeling techniques, and then discuss each of them in detail. We will also look at some related topics, including ways to evaluate forecast performance, intervention models, and combined ARIMA/regression models (ARMAX).
9.2 Common Issues

There are several issues which all three of these time series methods must address, and which are examined in this section:

- Should you transform the data prior to the analysis? Usually this boils down to a question of logs vs levels.
- How should you represent the series trend? Should you difference the series?
- How should you treat seasonality?
- How many parameters can you safely estimate given the available data?

Preliminary Transformations

Preliminary transformations are simple transformations with two purposes: to straighten out trends; and to produce approximately uniform variability in the series over the sample range.

What we are trying to eliminate in the second point is a systematic tendency for the magnitude of the variance to depend upon the magnitude of the data. Suppose that the variance of a series is significantly higher when its values are large. If we estimate a model for this series by least squares, the least squares algorithm will concentrate upon fitting the high-variance part, largely ignoring the rest of the data.

Usually, you will either do nothing to the data, take square roots, or take logarithms. How do you decide? The following guidelines should help.

- Do you usually think of the series in terms of its growth rates (GNP, money, prices, etc.)? If so, take logs.
- Is the percent growth rate of the series an almost meaningless concept? Usually, do nothing. Possibly take the square root if the series clearly is more variable at higher values.

Typically, if your decision is not obvious from looking at the data, your choice will make little difference to the forecasts.

Differencing

We will look at this together with the decision of whether or not to include an intercept. This is a crucial decision for two reasons. First, some forecasting methods (especially spectral procedures) are computationally sensitive to mistakes here, particularly to underdifferencing. Second, this determines the trend behavior of the forecasts, so an incorrect choice will have major effects on longer-term forecasts. Modeling methodologies which do not include the correct trend model are likely to produce poor forecasts.

The table on the following page suggests some differencing and intercept combinations for various types of series behavior. Often, you can decide how to difference solely on the basis of your knowledge of the data.
Seasonality

Seasonality poses several serious problems for forecasters:

- It can take many forms: additive or multiplicative, stationary or non-stationary, stochastic or deterministic.
- Seasonality usually accounts for most of the variance of the series, making it difficult to determine the non-seasonal behavior.
- Every method of dealing with seasonality induces degrees of freedom problems. This will either cause a loss of usable data points, or require estimation of seasonal parameters with very few data points.

If seasonality is essential to the task at hand, you have two basic courses of action:

- You can seasonally adjust the data, forecast the adjusted data and the seasonal part separately, and then combine them. The main problem is that the seasonal “model” is done independently and may be inappropriate for the data series.
- You can use a methodology which incorporates seasonality directly. All three of the forecasting methods permit this. However, you will only be able to handle adequately certain forms of seasonality.

Parsimony

While the “Principle of Parsimony” is usually associated with Box and Jenkins, the idea applies to other forecasting models as well. Keep in mind two general principles:

- Don’t ask the data to do too much work. The less data you have, the fewer parameters you can estimate, and the more important your judgment becomes.
- Don’t take your model too seriously. Time series models are designed to fit the serial correlation properties of the data, not explain them. The goal is to find a model which fits the data reasonably well with as few parameters as possible.

The most carefully thought-out model is worthless if you can’t estimate it with the data available. You might think that estimating four parameters with thirty data points is not asking too much of the data. However, experience has shown that a three parameter model which fits almost as well will forecast better most of the time. This is true even if the difference is statistically significant.
9.3 Exponential Smoothing

Background

The exponential smoothing (ES) methodology chooses one from a small group of models which focus upon the trend and seasonal behavior of the data. Because those two aspects dominate the variance of the series, properly chosen ES models perform well relative to more complicated methods on a wide range of data series.

These techniques were actually designed for dealing with data where there is a smooth “signal” obscured by substantial measurement noise. While they can adequately capture many economic data series, they do it primarily by not smoothing at all. In fact, it isn’t uncommon for the $\alpha$ parameter, which governs the amount of smoothing, to have an optimal value bigger than 1. This means that past data are overweighted. This typically arises because of time averaging of data. If, for instance, a monthly series is constructed as an average of daily values, a sudden change late in the month won’t fully be reflected in the observed data until the next period.

Advantages

ES techniques are computationally simple, so you can quickly apply them to a large number of series. Plus, the small set of choices simplify the process of choosing the “best” ES model and make them ideal for very small data sets.

Disadvantages

The class of models is too narrow for some data series. For instance, a “no trend” model actually is a random walk model, so its forecasts remain at the final smoothed level. For series which return to a mean level, this behavior is very bad, especially at longer horizons. Also, when you have enough data to entertain a richer class of models, other methods are likely to do better.

Using the ESMOOTH Instruction

The ESMOOTH instruction implements exponential smoothing. You will need to choose both the trend model and the seasonal model, and decide between setting the smoothing parameters yourself or estimating them. You choose the models with the TREND and SEASONAL options on ESMOOTH. The choices available for these are:

trend=[none]/linear/exponential/select
seasonal=[none]/additive/multiplicative/select

If you choose SELECT, RATS will test all three choices for that option and select the best fitting model.

You can set the smoothing parameters with the ALPHA, GAMMA, and DELTA options (all default to 0.3), or you can use the ESTIMATE option to have RATS estimate the parameters. With ESTIMATE, RATS chooses the parameters by minimizing the in-sample squared one-step forecast errors. If you have a small amount of data (twenty or fewer data points), you should not try to estimate. Setting the parameters to reasonable values will usually be superior to relying upon imprecise estimates.
Examples

The first example uses \texttt{ESMOOTH} to forecast a pair of U.S. interest rates. This example is supplied on the file \texttt{EXPSMO1.PRG}:

\begin{verbatim}
open data haversample.rat
calendar(m) 1978
data(format=rats) 1978:1 2007:4 fcm30 ftbs3

FCM30 = Yield on 30 Year Treasury Bonds
FTBS3 = Yield on 3 Month Treasury Bills

Forecast twelve periods out-of-sample using simple smoothing model with fixed parameter of .8. Append the forecasts to the series for ease in graphing.

esmooth(alpha=.8,forecast=ftbs3,steps=12) ftbs3
esmooth(alpha=.8,forecast=fcm30,steps=12) fcm30

Graph with more informative key labels, showing the last six months of actual data, followed by the forecasts. Include a vertical line at 2007:4.

graph(grid=(t==2007:4),key=below, $
   
   klabels=||"30 Year Bonds","3 Month Bills"||) 2
# fcm30  2006:11 2008:4
# ftbs3  2006:11 2008:4
\end{verbatim}

This next example “seasonally adjusts” Canadian retail sales, and graphs a comparison with the “official” seasonally adjusted version over a six year period in the middle of the sample. This is supplied on the file \texttt{EXPSMO2.PRG}:

\begin{verbatim}
open data oecdsample.rat
calendar(m) 1960
data(format=rats) 1960:1 2007:3 canrett canretts

Use \texttt{ESMOOTH} with selected trend and seasonal models, putting the “seasonally adjusted” series into \texttt{CANRETSAX}

esmooth(trend=select,seasonal=select,smooth=canretsax,initial=start) canrett

graph 3
# canrett  1994:1 1999:12
# canretts 1994:1 1999:12
# canretsax 1994:1 1999:12

Output from this example is shown in the description of \texttt{ESMOOTH} in the Reference Manual.
\end{verbatim}
9.4 Box-Jenkins Models

Background

The Box-Jenkins methodology provides a broad range of models for fitting the serial correlation patterns of both seasonal and non-seasonal data. The class of models is rich enough that there is often a choice which fits the data adequately with very few parameters.

The fitting process as described in textbooks (Box and Jenkins (1976), Brockwell and Davis (2002), DeLurgio (1998), Enders (2003), and Granger and Newbold (1986) are among the many worthy examples) is an interactive process between the forecaster and the data. We describe the process here briefly, but you will need to consult an appropriate text if you are new to ARIMA modeling.

Advantages

Box-Jenkins models cover a much larger range of series than exponential smoothing models. They require less data than spectral methods, which don’t use the data in quite so “sharp” a fashion. There is a well-developed literature on the methodology, and they have proven to be quite successful in practice.

Disadvantages

The selection of a model depends strongly upon individual judgment. The range of possible models (especially with seasonal data) requires you to make quite a few choices. As a result, two people facing the same data may come up with very different models. Fortunately, if two models provide a nearly identical fit, they will usually produce nearly identical forecasts.

The selection procedure can be difficult to apply, especially with seasonal data. If you have a hard time coming up with an adequate model and you have 100 or more data points, you might try spectral methods instead. On the other end, Box–Jenkins requires more data than exponential smoothing: Granger and Newbold, for instance, recommend at least 40-50 observations.

The Model Selection Process

First, you need to make sure that you are working with a stationary series. An examination of the series can help determine if some preliminary transformation is needed to give a stationary variance. Then, the sample autocorrelations and partial autocorrelations will help you determine whether you need to apply differencing or other transformations (such as taking logs) to produce a stationary series. Only then can you begin fitting an ARMA model to your stationary series.

In order to develop ARIMA models, you need to be familiar with the behavior of the theoretical autocorrelation and partial autocorrelation functions (ACF and PACF) for various “pure” AR, MA, and ARMA processes. See Enders (2003) or similar texts for descriptions and examples of theoretical ACF and PACF behavior.
Potential model candidates are determined by examining the sample autocorrelations and partial correlations of your (stationary) series, and looking for similarities between these functions and known theoretical ACF and PACF. For example, your sample correlations may behave like a pure MA(2) process, or perhaps an ARMA(1,1).

In most cases, you will have several possible models to try. Selecting a final model will usually require fitting different models and using information such as the summary statistics from the estimation (t-statistics, Durbin–Watson statistics, etc.), the correlation behavior of the residuals, and the predictive success of the model to determine the best choice (keeping in mind the principle of parsimony discussed earlier).

Some textbooks recommend an “automatic” model selection procedure which examines all the ARIMA models with a small number of AR and MA parameters (say, all combinations of between 0 and 4 with each), and chooses the model based upon one of the information criteria, like the Akaike or Schwarz criterion. (The procedure BJAUTOFIT, described later in this section, can be used to do this). For many series, this works fine. However, it’s hard to entertain models with lag skips using this type of search. For instance, the search just described would require checking 25 models, but if we allow for any set of lags for each polynomial, such as lags 1 on AR and 1 and 4 on the MA, we’re up to 1024 possible candidates, 32 combinations for each polynomial. For a series with seasonality, it’s very likely that the “best” model will have this type of lag structure. So understanding how to choose and adjust a model is still likely to be important in practice.

Tools
The primary RATS instructions for selecting, fitting, analyzing, and forecasting Box–Jenkins models are BOXJENK, CORRELATE, DIFFERENCE and UFORECAST (or the more general FORECAST). You will probably find it helpful to know how to use each of these instructions. However, we have also provided several procedures that tie these instructions together to allow you to do standard tasks with one or two simple procedure calls. We’ll introduce the instructions by walking through an example first, and then discuss the procedures.

A Worked Example
To demonstrate the tools used to estimate and forecast ARIMA models, we will work with an example from Enders (2003), section 2.10. The series being modeled is quarterly averages of the U.S. producer price index (PPI) from 1960 to 2002 quarter 1. This program is available in the file ARIMA.PRG. The first step is to read in the data and take a look at a graph of the series:

```RATS
open data quarterly.xls
calendar(q) 1960:1
data(format=xls,org=columns) 1960:1 2002:1 ppi
graph(header="Producer Price Index, 1995=100")
# ppi
```
Clearly, it makes little sense to model this in levels: the entire range of the series for the first twelve years is about the same as the one year drop towards the end, when the series has a much higher level. While we could check several transformations (and we’ll do that later when we demonstrate the BJTRANS procedure), this is a series which is naturally modeled in logs, so we’ll choose to do that.

The next step is to check the log transformed data to see if we have to difference to stationarity, and, if so, how many times. The easiest way to do this is with the procedure BJIDENT, described later in the chapter, but we’ll show here how to generate the required graphs “manually.” The statistics (autocorrelations and partial autocorrelations) are computed using the instruction CORRELATE. The example shows the set of options that we like to use for graphing these. Note that this graph includes the 0 lag statistics, which are always 1.0. At this stage of the modeling process (called the identification phase by Box and Jenkins), we find it helpful to leave this in. We’ll take it out in a later stage, when we’re expecting the autocorrelations at positive lags to be near zero.

```
set logppi = log(ppi)
* corr(results=pcorrs,partial=ppcorrs,number=35) logppi
graph(header="Correlations of Log PPI",key=below,\
    style=bar,nodates,min=-1.0,max=1.0,number=0) 2
# pcorrs
# ppcorrs
```

The resulting graph is shown on the next page.
The autocorrelations appear to decay very slowly, strongly suggesting that the series needs to be differenced to achieve stationarity.

```plaintext
diff logppi / dlogppi
*
corr(results=dpcorrs,partial=pdpcorrs,number=35) dlogppi
graph(header="Correlations of 1st Diff",key=below,${style=bar,nodates,min=-1.0,max=1.0,number=0} 2
# dpcorrs
# pdpcorrs
```

After differencing, there's a rapid drop off in the autocorrelations, which would generally indicate a stationary process. However, the rather persistent bulge in the 10-25 zone indicate that a second differencing might be required. We'll work with the first difference for now, and keep in mind that we might need to go to a second difference if the series proves to be hard to fit.
Looking at the correlations of the first differenced series, it’s clear that a low order MA model won’t work, since the autocorrelations are large out to lag seven. The partial autocorrelations help to identify low order AR’s. It looks like an AR(1), or possibly AR(2) might be adequate. Another possibility is an ARMA(1,1). The “signature” for this is an exponential decline on the correlations starting with the first lag. (An AR(1) has an exponential decline starting with lag zero).

With three candidate models chosen, we use the instruction BOXJENK to estimate them. With BOXJENK, you can either estimate the model using the differenced dependent variable, or you can give the original dependent variable and add a DIFFS option to include the differencing as part of the instruction. This has no effect on the estimates, and the only change in the output will be in the $R^2$ and related measures. There are two advantages to having BOXJENK do the differencing:

1. It allows comparable $R^2$ for models with different choices for the number of differences. Note, however, that the $R^2$ is not considered to be an important summary statistic in ARIMA modeling.
2. If you define a forecasting equation using BOXJENK, it will be for the variable of interest, and not for its difference.

RATS offers two criterion functions for fitting ARMA models: conditional least squares and maximum likelihood. The default is conditional least squares—use the MAXL option on BOXJENK to get maximum likelihood. Each method has its advantages. Conditional least squares has a better behaved objective function, and thus is less likely to require “fine tuning” of the initial guess values. However, there are several ways to handle the pre-sample values for any moving average terms, so different software packages will generally produce different results. Maximum likelihood is able to use the same sample range for all candidate models with the same set of differencings, and so makes it easier to compare those models. The RATS procedure BJAUTOFIT uses maximum likelihood for precisely this reason. In addition, maximum likelihood estimates should give the same results across software packages.

The most interesting summary statistics in the BOXJENK output are the standard error of estimate (for conditional least squares) or the likelihood function (for maximum likelihood) as your measure of fit, and also the $Q$ statistic. Ideally, you would like the $Q$ to be statistically insignificant. In addition, it’s standard practice to examine the autocorrelations of the residuals, looking for large individual correlations. The $Q$ can pick up general problems with remaining serial correlation, but isn’t as powerful at detecting a few larger-than-expected lags. For residuals, it generally makes more sense to do the autocorrelations only (not partials), and this time, we leave off the lag zero (entry one).

Our model is going to include an intercept, as we expect a non-zero mean for the log first difference. There are three main ways that software packages deal with the intercept:
1. Remove the mean from the differenced data, then add it back after estimating the model on the mean zero data.
2. Include the intercept as a coefficient in a “reduced form” model.
3. Separate the model into a “mean model” and a “noise model”.

By default, RATS uses the third of these. (You can implement the first by using the DEMEAN option.) For an AR(1) model, the difference between 2 and 3 is that 2 estimates the model in the form $y_t = c + \phi y_{t-1}$, while 3 does $y_t - \alpha = \phi(y_{t-1} - \alpha)$. The two models should produce identical answers (with $c = \alpha(1 - \phi)$) given the same method of estimation. The advantages of the form chosen by RATS is that the estimated coefficient on the constant has a simple interpretation as the mean of the differenced process, and also, that it generalizes better to intervention models (Section 9.6) and the like.

These instructions estimate the AR(1) model and graph its residual autocorrelations.

```plaintext
boxjenk(diffs=1,constant,ar=1) logppi correlate(results=rescorrs,number=35) %resids graph(header="AR(1)",style=bar,nodates,min=-1.0,max=1.0,number=1) # rescorrs 2 *
```

The $Q$ for this is 58.53 with 35 degrees of freedom (the output shows 36–1, which is the number of autocorrelations minus the number of estimated ARMA parameters). This is highly significant. The graph of the residual autocorrelations doesn't show any specific problem, just a many fairly large autocorrelations of similar size. That usually indicates that the model has too few parameters to do the job.
With AR(1) being inadequate, we try the other two models.

```
boxjenk(diffs=1,constant,ar=2) logppi
correlate(results=rescorrs,number=35) %resids
graph(header="AR(2) ",$
        style=bar,nodates,min=-1.0,max=1.0,number=1)
# rescorrs 2 *
*
boxjenk(diffs=1,constant,ar=1,ma=1) logppi
correlate(results=rescorrs,number=35) %resids
graph(header="ARMA(1,1) ",$
        style=bar,nodates,min=-1.0,max=1.0,number=1)
# rescorrs 2 *
```

The ARMA(1,1) has a lower standard error of estimate, and a smaller Q, and thus appears to be preferable. Its residual autocorrelation graph looks like:

![Residual Autocorrelation Graph](image)

Although the autocorrelations are generally fairly well-behaved, the one thing that stands out is that the larger autocorrelations seem to come at lag multiples of four, indicating that there might be some type of seasonal effect remaining in the data. There are several possible ways of trying to handle this, and it usually isn’t clear which is best until you try them and see how they fit.

One is to take the basic model (here an ARMA(1,1)) and add either an AR or MA term at lag four. To put MA coefficients on lags 1 and 4, leaving out 2 and 3, use the option \texttt{MA=|1,4|} on \texttt{BOXJENK}. The other is to use the \texttt{SAR} or \texttt{SMA} options to add a seasonal AR or MA polynomial. Note that the option \texttt{SAR=1} adds a single seasonal AR polynomial (which multiplies by the regular AR) at lag four. \texttt{SAR=4} would give you four terms in the seasonal AR polynomial, thus lags 4, 8, 12 and 16, which you are very unlikely to want. The first model below adds the one seasonal AR, while the second adds a lag 4 to the regular MA polynomial. It appears that the second of the two is somewhat better than the simple ARMA(1,1) model.
Chapter 9: Univariate Forecasting

```plaintext
boxjenk(diffs=1,constant,ar=1,ma=||1,4||) logppi
 correlate(results=rescorrs,number=35) %resids
graph(header=\"(1,1)x(1,0)\",$
    style=bar,nodates,min=-1.0,max=1.0,number=1)
# rescorrs 2 *
boxjenk(diffs=1,constant,ar=1,ma=||1,4||) logppi
 correlate(results=rescorrs,number=35) %resids
graph(header=\"(1,(1,4))\",$
    style=bar,nodates,min=-1.0,max=1.0,number=1)
# rescorrs 2 *
```

We’ll now use our favored model to do some forecasts. To do this, we need to include the `DEFINE` option on the `BOXJENK` that estimates the model. The instruction `UFORECAST` is used to compute the forecasts. In the first case, we’re going to estimate the model through 1997:4, holding back the last four years of data, then forecast the rest. The graph then compares the forecasts with the actuals. Note, by the way, that these are multiple step dynamic forecasts. That is, the forecast for 2002:1 is made using only data through 1997:4.

```plaintext
boxjenk(diffs=1,constant,ar=1,ma=||1,4||,define=favored) logppi * 1997:4
uforecast(equation=favored) forecast 1998:1 2002:1
graph(header=\"Forecasts vs Actuals\") 2
# forecast
# logppi 1997:1 *
```

This is rather typical of time series forecasts. Note that, after the first year or so, the forecast is almost a straight line. As mentioned earlier in the chapter, the long-term forecasts are dominated by the choice of differencing plus the intercept. Here, one difference plus an intercept means the series trends at a fairly constant rate.

The stationary part of the model captures the slower than typical inflation rate during 1997, and extrapolates a low rate of inflation for about one year before...
settling down to the roughly 3.5% per year historical rate of increase. (The intercept is .009 quarterly, so about 3.6% per year). It does not capture the absolute decline during 1998, nor should we expect it to.

The steady two year decline seen in the PPI from 1997 through 1998 is very rare in the data; in fact, there is only one other episode similar to it in forty years (in the mid 1980’s). Given that history, the model will, as it did here, predict that any downward slide will be short-lived. While the model’s forecasts are a bit high for the first year and a half out of sample, note that, if it had come closer during that period, it would likely have underestimated the sharp rise that started in 1999, and would have made a much larger error towards the beginning of 2001.

Now, we use the same model to do a series of one-step-ahead forecasts over that sample. Naturally, the one step forecasts track much better than the multi-step forecasts. The main use of analysis of one step forecasts is to check the adequacy of the fitted model.

```
uforecast(equation=favored,static) onestep 1998:1 2002:1
graph(header="One-Step Forecasts vs Actuals") 2
# onestep
# logppi 1997:1 *
@uforeerrors logppi onestep
```

**Theil U Statistics**

The `UFOREERRORS` procedure is designed to work only with a series of one-step-ahead forecasts. To look at longer forecasts, use the instruction `THEIL`. This does a series of multiple step forecasts and does parallel analysis on each horizon. One of the statistics that it computes is Theil’s U statistic. This is the ratio between the RMS errors of the model’s forecasts to the RMS errors of a “naïve” forecast of no change in the dependent variable from the previous value. See Chapter 8 and the description of THEIL in the Reference Manual for details. This has the advantage over the basic error statistics like the RMSE as it is scale independent, and also offers a quick comparison with another forecasting procedure. Here’s how to use this for our ARMA model. We compute forecasts for (up to) 8 steps ahead for each starting date in the range 1998:1 through 2001:4. The model is re-estimated after each set of forecasts to use the next available data point.

```
theil(setup,steps=8,to=2002:1) 2002:1 is the end of the actual data
# favored
do time=1998:1,2001:4
    theil time
    boxjenk(noprint, diffs=1, constant, ar=1, ma=\|1,4\|, define=favored) logppi * time
end do time
theil(dump)
```
ARMA Models

ARMAX models combine regression models with ARMA terms for the residuals. To implement this in RATS, you use the \texttt{REGRESSORS} option on the \texttt{BOXJENK}, and follow the instruction with a supplementary card listing (in regression format) the variables in the regression model. The ARMA noise terms are handled using the standard \texttt{AR} and \texttt{MA} options. You can also use the \texttt{INPUTS} option, though that is designed primarily for implementing transfer function and intervention models (Section 9.6).

The basic process is as follows: First, develop an appropriate regression model. Next, fit an ARIMA model to the regression residuals using the standard modelling techniques described above. Finally, use \texttt{BOXJENK} to estimate the full ARMAX model.

The following is part of an example from Diebold (2004). The series being forecasted is liquor sales. The series has both a strong trend and seasonal. The basic regression model is a full set of seasonals plus trend and trend squared. From this, an AR(3) model is identified for the noise term. The full model is estimated using the \texttt{BOXJENK} instruction shown.

\begin{verbatim}
seasonal seasons
linreg lsales 1968:1 1993:12 resids
   # time time2 seasons{0 to -11}
boxjenk(regressors,ar=3) lsales 1968:1 1993:12 resids
   # time time2 seasons{0 to -11}
\end{verbatim}

Note again that the \texttt{BOXJENK} instruction is designed with a mean model (which here is given by the set of regressors) and a noise model, and not a combined “reduced” form. With a simple ARMA model, the only difference is in the interpretation of the coefficients. And that will still be the case with these types of deterministic regressors, as any linear combination of their lags is just a linear combination of the regressors themselves. However, if you have regressors which aren’t deterministic, the models aren’t equivalent. Some programs will set up a model (AR(1) with one regressor) as $s_t = \phi s_{t-1} + \alpha + bx_t + \varepsilon_t$, while RATS does $(1-\phi L)(s_t - (\alpha + bx_t)) = \varepsilon_t$. 
9.4.1 The ARIMA Procedures

RATS includes several procedures which are designed to simplify the process of selecting, estimating and forecasting ARIMA models. We introduce these procedures below, and apply them to our PPI example. See Section 1.10 for an introduction to the use of procedures.

All of these procedures need to be compiled (brought into the program) before they can be used. You can do this directly using SOURCE instructions, or by including them in your procedure library. Or, you can let RATS search for .SRC files with names matching the procedure names.

BJTRANS Procedure

The procedure BJTRANS (from BJTRANS.SRC) can be helpful in determining the appropriate preliminary transformation if you are unsure which should be used. It graphs the series, its square root and its log on a single graph so you can compare their behavior. As mentioned on page 315, you’re looking for the transformation which “straightens” out the trend (if there is one), and produces approximately uniform variability. The graph also shows a (very crude) log likelihood of a model under the three transformations. You shouldn’t necessarily choose the transformation based upon which of these is highest, as a close decision might be reversed with a careful model for the transformed processes. However, if there is a very clear difference among the likelihoods, they are likely to be pointing you to the best choice.

The syntax is:

@BJTrans series start end

The following is the graph produced by applying BJTRANS to the PPI series in our example. Now, this is a series which we expect to be best modeled in logs, and the log likelihood values would support that. The levels (“None”) are clearly inadequate. Note, from the graph, that there is effectively no visible short-term movements in the first half of the sample, while it is considerably less smooth in the second half, but the log transform gives a more uniform variability across the full sample.
Chapter 9: Univariate Forecasting

BJIDENT Procedure
The procedure **BJIDENT** (from the file **BJIDENT.SRC**) assists in identifying a Box-Jenkins model by computing and graphing the correlations of the series in various differencing combinations. Here is the syntax for the **BJIDENT** procedure, with the most common set of options:

\[@bjident\(\) \text{options} \) \text{series} \text{ start} \text{ end}\]

**Parameters**
- **series**: series for which you want to identify a model
- **start, end**: range over which you want to compute correlations. Defaults to the maximum usable range given the selected **DIFFS/SDIFFS**.

**Options**
- **diffs=**maximum regular differencings\([0]\)
- **sdiffs=**maximum seasonal differencings\([0]\)
  \(\text{BJIDENT}\) examines every combination of regular and seasonal differencing between 0 and the maximums indicated. For instance, with each set at 1, it will do 0x0, 0x1, 1x0 and 1x1.

- **number=**number of autocorrelations to compute\([25\text{ if possible}]\)
- **method=yule/\text{burg}**
  Sets the number of autocorrelations to compute and graph. The \text{METHOD} option selects the method to be used in computing the correlations (see the description of the **CORRELATE** instruction). \text{METHOD=YULE} is used by many other programs.

- **trans=[none]/log/root**
  Selects the preliminary transformation.

- **[graph]/\text{nograph}**
  If **GRAPH**, **BJIDENT** creates high-resolution graphs of the correlations.

- **report/[noreport]**
- **qstats/[noqstats]**
  If you use the **REPORT** option, **BJIDENT** will create a table of the computed statistics for the various differencing combinations. If you include **QSTATS** as well, it will compute Ljung-Box \(Q\) statistics and their significance levels. While some software does the \(Q\) statistics as part of their “ident” functions, we recommend against it. The \(Q\)'s are tests for white noise; they have no real implications for stationarity decisions, and at this point, we don’t expect to see the correlations pass a white noise test even with the correct number of differencings.
Examples

Recall the fairly complicated **DIFF**, **CORR** and **GRAPH** instructions that we used in our earlier example to produce graphs of the correlations for both **RATE** and the first difference of **RATE**. Using **BJIDENT**, the following is all you need to produce even fancier versions of these graphs, plus the graph for the second difference as well:

```plaintext
open data quarterly.xls
calendar(q) 1960
data(format=xls,org=columns) 1960:1 2002:1 ppi
@bjident(trans=log,diffs=2) ppi
```

The next example produces graphs for all combinations of 0 or 1 regular difference and 0 or 1 seasonal differences for the log of **MWHRS** (which is a monthly series of demand for electrical power). It shows 36 correlations.

```plaintext
open data mwhrs.dat
calendar(m) 1980
data(format=free,org=columns) 1980:1 1989:12 mwhrs
set lmwhrs = log(mwhrs)
@bjident(number=36,diffs=1,sdiffs=1) lmwhrs
```

This example looks at 24 autocorrelations of two series, producing for each both the graph and a table of the correlations, partial autocorrelations and sequential Q-statistics.

```plaintext
open data house.dat
cal(m) 1968:1
data(format=prn,org=columns) 1968:1 1996:6 starts completions
@bjident(number=24,report,qstats) starts 1968:1 1991:12
@bjident(number=24,report,qstats) completions 1968:1 1991:12
```
REGCORRS Procedure

REGCORRS computes and optionally graphs the residual autocorrelations, computing the $Q$-statistic and the AIC and SBC. It can be used after any type of single-equation estimation, not just after BOXJENK.

@regcorrs( options ) resids

Parameters

resids Series of residuals to analyze. You can omit this if you want to use the residuals from the most recent estimation.

Options

[graph]/nograph
print/[noprint]
[qstats]/noqstats
report/[noreport]
[criteria]/nocriteria
title="title for report window"
header="graph header string"
footer="graph footer string"

If GRAPH, REGCORRS creates high-resolution graphs of the correlations. If PRINT, the output from CORRELATE is included in the output window, and, if, in addition, you use QSTATS, that output will include the $Q$–statistics. If REPORT, a separate table is displayed with the correlations, partial autocorrelations and $Q$–statistics for different lag lengths.

number=number of autocorrelations to compute [25 if possible]
dfc=degrees of freedom correction for $Q$ statistics
method=[yule]/burg

NUMBER sets the number of autocorrelations to compute and graph. DFC should be set to the number of ARMA parameters estimated; BOXJENK sets the variable %NARMA equal to this. The METHOD option selects the method to be used in computing the correlations (see the description of the CORRELATE instruction).

METHOD=YULE is used by most other programs.

Variables Defined

In addition to the Q-statistic variables (%QSTAT, %QSIGNIF and %NDFQ), REGCORRS also computes %AIC and %SBC, which are the Akaike Information Criterion and the Schwarz Bayesian Criterion. The values of these will also be displayed on the graph.
BJAUTOFIT Procedure

The **BJAUTOFIT** procedure estimates (by maximum likelihood) all combinations of ARMA terms in a given range, displaying a table with the values of a chosen information criterion, with the minimizing model having its value highlighted with a *. Note that this can take quite a while to complete if the $p_{\text{MAX}}$ and $q_{\text{MAX}}$ are large. Also, note that some of the more complicated models may not be well-estimated with just a simple use of **BOXJENK(MAXL)**. However, these estimation problems generally arise because of poorly identified parameters, which means that a simpler model would fit almost as well and would thus be preferred anyway.

The syntax is as follows:

@@BJAutoFit(options) series start end@  

**Parameters**

- **series**
  Series to be forecast.

- **start end**
  Range of entries to use in estimation. By default, the largest range possible given the data and the number of differences.

**Options**

- **pmax=** maximum number of AR lags to consider [0]
- **qmax=** maximum number of MA lags to consider [0]
- **crit=[aic]/bic/sbc** (criterion to use—BIC and SBC are synonyms)
- **const/[noconst]** (estimate mean as part of model)
- **demean/[nodemean]** (extract mean before estimating ARMA models)
- **diffs=** number of preliminary differences [0]
- **sdiffs=** number of preliminary seasonal differences [0]

**Example**

We use **BJAUTOFIT** on the log PPI data, looking at models up to an ARMA(4,4).

@bjautofit(diffs=1, constant, pmax=4, qmax=4, crit=sbc) logppi

The results confirm our choice of a 1,1 model:

BIC analysis of models for series LOGPPI

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>-956.0594</td>
<td>-998.1801</td>
<td>-1009.6776</td>
<td>-1009.0213</td>
<td>-1017.9265</td>
</tr>
<tr>
<td>1</td>
<td>-1026.8179</td>
<td>-1033.0090*</td>
<td>-1028.1232</td>
<td>-1023.6617</td>
<td>-1026.1306</td>
</tr>
<tr>
<td>2</td>
<td>-1028.7273</td>
<td>-1028.0621</td>
<td>-1023.0697</td>
<td>-872.6370</td>
<td>-936.0209</td>
</tr>
<tr>
<td>3</td>
<td>-1027.3047</td>
<td>-1023.6922</td>
<td>-1025.8206</td>
<td>-1015.1978</td>
<td>-919.0232</td>
</tr>
<tr>
<td>4</td>
<td>-1025.8509</td>
<td>-1024.9963</td>
<td>-1020.1469</td>
<td>-1019.5336</td>
<td>-1014.9910</td>
</tr>
</tbody>
</table>
9.5 Spectral Forecasting

Background
Spectral forecasting uses frequency domain methods to fit a “generic” Box-Jenkins model to the data. The preparations (transformation and differencing) are the same as for a BJ model.

It is a non-parametric method, so parsimony is not a well-defined concept. Technically, the estimates of cycles are smoothed to eliminate possibly spurious effects due to sampling error.

The procedure used is technically very complex. For those of you who are familiar with $z$-transforms, we describe the algorithm in Section 15.8.

Advantages
The advantages of spectral forecasting tend also to be disadvantages. It is an automatic procedure once you have chosen the preliminary transformation and trend model. Given adequate data, it can produce good forecasts quickly and painlessly. It can do better than Box-Jenkins, particularly when no model with just a few parameters seems to work well.

Disadvantages
As an automatic procedure, there are no diagnostics along the way to steer us away from problems. It is less tolerant of mistakes in preliminary transformation and requires more data (100+ observations) than parametric techniques.

Procedures
You can do spectral forecasting using the procedure `SPECFORE`. This is on the file `SPECFORE.SRC`. The syntax for the procedure is:

```
@specfore(options) series start end forecasts
```

Parameters
- `series` Series to forecast
- `start` Range of entries to forecast
- `end` Range of entries to forecast
- `forecasts` Series for computed forecasts
Options

The options select the transformation and trend model.

- **diffs**=
  - Number of regular differences [0]
- **sdiffs**=
  - Number of seasonal differences [0]
- **constant**/noconstant
  - Select NOCONSTANT to exclude the intercept from the differenced model.

- **trans**=[none]/log/root
  - Select the appropriate preliminary transformation.

Example

This computes a spectral method forecast of the five year US treasury bond yield. It’s example file SPECFORE.PRG.

```
open data haversample.rat
calendar(m) 1960
data(format=rats) 1960:1 2007:4 fcm5

    Compute a forecast for the last 16 months of actual data
@specfore(diffs=1) fcm5 2006:1 2007:4 insample

    Graph the forecasts and actuals:
graph(key=below,header="Interest Rate Forecast") 2
# fcm5 2006:1 2007:4
# insample 2006:1 2007:4

    Compute an 20-month out-of-sample forecast and graph the results:
@specfore(diffs=1) fcm5 2007:5 2008:12 outofsample
graph(key=below,header="Interest Rate Forecast") 2
# fcm5 2006:1 2007:4
# outofsample
```
9.6 Intervention Modeling

Background

Intervention modeling allows you to deal with simple types of structural breaks while using an ARIMA model. It does this by adding a deterministic component, such as a dummy variable, to the basic ARIMA model. Its scope is somewhat limited because it requires that the date and general form of the change be known. It also requires the rather strong assumption that the time series properties of the series stay the same after the break. For instance, if a country chooses to devalue its currency and then attempts to keep its value in line with (say) the US dollar, we would know the date, and the form of the break, but the time series structure would likely be quite different before and after, making intervention modeling impractical.

Tools

Intervention models are estimated using BOXJENK. However, before running BOXJENK, you must create the deterministic components which will create the effect that you want. These are added to the model using the INPUTS option. Usually there is just one input, so we’ll demonstrate that:

boxjenk(inputs=1, other options) series start end
  # input p d q

For intervention modeling, it is usually simplest to keep the time series in its undifferenced form. Then, by using the APPLYDifferences option you will be able to figure out the form of the input based upon what change it makes to the level of the series. If you work with the differenced series, you will have to work out the input as a shift in the differenced series, which is usually much harder.

Here are some examples of intervention inputs:

For a permanent change in level beginning at $T_0$

set change = (t>=t0)

For a short term change in level from $T_0$ to $T_1$

set change = (t>=t0 and t<=t1)

For a permanent change in growth rate beginning at $T_0$

set change = %max(0.0, t-(t0+1))
Notes

Even if you know the date of the intervention and are satisfied that you understand its form, and you are willing to assume that the time series properties are the same except for the structural break, you still have one problem: figuring out the proper form for the base ARIMA model. You can’t apply the standard identification techniques to the original series, as the autocorrelations will be thrown off by the structural break. There are a couple of ways to approach this:

1. You can apply the identification procedures to whichever stretch of “stable” data is longest; that is, restrict your attention only to the part of the sample before the break, or the part after it.

2. You can apply the identification procedure to the time series after accounting for the type of change that you are planning to permit. This works best for a level change, as you can just take the residuals from the regression on CONSTANT and your dummy variable. If you have a growth rate change, you’re almost certainly looking at data which, at minimum, will be differenced once. The equivalent procedure in that case would be to difference the data, and then take the residuals from the regression on CONSTANT and a level change dummy starting at \( T_0 \).

The “denominator lags” parameter on the input line can be used if there is a “phase-in” before the permanent change takes full effect. When applied to a temporary effect, the denominator lags create a “phase-out.” A single denominator lag allows for a geometric convergence to the new level. If you use this, you have to be careful interpreting the coefficients. In all the examples above, if you use a simple input without lags, the coefficients on the CHANGE variable will be easy to interpret: in the first two, they will show the size of the level shift, and in the last the difference in the growth rate. However, in the two permanent effects, if the coefficient on the input is 3.0 with a denominator lag coefficient of .6, the long-run effect will be 3.0/(1–.6) or 7.5.

Example

The U.S. stock market took a very large tumble in October, 1987. We will apply intervention techniques to the S&P 500. Because the series we are using consists of monthly averages (not the ideal choice for a serious analysis of this event—daily closing prices would be better), the October, 1987 value does not fully show the decline, since over half the values averaged into the October value were from before the large drop on October 19. This means that we’re going to need a lag in the numerator as well, to give the effect two months to take full effect. We’ll also look at two possible ways to model the change: a permanent drop, or a temporary drop with a phase out (see DeLurgio (1998) for another treatment of this same basic example).

We won’t go through the process of choosing the base ARIMA model for this series. It proves to be a relatively easy choice of an ARIMA(0,1,1) with a CONSTANT on the logs.
(The ARIMA(0,1,1) is the expected model for the time-averages of a series which is likely to be fairly close to a random walk in continuous time). This is example INTERVEN.PRG.

\begin{verbatim}
cal(m) 1947:1
open data haversample.rat
data(format=rats) 1947:1 1993:12 sp500
set logsp_500 = log(sp500)

The permanent effect is done using a dummy starting in 1987:10. The temporary effect is a spike at 1987:10.

set perm = t>=1987:10
set temp = t==1987:10

The permanent effect is done with lags 0 and 1 on perm

boxjenk(diff=1,applydiff,constant,inputs=1,ma=1) logsp_500 $ 1980:1 1993:12 presids
# perm 1

The temporary effect is done with numerator lags 0 and 1 and a single denominator lag.

boxjenk(diff=1,applydiff,constant,inputs=1,ma=1) logsp_500 $ 1980:1 1993:12 tresids
# temp 1 1

The following is the output from the two estimated models. Don’t be deceived by the extremely high $t$-stat on the \texttt{D_TEMP{1}} coefficient in the second model. It doesn’t mean the second model is better. In fact, the first model is a special case of the second with the coefficient on \texttt{D_TEMP{1}} fixed at 1.0, so the two are almost identical. The results show a roughly 25% drop in the index through the two month period in October and November.
\end{verbatim}
Box-Jenkins - Estimation by Gauss-Newton
Convergence in 6 Iterations. Final criterion was 0.0000075 <= 0.0000100
Dependent Variable LOGSP_500
Monthly Data From 1980:01 To 1993:12
Usable Observations 168 Degrees of Freedom 164
Centered R**2 0.995444 R Bar **2 0.995360
Uncentered R**2 0.999968 T x R**2 167.995
Mean of Dependent Variable 5.4391291443
Std Error of Dependent Variable 0.4591986521
Standard Error of Estimate 0.0312785545
Sum of Squared Residuals 0.1604490677
Log Likelihood 345.73271
Durbin-Watson Statistic 1.995114
Q(36-1) 28.681593
Significance Level of Q 0.76579520

<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>0.010310788</td>
<td>0.002907481</td>
<td>3.54630</td>
<td>0.00050947</td>
</tr>
<tr>
<td>2. MA(1)</td>
<td>0.200127855</td>
<td>0.077450051</td>
<td>2.58396</td>
<td>0.01063889</td>
</tr>
<tr>
<td>3. N_PERM(0)</td>
<td>-0.128493661</td>
<td>0.031743366</td>
<td>-4.04789</td>
<td>0.00007949</td>
</tr>
<tr>
<td>4. N_PERM(1)</td>
<td>-0.137958367</td>
<td>0.031372235</td>
<td>-4.39747</td>
<td>0.00001963</td>
</tr>
</tbody>
</table>

Box-Jenkins - Estimation by Gauss-Newton
Convergence in 12 Iterations. Final criterion was 0.0000040 <= 0.0000100
Dependent Variable LOGSP_500
Monthly Data From 1980:01 To 1993:12
Usable Observations 168 Degrees of Freedom 163
Centered R**2 0.994740 R Bar **2 0.994611
Uncentered R**2 0.999963 T x R**2 167.994
Mean of Dependent Variable 5.4391291443
Std Error of Dependent Variable 0.4591986521
Standard Error of Estimate 0.0337106133
Sum of Squared Residuals 0.1852340878
Log Likelihood 333.66663
Durbin-Watson Statistic 1.972291
Q(36-1) 32.881322
Significance Level of Q 0.57076924

<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>0.008715909</td>
<td>0.003419433</td>
<td>2.54893</td>
<td>0.01172839</td>
</tr>
<tr>
<td>2. MA(1)</td>
<td>0.317061758</td>
<td>0.074340267</td>
<td>4.26501</td>
<td>0.00003377</td>
</tr>
<tr>
<td>3. N_TEMP(0)</td>
<td>0.001909992</td>
<td>0.020301010</td>
<td>0.09408</td>
<td>0.92515830</td>
</tr>
<tr>
<td>4. N_TEMP(1)</td>
<td>-0.002947065</td>
<td>0.021264954</td>
<td>-0.13859</td>
<td>0.88994688</td>
</tr>
<tr>
<td>5. D_TEMP(1)</td>
<td>-0.952734786</td>
<td>0.072001530</td>
<td>-13.23215</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>
Chapter 10

Vector Autoregressions

Vector autoregressions (VAR’s) are dynamic models of a group of time series. In “Macroeconomics and Reality” and later papers, Sims has proposed using VAR’s as an alternative to large simultaneous equations models for studying the relationship among the important aggregates. The two main uses of this methodology are:

• to test formally theories which imply particular behavior for the vector autoregression.
• to learn more about the historical dynamics of the economy.

With the use of Bayesian techniques, VAR’s have also been employed very successfully for small multivariate forecasting models.

VAR’s have become a key tool in modern macroeconometrics. While there is some coverage of the topic in econometrics books such as Greene (2003) and introductory time series books like Diebold (2004), we would recommend more specialized books such as Enders (2003) and Hamilton (1994). Of these two, Enders is the more applied, Hamilton, the more theoretical.

Estimation
Testing Lag Length
Exogeneity Tests
Orthogonalization and Covariance Models
Impulse Responses
Decomposition of Variance
Historical Decomposition
Cointegration and Error-Correction
Forecasting with a Prior
Sequential Estimation and the Kalman Filter
Conditional Forecasting
Chapter 10: Vector Autoregressions

10.1 Setting Up a VAR

Description

Formally, a vector autoregression may be written

\[
y_t = \sum_{s=1}^{L} \Phi_s y_{t-s} + u_t \quad \quad E(u_t'u_t') = \Sigma
\]

where \(y\) is an \(N\)-vector of variables and \(\Phi_s\) is an \(N \times N\) matrix. There are a total of \(N^2L\) free coefficients in this model.

Standard VAR’s

You can set up a standard VAR using the VAR (Setup/Estimate) wizard on the Statistics menu, or by using the following instructions directly:

\[
\text{system(model=\text{modelname})} \quad \quad \text{variables} \quad \text{list of endogenous variables} \quad \quad \text{lags} \quad \text{list of lags} \quad \quad \text{deterministic} \quad \text{list of deterministic/additional variables in regression format} \quad \quad \text{end(system)}
\]

The lags listed on LAGS are usually consecutive, for instance, 1 TO 12, but you can skip lags (for instance, 1 2 3 6 12). The list of deterministic variables is usually just CONSTANT and possibly seasonal or other dummies, but they can be any variables other than the lagged endogenous variables.

For example:

\[
\text{system(model=canusa)} \quad \quad \text{variables usam1 usatbill canm1 cantbill canusxr} \quad \quad \text{lags} \quad \text{1 to 13} \quad \quad \text{det} \quad \text{constant} \quad \quad \text{end(system)}
\]

defines a five-equation, 13–lag VAR model. Note that these instructions simply define the VAR system. The model then needs to be estimated, which is usually done using the ESTIMATE instruction (Section 10.2).

Near–VAR’s

The equations set up by the process above will always have precisely the same set of variables on their right hand sides. You cannot use different sets of dummy variables in DETERM or different lag structures, or leave some endogenous variables out of some of the equations. A “near–VAR” is a model which is similar to a VAR, but has some slight differences among the equations. To construct a near–VAR, you need to set up the equations separately using the instruction EQUATION. These equations are then bound into a system using SYSTEM and END(SYSTEM) without the other VAR instructions (VARIABLES, LAGS, etc.).
Here’s an example of a near–VAR setup—the equation for USA includes only the lags of USA itself, while the other two regions are full VAR equations.

```r
equation usaeq usagdp  
   # constant usagdp{1 to 4}  
equation mexeq mexgdp  
   # constant usagdp{1 to 4} mexgdp{1 to 4} centgdp{1 to 4}  
equation centeq centgdp  
   # constant usagdp{1 to 4} mexgdp{1 to 4} centgdp{1 to 4}  
system usaeq mexeq centeq  
end(system)
```

Preliminary Transformations

You should choose the transformation for each series (log, level or other) that you would pick if you were looking at the series individually. Thus, you transform exponentially growing series, such as the price level, money stock, GNP, etc. to logs. You will usually leave in levels non-trending series, such as interest or unemployment rates. This is especially important for interest rates in a VAR including prices or exchange rates (which should be in logs) because real interest rate and parity conditions can be expressed as a linear relationship among the variables.

While preserving interesting linear relationships is desirable, you should not shy away from obvious transformation choices to achieve them. For instance, in Doan, Litterman and Sims (1984), two of the variables in the system were government receipts and expenditures. These are obvious candidates to be run in logs, which makes the deficit a non-linear function of the transformed variables. If we had used levels instead of logs, it would have made studying the budget deficit easier as we would not have needed to linearize. However, the predictions would have been unrealistic since the growth relationship between receipts and expenditures would have been distorted.

Should I Difference?

Our advice is no, in general. In Box–Jenkins modeling for single series, appropriate differencing is important for several reasons:

- It is impossible to identify the stationary structure of the process using the sample autocorrelations of an integrated series.
- Most algorithms used for fitting ARIMA models will fail when confronted with integrated data.

Neither of these applies to VAR’s. In fact, the result in Fuller (1976, Theorem 8.5.1) shows that differencing produces no gain in asymptotic efficiency in an autoregression, even if it is appropriate. In a VAR, differencing throws information away (for instance, a simple VAR on differences cannot capture a co-integrating relationship), while it produces almost no gain.
Chapter 10: Vector Autoregressions

Trend or No Trend?
In most economic time series, the best representation of a trend is a random walk with drift (Nelson and Plosser (1982)). Because of this, we would recommend against including a deterministic trend term in your VAR. In the regression

$$y_t = \alpha + \gamma t + \beta_1 y_{t-1} + \ldots + \beta_p y_{t-p} + u_t$$

if we expect to see a unit root in the autoregressive part, $\gamma$ becomes a coefficient on a quadratic trend, while $\alpha$ picks up the linear trend. As you add variables or lags to the model, there is a tendency for OLS estimates of the VAR coefficients to “explain” too much of the long-term movement of the data with a combination of the deterministic components and initial conditions (see Sims (2000)). While this may seem to “improve” the fit in-sample, the resulting model tends to show implausible out-of-sample behavior.

How Many Lags?
If you want to, you can rely on information criteria, such as the Akaike Information Criterion (AIC) or the Schwarz or Bayesian Information Criterion (variously BIC, SBC, SIC), to select a lag length. Example 10.1 shows how these can be computed. However, these tend to be too conservative for most purposes. Where we are including an identical number of lags on all variables in all equations, the number of parameters goes up very quickly—we add $N^2$ parameters with each new lag. Beyond the first lag or two, most of these new additions are likely to be unimportant, causing the information criteria to reject the longer lags in favor of shorter ones. However, some of the equations are likely to need longer lags to handle the dynamics of the data adequately. It is also possible to use a systematic procedure to choose a different number of lags for each variable in each equation. This, however, simply shifts the type of “overfitting.” The use of priors (see Section 10.10) is an alternative to relying on short lags or data-driven selection methods.

If the data are adequate, it is recommended that you include at least a year’s worth of lags. In fact, a year’s worth plus one extra can deal with seasonal effects, which can often be present even with seasonally adjusted data (which sometimes “overadjust”).
10.2 Estimation

Once you have defined a VAR model, you need to estimate it. If you use the VAR (Setup/Estimate) wizard, this is done automatically. Otherwise, you will normally use the ESTIMATE instruction, although you can also use LINREG or SUR for this. All three methods are described below.

Using ESTIMATE

Once you have set up a system, use ESTIMATE to do the regression. This estimates all equations in the system independently using least-squares. Usually, the syntax is quite simple. The instruction

\texttt{ESTIMATE start end}

- estimates each of the equations over the range \texttt{start} to \texttt{end}.
- prints the output for each, following each with a table of F-tests for exclusion of lags of each of the variables.
- computes and saves the covariance matrix of residuals (required for variance decompositions) in the array \%SIGMA.

You can omit \texttt{start} and \texttt{end} if you want to use the maximum range or if you have set a \texttt{SMPL}. ESTIMATE is a quick, efficient instruction which makes the fullest use of the structure of the VAR. However, because it does an equation by equation estimate of the full system, you cannot do any hypothesis tests using the regression-based instructions like EXCLUDE and RESTRICT. This is not a serious problem, because there are very few interesting hypotheses which can be expressed as restrictions on an individual equation from the system.

Using LINREG

You can use a set of LINREG instructions if you really want to be able to focus on an individual equation. If you want to compute or print the variance-covariance matrix of the system, you will need to save the residuals and use the instruction VCV to compute and print the VCV matrix.

Using SUR

If you have a near-VAR system, there may be some gain to using SUR (Seemingly Unrelated Regressions) to estimate the system instead of ESTIMATE. However, this is really only feasible for small systems because the number of parameters quickly gets excessive. RATS can accurately handle systems with over 200 parameters, but the improvement in efficiency will usually not be worth the trouble. The standard results on SUR vs OLS (Greene (2003), p 343) tell us there can be a gain in efficiency only if

- the residuals are correlated across equations, and
- the regressors are not the same in each equation.

When the residual correlation is not large and there is much duplication in regressors, the improvement will be small.
10.3 Data Analysis

Goals and Tools

As mentioned in the introduction to the chapter, we have two basic goals in using VAR’s to analyze a set of series:

- To test formally theories which imply particular behavior for the vector autoregression.
- To learn more about the historical dynamics of the economy.

We are somewhat limited in what we can test formally using simple VAR’s: most of the interesting hypotheses boil down to multivariate generalizations of the Granger-Sims causality tests (Section 6.6). Because the hypotheses will usually be joint restrictions across equations, a simple EXCLUDE instruction will not suffice. We use the special instruction RATIO which tests such cross-equation restrictions. Section 10.4 discusses testing procedures. The analysis of cointegrating relationships is covered in Section 10.8.

Mostly, VAR’s are used to study the dynamics of the data. The coefficients in the estimated VAR are of little use themselves. Instead, we use a set of instructions which provide equivalent information in a more palatable form. The three key instructions are:

IMPULSE computes impulse responses: if the system is shocked by $x$, $y$ is the expected response.

ERRORS computes the decomposition of variance. This decomposes the variance of the forecast errors in a series into the parts attributable to each of a set of innovation (shock) processes.

HISTORY computes a historical decomposition. The historical data is decomposed into a trend (forecast) and the accumulated effects of the residuals.

All of these can be handled using the VAR (Forecast/Analyze) wizard on the Statistics menu.

In addition, the instruction CVMODEL can be used to identify, estimate and test models for the contemporaneous relationship among the innovations in the VAR.
10.4 Hypothesis Tests

Background

There are relatively few interesting hypotheses which you can test using just the estimates of a single equation from a VAR. Even the block $F$–tests produced by `ESTIMATE`, which indicate whether variable $z$ helps to forecast variable $x$ one-step ahead, are not, individually, especially important. $z$ can, after all, still affect $x$ through the other equations in the system.

Thus, most hypotheses will include more than one equation. The testing procedure to use is the Likelihood Ratio. The test statistic we recommend is

\[
(1) \quad (T - c) \left( \log |\Sigma_r| - \log |\Sigma_u| \right)
\]

where $\Sigma_r$ and $\Sigma_u$ are the restricted and unrestricted covariance matrices and $T$ is the number of observations. This is asymptotically distributed as a $\chi^2$ with degrees of freedom equal to the number of restrictions. $c$ is a correction to improve small sample properties: Sims (1980, p.17) suggests using a correction equal to the number of variables in each unrestricted equation in the system. To help make this correction, `ESTIMATE` sets the variable `%NREG` equal to the number of regressors per equation, and `%NREGSYSTEM` to the number in the whole VAR.

Computing the Statistic

You can compute (1) yourself using the variable `%LOGDET` defined by `ESTIMATE`, or you can use the special instruction `RATIO`. To use `RATIO`, you need to compute the two sets of residuals series. To compute it directly, you need to save the `%LOGDET` values into variables with different names after each of the `ESTIMATE` instructions. Example 10.1 uses both techniques.

Note, by the way, that some hypotheses might have a non-standard distribution in the presence of unit roots (Sims, Stock and Watson (1990)). Their result won’t affect the lag length test (Example 10.1), but will affect the exogeneity test (Example 10.2).

The Examples

Both the examples on the next two pages use the same data set. This is a five variable system comprising real GDP ($GDPH$), unemployment rate ($LR$), gross private fixed investment ($IH$), implicit price deflator ($DGNP$) and $M2$ ($FM2$). All source variables are seasonally adjusted. We transform all but the unemployment rate to logs. Both examples use the same set of initial instructions.

It’s important in doing the lag length tests to make sure that the regressions are run over the same time period. The default behavior for `ESTIMATE` would produce a shorter regression range for the model with longer lags.
Example 10.1 Testing Lag Length

To test formally one overall lag length versus another, we need to run two VAR’s over the same time period. Here this is done by fixing the start period of estimation to 1961:1 on the ESTIMATE instructions. The multiplier correction is the %NREG from the longer model (5x8+1 = 41) and the degrees of freedom is 5x5x4=100 (4 lags of each of 5 variables in 5 equations), which is here calculated using the difference between the values of %NREGSYSTEM.

In addition to the test of 8 vs 4, the example generates a table of the Akaike Information Criterion and Schwarz Bayesian Criterion for the different lag lengths. It’s a good idea to use a version of the AIC which corrects for degrees of freedom. When applied to a VAR, the standard AIC will often “select” a model which nearly exhausts the degrees of freedom. The table also includes a set of sequential likelihood ratio tests and their significance levels. These test for the significance of the last set of lags added. The procedure VARLagSelect, which is also demonstrated, can be used to select the lag length selection automatically.

This example below is available in the file VARLAG.PRG.

```plaintext
open data haversample.rat
calendar(q) 1959
data(format=rats) 1959:1 2006:4 dgnp fm2 gdph ih lr *
set loggdp = log(gdph)
set logm2  = log(fm2)
set logp   = log(dgnp)
set logi   = log(ih)
set unemp  = lr *
system(model=usamodel)
variables loggdp unemp logi logp logm2
lags 1 to 8
det constant
end(system)
estimate(noprint,resids=resids8) 1961:1 *
compute logdetu=%logdet
compute mcorr=%nreg,ntotal=%nregsystem *
system(model=usamodel)
variables loggdp unemp logi logp logm2
lags 1 to 4
det constant
end(system)
estimate(noprint,resids=resids4) 1961:1 *
compute logdetr=%logdet
```
Using RATIO

calculate nrestr=nrestr-\%nregsystem
ratio(degrees=nrestr,mcorr=mcorr,title="Test of 8 vs 4 Lags")
# resid8
# resid4

Using calculated statistic

cdf(title="Test of 8 vs 4 Lags") chisqr $
($)\n($\n(%nobs-mcorr)*(logdetr-logdetu) nrestr

Using VARLagSelect procedure

@varlagselect(lags=8,crit=aic)
# loggdp unemp logi logp logm2

This shows the workings of VARLagSelect, showing columns for both AIC and SBC, and also shows sequential LR tests. Note that this uses a version of AIC with small-sample correction.

disp
report(action=define,$
  hlabel=||"Lags","AIC","SBC","LR Test","P-Value"||)
dec real lastll
  do lags=1,8
    system(model=usamodl)
    variables loggdp unemp logi logp logm2
    lags 1 to lags
    det constant
  end(system)
  estimate(noprint) 1961:1 *
    compute ll =%logl
    compute sbc=-2.0*ll/%nobs+%nregsystem*log(%nobs)/%nobs
    compute aic=-2.0*ll/%nobs +$
           %nregsystem*2.0*%nvar /(%nvar*%nobs-%nregsystem-1)
  report(row=new,atcol=1,align=decimal) lags aic sbc
  if lags>1
    report(row=current,atcol=4) 2*(ll-lastll) $
       %chisqr(2*(ll-lastll),%nvar*%nvar)
    compute lastll=ll
  end do lags
report(action=format,atcol=2,tocol=2,special=onestar,tag=min,align=decimal)
report(action=format,atcol=3,tocol=3,special=onestar,tag=min,align=decimal)
report(action=format,atcol=2,tocol=3,width=8)
report(action=format,atcol=4,tocol=5,picture="*.*####")
report(action=show)
Example 10.2 Block Exogeneity

A block exogeneity test has as its null hypothesis that the lags of one set of variables do not enter the equations for the remaining variables. This is the multivariate generalization of Granger–Sims causality tests (Section 6.6). We test here that the real variables (LOGGDP, UNEMP and LOGI) as a block are exogenous. We estimate two systems, with and without the lags of LOGP and LOGM2.

This is example VARCAUSE.PRG.

```rats
open data haversamp.rat
calendar(q) 1959
data(format=rats) 1959:1 2006:4 dgnp fm2 gdph ih lr
* set loggdp = log(gdph)
set logm2  = log(fm2)
set logp   = log(dgnp)
set logi   = log(ih)
set unemp  = lr
*
system(model=unrestricted)
variables loggdp unemp logi
lags 1 to 4
det constant logp{1 to 4} logm2{1 to 4}
end(system)
estimate(resids=unresids)
compute ntotal=%nregsystem,mcorr=%nreg,loglunr=%logl
compute logdetunr=%logdet
*
system(model=restricted)
variables loggdp unemp logi
lags 1 to 4
det constant
end(system)
estimate(resids=resresids)
compute loglres=%logl
compute logdetres=%logdet

Using RATIO

ratio(degrees=ntotal-%nregsystem,mcorr=mcorr,$
   title="Exogeneity of Real Variables")
# unresids
# resresids

Using %LOGL statistics. The standard LR test of \(-2\times(L(R)–L(U))\) is adjusted by multiplying by \((T–c)/T\) to implement the multiplier correction.

compute teststat=-2.0*(%nobs-mcorr)/%nobs*(loglres-loglunr)
cdf(title="Exogeneity of Real Variables") chisqr $
   teststat ntotal-%nregsystem
```
Chapter 10: Vector Autoregressions

10.5 Orthogonalization

Moving Average Representation

**IMPULSE**, **ERRORS** and **HISTORY** are all based upon the moving average representation of a vector time series:

\[
y_t = X_t \beta + \sum_{s=0}^{\infty} \Psi_s u_{t-s}
\]

where

- \( y \) is an \( M \)-variate stochastic process
- \( X_t \beta \) is the deterministic part of \( y_t \)
- \( \{u_t\} \) is an \( N \)-variate white noise process: if \( t \neq s \), \( u_t \) and \( u_s \) are uncorrelated. Usually \( N=M \), but if there are some linear identities connecting the \( y \) values, \( M \) can be greater than \( N \). \( u \) is called an innovation process for \( y \).

What is Orthogonalization?

There are many equivalent representations for this model: for any non-singular matrix \( G \), \( \Psi \) can be replaced by \( \Psi G \) and \( u \) by \( G^{-1} u \). A particular version is obtained by choosing some normalization.

If \( \Psi \) is normalized to be the identity matrix, each component of \( u_t \) is the error that results from the one step forecast of the corresponding component of \( y_t \). These are the non-orthogonal innovations in the components of \( y \); non-orthogonal because, in general, the covariance matrix \( \Sigma = E(u_t u_t') \) is not diagonal.

It is often more useful to look at the moving average representation with orthogonalized innovations. If we choose any matrix \( G \) so that

\[
G^{-1} \Sigma G^{-1} = I
\]

then the new innovations \( v_t = G^{-1} u_t \) satisfy \( E(v_t v_t') = I \). These orthogonalized innovations have the convenient property that they are uncorrelated both across time and across equations. Such a matrix \( G \) can be any solution of \( GG' = \Sigma \). There are many such factorizations of a positive definite \( \Sigma \), among them:

- those based on the Choleski factorization, where \( G \) is chosen to be lower triangular (next page),
- those based upon eigen decompositions,
- structural decompositions of the form suggested by Bernanke (1986) and Sims (1986).
Why Orthogonalize?

Orthogonalized innovations have two principal advantages over non-orthogonal ones:

- Because they are uncorrelated, it is very simple to compute the variances of linear combinations of them.
- It can be rather misleading to examine a shock to a single variable in isolation when historically it has always moved together with several other variables. Orthogonalization takes this co-movement into account.

The greatest difficulty with orthogonalization is that there are many ways to accomplish it, so the choice of one particular method is not innocuous. The Bernanke-Sims style decompositions, described later in this section, are designed to overcome some of the objections to the methodology by modeling the decomposition more carefully.

10.5.1 Choleski Factorization

The standard orthogonalization method used by RATS is the Choleski. Given a positive-definite symmetric matrix $\Sigma$, there is one and only one factorization into $GG'$ such that $G$ is lower triangular with positive elements on the diagonal. This is the Choleski factorization.

We can obtain several related decompositions by reordering rows and columns of $\Sigma$. For instance, if the $\Sigma$ matrix is

$$
\begin{bmatrix}
1.0 & 4.0 \\
4.0 & 25.0
\end{bmatrix}
$$

its Choleski factor is

$$
\begin{bmatrix}
1.0 & 0.0 \\
4.0 & 3.0
\end{bmatrix}
$$

If we interchange variables 1 and 2 in the covariance matrix,

$$
\begin{bmatrix}
25.0 & 4.0 \\
4.0 & 1.0
\end{bmatrix}
$$

has a factor of

$$
\begin{bmatrix}
5.0 & 0.0 \\
0.8 & 0.6
\end{bmatrix}
$$

If we switch the rows on the latter array, a new factor of the original $\Sigma$ is obtained:

$$
\begin{bmatrix}
0.8 & 0.6 \\
5.0 & 0.0
\end{bmatrix}
$$

We describe the first factorization of $\Sigma$ as the decomposition in the order 1-2 and the second as the order 2–1.

The Choleski factorization is closely related to least squares regression. If we are decomposing the covariance matrix of a set of variables, the $i$th diagonal element of the factor is the standard error of the residual from a regression of the $i$th variable on variables 1 to $i$–1.
Chapter 10: Vector Autoregressions

There is a different factorization for every ordering of the variables, so it will be next to impossible to examine all of them for systems with more than three variables. Usually, you will decide upon the ordering based mostly upon a “semi-structural” interpretation of the model: you might feel, for instance, that within a single time period, movements in A precede movements in B, so A should precede B in the ordering.

Note that when the residuals are close to being uncorrelated, the order of factorization makes little difference. With low correlation, very little of the variance in a variable can be explained by the other variables.

10.5.2 Structural Decompositions

The Choleski factorizations of the previous page suffer from the problem of imposing a “semi-structural” interpretation on a mechanical procedure. For instance, the “money” innovation is different (sometimes radically so) if you put money first in the ordering than if you put it last. There will rarely be a nice, neat, publicly acceptable way to order the VAR variables.

Bernanke (1986) and Sims (1986) have proposed alternative ways of looking at the factorization problem which impose more of an economic structure. These have been dubbed “structural VARS” or “identified VARS.” The theory of these is developed more fully in Hamilton (1994), and Enders (2003) covers them from a practical standpoint.

First, note that in the VAR model

\[ y_t = \sum_{s=1}^{L} \Phi_s y_{t-s} + u_t \quad ; \quad E(u_t u_t') = \Sigma \]

the lag coefficients \( \Phi_s \) can be estimated by single equation OLS regardless of any restrictions on the \( \Sigma \) matrix. Suppose now that we write down a model for the (non-orthogonal) innovation process \( u \), such as

\[
\begin{align*}
    u_{1t} &= v_{1t} \\
    u_{2t} &= \gamma u_{1t} + v_{2t} \\
    u_{3t} &= \delta u_{1t} + v_{3t}
\end{align*}
\]

where we assume the \( v \)'s are orthogonal. This puts a restriction on \( \Sigma \): there are six free elements in \( \Sigma \) (in general, there are \( N(N+1)/2 \) and only five in this setup: \( \gamma, \delta \) and the variances of the \( v \)'s. We can obtain a related Choleski factorization by adding either \( u_{2t} \) to the \( u_{3t} \) equation (order 1–2–3) or vice versa (order 1–3–2). The above model causes \( u_2 \) and \( u_3 \) to be related only through \( u_1 \).

In general, if we write the innovation model as

\[ Au_t = v_t \quad ; \quad E(v_t v_t') = D \quad ; \quad D \text{ diagonal} \]
and assume Normal residuals, we need to maximize over the free parameters in \( A \) and \( D \) the likelihood-based function:

\[
\frac{T}{2} \left( \log |A|^2 - \log |D| \right) - \frac{T}{2} \text{trace}(D^{-1}ASA')
\]

where \( S \) is the sample covariance matrix of residuals.

When you write them down, innovation models *look* a lot like standard simultaneous equations models. However there are no “exogenous variables.” Identification comes from the restriction on the covariance of the \( v \)'s. If you want to check identification, you need to show there is no matrix \( F \) such that

- \( FA \) has the same structure as \( A \) (1’s on the diagonal, 0’s in the same places), and
- \( FDF' \) is diagonal

Unfortunately, there are no simple counting rules like the order condition to verify identification. If you have more than \( N(N-1)/2 \) free parameters in \( A \), your model is definitely *not* identified, but it may be unidentified even with fewer.

In his book, Hamilton derives conditions for *local* identification. However, the major problem faced in working with structural decompositions is a failure of *global* identification. For example, if we start with \( D \) being the identity and \( A \) matrix

\[
\begin{bmatrix}
1.0 & 0.5 & 0.0 \\
0.0 & 1.0 & 1.2 \\
-0.3 & 0.0 & 1.0
\end{bmatrix}
\]

there is an alternate factorization with

\[
\begin{bmatrix}
1.0 & 1.253 & 0.0 \\
0.0 & 1.0 & 1.925 \\
-2.303 & 0.0 & 1.0
\end{bmatrix}
\]

The likelihood function will have a peak at each of these. Which one will be selected by the estimation process will depend upon the initial guess values. Which one is meaningful is up to you and whatever prior information you might have. These two, for instance, have identical signs for the coefficients, but the relative size of the coefficients is quite different, particularly in the first and third equations. Also, the variances are identical in the first factorization and quite different (2.506,1.604 and 7.678) in the second. Also complicating the process of estimating the model is that the likelihood function can have “ridges” where the function value falls short of the maximum. Be sure that you try some different initial values—don’t rely on a single shot at estimation.

While this example demonstrates how ill-mannered the likelihood function for structural models can be, it should be noted that the “model” that this describes is not likely to be useful anyway. Consider that this says that residual 1 comes from variables 1 and 2, 2 comes from 2 and 3 comes from 1 and 3, *and* that despite the model practically being circular, the derived residuals are uncorrelated. The require-
ment that the model produce orthogonal shocks is a very strong one. The closer your
model is to a recursive, or block recursive form, where you can rely upon orthogonality
by construction, the more likely that you will be able to get a satisfactory result.
Sims’ 1986 model, for instance, is nearly block recursive, with investment in one
block, interest rates and money in a second and GNP, prices and unemployment in a
third. Aside from some zero restrictions, the only deviation from block recursiveness
is in the monetary sector, where the equation identified as “money demand” depends
upon GNP and prices, variables pulled up from the third block.

In RATS, structural VARs are estimated using the instruction `CVMODEL`. This actually
accepts a broader class of models than was described above. The general form is

\[(6) \quad A\mathbf{u}_t = B\mathbf{v}_t \quad ; \quad E(\mathbf{v}_t\mathbf{v}_t') = \mathbf{D} \quad ; \quad \mathbf{D} \text{ diagonal}\]

Of course, \(B=I\) gives the model from before. Typically, a model will use just one of \(A\)
and \(B\). A “B” model would come from a view that you know what the orthogonal
shocks are and are using the \(B\) matrix to tell which variables they hit. A model using
both \(A\) and \(B\) would likely have just a few well placed free coefficients in \(B\) to allow
for correlations among residuals in structural equations. For instance, if you have
two structural equations for \(\mathbf{u}\), but are unwilling to restrict them to having uncorre-
lated residuals, a \(B\) matrix with a non-zero coefficient at the off-diagonal location
linking the two will allow them to be correlated.

Before running `CVMODEL`, you need to create a `FRML` which describes your \(A\) or \(B\)
matrix. This must be declared as a `FRML[RECT]`, which is a formula which produces
a rectangular matrix. Whatever free parameters you will have in this also need to be
put into a parameter set using `NONLIN`. For instance, for the small model:

\[
\begin{align*}
    u_{1t} &= v_{1t} \\
    u_{2t} &= \gamma u_{1t} + v_{2t} \\
    u_{3t} &= \delta u_{1t} + v_{3t}
\end{align*}
\]

the set up would be

```
nonlin gamma delta
dec frml[rect] afrml
frml afrml = |1.0,0.0,0.0| -gamma,1.0,0.0| -delta,0.0,1.0|
```

We also need the covariance matrix of residuals, which will usually come from an
`ESTIMATE` instruction on a VAR. Now the covariance matrix itself is a sufficient
statistic for estimating the free coefficients of the model. However, in order to obtain
standard errors for the coefficients, or to test overidentifying restrictions, we need to
know the number of observations. RATS keeps track of the number of observations on
the most recent estimation, so if you have just estimated the VAR, you won’t need to
do anything about that. However, if the covariance matrix was estimated separately,
you should use the option `OBS=number of observations` on `CVMODEL`. 
CVMODELL provides three choices for the (log) maximand that it uses in estimating the free coefficients in (6). All are based upon Normally distributed ν’s, but differ in their handling of the D matrix. They fall into two general forms:

\[
(7) \quad \frac{T-c}{2} \left\{ \log |A|^2 - \log |B|^2 \right\} - \left( \frac{T-c}{2} + \delta + 1 \right) \sum_i \log \left( B^{-1} ASA'B^{-1} \right)_{ii}
\]

\[
(8) \quad \frac{T-c}{2} \left\{ \log |A|^2 - \log |B|^2 \right\} - \frac{T}{2} \sum_i \left( B^{-1} ASA'B^{-1} \right)_{ii}
\]

With \( c = 0 \) and \( \delta = -1 \) in (7), you have the concentrated likelihood. This form is selected by using \text{DMATRIX=CONCENTRATE}, which is the default. Other values of \( \delta \) have D integrated out. This is \text{DMATRIX=MARGINALIZED}, combined with \text{PDF=value of \( \delta \)}. This uses a prior of the form \( |D|^{-\delta} \) (PDF stands for Prior Degrees of Freedom). With \( c = 0 \) in (8), you have the likelihood with \( D = I \). This is selected with \text{DMATRIX=IDENTITY}. This requires a different parameterization of the basic factoring model than the other two. The concentrated and marginalized forms both assume that the parameterization of the A and B matrices includes a normalization, generally by putting 1’s on the diagonal. With \text{DMATRIX=IDENTITY}, the normalization is chosen by making the ν’s have unit variance, so the diagonal in A or B has to be freely estimated. This was the choice in Sims and Zha (1999), as they had some concern that the normalization of one of their equations was not innocuous. The maximum likelihood estimates aren’t affected by the choice of normalization, but the Monte Carlo integration process (chapter 13) is.

In all of these, c is for correcting the degrees of freedom if you’re examining the posterior density. You provide the value of c using the option \text{DFC=value of c}.

CVMODELL offers three estimation methods: BFGS, SIMPLEX and GENETIC. BFGS (which is described in Section 7.2) is the only one of the three that can estimate standard errors. The others can only do point estimates. However, BFGS, as a hill-climbing method, depends crucially on whether you are starting on the right “hill.” If you want to play safe, start out with the GENETIC method, which explores more broadly (but, unfortunately, also more slowly). You can use the \text{PMETHOD} and \text{PITERS} options to choose a preliminary estimation method before switching to BFGS.

You can also use \text{CVMODELL (METHOD=EVAL)} to evaluate the log likelihood at the initial guess values. Note that while the integrating constants aren’t included in (7) and (8), they \textit{are} included in the log likelihood that CVMODELL produces.

The following code estimates the original structural VAR model. SIGMA is the (previously estimated) covariance matrix of residuals:

```plaintext
compute gamma=0.0,delta=0.0
cvmodel(method=bfgs,a=afrml) sigma
```
This is the output produced:

Covariance Model-Concentrated Likelihood - Estimation by BFGS
Convergence in 8 Iterations. Final criterion was 0.0000001 < 0.0000100
Observations 250
Log Likelihood -859.08482342
Log Likelihood Unrestricted -856.27298288
Chi-Squared(1) 5.62368107
Significance Level 0.01771939

<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. GAMMA</td>
<td>0.4870840231 0.0633546856</td>
<td>7.68821</td>
<td>0.00000000</td>
<td></td>
</tr>
<tr>
<td>2. DELTA</td>
<td>0.6682331021 0.0607874856</td>
<td>10.99294</td>
<td>0.00000000</td>
<td></td>
</tr>
</tbody>
</table>

In the header, it displays the log likelihood of the estimated model, and the log likelihood of an unrestricted model. For a just identified model, those should be equal, or you got stuck on a ridge. If the model is overidentified (which this one is), it produces a likelihood ratio test for the overidentifying restrictions. In this case, the restriction is significant at just about the 1% level.

The log likelihood is accessible for further analysis as %FUNCVAL or %LOGL, while the test statistic and significance level are %CDSTAT and %SIGNIF. To get the factor of the Σ that you will need for impulse responses, use the FACTOR option on CVMODEL.

CVMODEL is even more general than it appears. Suppose that you have a model which generates a direct model for the covariance matrix, not an “A” or “B” matrix. You can provide a V formula (a FRML[SYMMETRIC]) which gives the covariance matrix itself. The likelihood used is

\[
\frac{T-c}{2} \log |V^{-1}| - \frac{T}{2} \text{trace}(SV^{-1})
\]

For instance, a simple “factor” model would model the covariance matrix as \( \Lambda \Lambda' + D \), where \( \Lambda \) are the loadings on the orthogonal factors and \( D \) is a diagonal matrix. With \( \Lambda \) constructed by a vertical concatenation of LLEAD and LREM, and \( D \) being a VECTOR, the following would estimate the free parameters in a fit to the covariance matrix \( R \).

```
cvmodel(v=%outerxx(llead~~lrem)+%diag(d.*d),obs=%nobs) r
```
10.5.3 Blanchard-Quah/Long-Run Restrictions

Technically, the decomposition proposed by Blanchard and Quah (1989) is a special case of a structural VAR model. However, it reflects a completely different approach to the task of interpreting a VAR.

Blanchard and Quah (BQ) start with (assumed to be) orthogonal shocks. In a two variable model, one is assumed to represent a “supply” shock and one a “demand” shock. It is expected that both shocks will affect both variables, meaning that we must find a restriction somewhere else in order to identify the shocks. The BQ assumption is that the “demand” shock will have no permanent effect on one of the two variables.

Now the question arises: what does it mean for a shock to have only a temporary effect on a variable? This has to be defined in a way that produces a restriction that can be imposed upon a factorization. BQ accomplish this by assuming that the target variable is an integrated variable which is put into the VAR in differenced form.

Translating the moving average representation on the difference to the corresponding levels requires merely taking running sums of the MAR coefficients. That is, if

\[(10) \quad (1 - L)y_t = \sum_{s=0}^{\infty} \Psi_s v_{t-s} , \text{ then} \]

\[(11) \quad y_t = \sum_{s=0}^{\infty} \Psi_s^* v_{t-s} , \text{ where } \Psi_s^* = \sum_{l=0}^{s} \Psi_l \]

The restriction that a component of \( v \) has no long-run effect on a component of \( y \) can now be written

\[(12) \quad \sum_{l=0}^{\infty} \Psi_l(i, j) = 0 \]

This is still not in a form where it can be imposed as a restriction on the factorization. However, if the VAR has been transformed so as to have an invertible autoregressive representation (in effect, all the variables in it are now stationary), then the infinite sums of the moving average representation can be obtained by inverting the autoregressive representation. If we write the desired model in the inverted form

\[(13) \quad y_t = \Phi(L)^{-1} B v_t , \text{ with } B B' = \Sigma, \ E v_t v_t' = I \text{ and } \Phi(L) = 1 - \sum_{s=1}^{l} \Phi_s L^s \]

then the matrix of sums of MA coefficients (the long-run response matrix) is

\[(14) \quad \Phi(1)^{-1} B \]

\( \Phi(1) \) can be obtained after an ESTIMATE instruction as %VARLAGSUMS, and more generally can be computed using %MODELLAGSUMS(model).

The BQ restriction on \( B \) (in a two-variable system) is that one of the four elements of this be zero. For simplicity, assume that this is the (1,2) element. (To make it the 1
row, you need to order the variables correctly when setting up the VAR). Now \( \Phi(1)^{-1} B \) is a factor of \( \Phi(1)^{-1} \Sigma \Phi(1)^{-1} \) and the restriction can be imposed by making it the Choleski factor of that matrix. The specialized function \( \%\text{BQFACTOR} \) does the calculation of the required factor \( B \).

\[ \text{compute } \text{bqfactor} = \%\text{bqfactor}(%\text{sigma},%\text{varlagsums}) \]

This generalizes to \( N \) variables by choosing a restriction that the long-run response matrix is lower triangular; that’s what you’ll get if you apply \( \%\text{BQFACTOR} \) to a larger system. That, however, is unlikely to be a very interesting structure. More commonly, you’ll have a combination of restrictions on the long-run matrix and restrictions on \( B \) itself. This gives you a combination of short (actually contemporaneous) and long run restrictions.

Unlike the models which are entirely contemporaneous, models with long-run restrictions cannot easily be handled if they’re over-identified. Because the maximum likelihood estimates of the lag coefficients are just the OLS estimates regardless of the value of \( \Sigma \), they can be concentrated out, allowing us to focus only on modelling the contemporaneous relationship. For a just-identified model of \( \Sigma \), long-run restrictions don’t restrict the lag coefficients (even though they’re a function of them), since we have enough freedom in the \( B \) matrix to make the restrictions work while achieving the overall maximum for the likelihood. (All just-identified structural VAR’s have the same likelihood). If the model is over-identified, however, it’s almost certain that you can achieve a higher likelihood by adjusting the lag coefficients. While this could be done using \( \text{NLSYSTEM} \), it involves estimating all the coefficients subject to rather nasty non-linear constraints.

If (as is typical) the restrictions in the model are done entirely by zero restrictions on elements of \( B \) and elements of \( \Phi(1)^{-1} B \), then these can be combined into the form

\[ (15) \quad R \text{vec}(B) = 0 \]

where \( R \) is a \( N(N-1)/2 \times N^2 \) matrix. This can be inverted to create the unrestricted parameters \( \Theta \) where

\[ (16) \quad \text{vec}(B) = R^\perp \Theta \]

where \( R^\perp \) is the orthogonal complement of \( R \) (computed with the \( \%\text{PERP} \) function). The procedure \( \%\text{ShortAndLong} \) (on \( \text{SHORTANDLONG.SRC} \)) can be used to get the factor matrix \( B \). Input to this are the \( \Phi(1)^{-1} \), \( \Sigma \) and pattern matrices for the long and short run restrictions. The pattern matrices have non-zero elements in the slots which aren’t restricted, and zeros in the slots which are restricted to be zero. Each row represents a variable and each column represents a shock. So a 0 in the 1,2 slot of the long-run pattern means that shock 2 is restricted to have no long-run effect on variable 1.

The following is an example of the use of \( \%\text{ShortAndLong} \). It’s important to make sure that you don’t just have the restriction count correct (there should be
Chapter 10: Vector Autoregressions

$N(N-1)/2$ zeros to just identify the model, but that the combination of restrictions is able to distinguish the shocks: if you look at a pair of columns, there has to be some difference in the restrictions. Note also that you can't have a full row or a full column of zeros.

```
dec rect lr(4,4)
dec rect sr(4,4)
input sr
  . 0 0 .
  . 0 . .
  . . . .
  . . . .
input lr
  . 0 0 0
  . . . .
  . . . .
  . . . .
@shortandlong(sr=sr,lr=lr,masums=inv(%varlagsums),factor=b) %sigma
```

While you could use any non-zero value to represent the non-zero slots, we’ve found that using the . (missing value) makes this easier to read and understand.

While `@ShortAndLong` can’t estimate a model unless it’s just identified, you can input into it an underidentifying set of restrictions so it can compute the representation (16) in terms of a reduced number of parameters. To do this, add the options NOESTIMATE and RPERP=R Perp Matrix. You can then use this with `CVMODEL` to estimate with additional types of restrictions. This, for instance, has two short-run, three long-run and one additional restriction that variable 3 not enter shock 3.

```
input sr
  . 0 0 .
  . . . .
  . . . .
  . . . .
input lr
  . 0 0 0
  . . . .
  . . . .
  . . . .
@shortandlong(sr=sr,lr=lr,masums=inv(%varlagsums),factor=b) %sigma
compute bv=%vec(%bqfactor(%sigma,%varlagsums))
compute [vect] theta=%ginv(rperp)*bv
dec frml[rect] bf af
frml bf = %vectorect(rperp*theta,4)
frml af = inv(bf(0))
nonlin(parmset=base) theta
nonlin(parmset=r8) af(0)(3,3)==0.0
cvmodel(parmset=base+r8,b=bf,iters=400) %sigma
```
10.5.4 Other Factorizations

All of the examples seen so far have produced a complete factorization of the covariance matrix. An alternative approach is to isolate one shock with certain characteristics. This might be the requirement that a shock produce certain responses, or that a shock is a particular linear combination of the non-orthogonal shocks. The Blanchard-Quah factorization is actually a form of this: one shock (the demand shock) has the property that it has a zero long-run response, the other is just whatever shock is required to complete a factorization.

You can construct a factorization around any single (non-zero) shock; some rescaling of it will be part of a factorization. Why do we need a factorization if we’re just interested in the one shock? This is mainly because the decomposition of variance (Section 10.6) can only be computed in the context of a full factorization. In general, there will be many ways to complete the factorization, but the fraction of the variance explained by the shock of interest will be the same for all of them.

Using ForcedFactor

The procedure FORCEDFACTOR (from the file FORCEDFACTOR.SRC) computes a factorization which includes a (scale of a) specified column in the factorization (which forces an orthogonalized component to hit the variables in a specific pattern), or which includes a (scale of a) specified row in the inverse of the factorization (which forces an orthogonalized component to be formed from a particular linear combination of innovations). The syntax for this is explained in the procedure file. For example, in a four variable system where the first two variables are interest rates:

```
@ForcedFactor sigma ||1.0,-1.0,0.0,0.0|| f1
@ForcedFactor(force=column) sigma ||1.0,1.0,0.0,0.0|| f2
```

F1 will be a factorization where the first orthogonal component is the innovation in the difference between the rates. F2 will be a factorization where the first orthogonal component loads equally onto the two interest rates, and hits none of the other variables contemporaneously.

Another example is from King, Plosser, Stock and Watson (1991). They need a factorization which hits the three variables in the system equally in the long run. An initial calculation comes up with a row in the inverse of the factor that produces the correct long run loadings. FORCEDFACTOR then creates the full factor.

```
compute atilde=||1.0|1.0|1.0||
compute d=inv(%innerxx(atilde))*tr(atilde)*lrsum
@forcedfactor(force=row) %sigma d f
```

ForcedFactor will also allow you to control more than one column in the factorization, but the columns other than the first will be linear combinations of itself and the columns to its left. (That is, you can control the space spanned by some set of columns, but not the columns themselves).
10.5.5 Examples

The interpretations of the results from the instructions ERRORS, IMPULSE and HISTORY all depend crucially on the factorization chosen for the covariance matrix of residuals. You can provide the instruction with the SYMMETRIC array $\Sigma$ using the VCV matrix parameter that is available on each of these instructions. That causes RATS to do a Choleski decomposition in the order that the equations were listed. If you are putting the model in with supplementary cards, you can reorder the decomposition by rearranging the supplementary cards. That process is described on the next page. First, however, we will describe how to input a general decomposition.

The FACTOR option

In general, we need a matrix $G$ satisfying $GG' = \Sigma$. With this computed, you use the option FACTOR=G on the instruction.

For a Structural VAR

You get $G$ using the FACTOR option on CVMODEL.

```r
nonlin gamma delta
dec frml[rect] afrml
frml afrml = ||1.0,0.0,0.0|-gamma,1.0,0.0|-delta,0.0,1.0||
compute gamma=0.0,delta=0.0
cvmodel(method=bfgs,factor=sfactor,a=afrml) %sigma
errors(model=model3,factor=sfactor,steps=24)
```

For a Blanchard-Quah Decomposition

You get $G$ by direct calculation as described in Section 10.5.3. This relabels the shocks to match their interpretations.

```r
estimate
compute bqfactor=%bqfactor(%sigma,%varlagsums)
impulse(model=rcmodel,factor=bqfactor,steps=80,results=responses,$
labels=||"Permanent","Transitory"||)
```

For an Alternative Choleski Ordering

The %PSDFACTOR(sigma,order) function computes $G$. This takes as its arguments the covariance matrix and a vector of integers describing the ordering.

```r
errors(model=model4,v=v,steps=24)
compute dec3241=%PSDFACTOR(vsigma,||3,2,4,1||)
errors(model=model4,factor=dec3241,steps=24)
compute dec4321=%PSDFACTOR(vsigma,||4,3,2,1||)
errors(model=model4,factor=dec4321,steps=24)
```
Reordering Equations

There are two ways to provide these instructions with the system that you wish to analyze: with the `MODEL` option, as was demonstrated on the last page, or with supplementary cards listing the equations. The main set of supplementary cards for each of the instructions has the form:

```
# equation  series  newstart  column
```

`column` is the column of the covariance matrix $\Sigma$ that corresponds to this equation (by default, `column` is the supplementary card position—`column`=1 for card one, `column`=2 for card two, etc.). For instance, suppose you set up and estimate the system with:

```r
system 1 to 4
variables gnp m1 cpr ppi
lags 1 to 4
det constant
end(system)
estimate(cvout=vsigma)
```

Equations 1,2,3 and 4 have dependent variables `GNP`, `M1`, `CPR` and `PPI`, respectively. The first column of $\text{VSIGMA}$ is associated with equation 1, the second column with equation 2, etc. The set of supplementary cards:

```
# 1
# 2
# 3
# 4
```

requests decomposition in the order $\text{GNP}\rightarrow\text{M1}\rightarrow\text{CPR}\rightarrow\text{PPI}$, while

```
# 3 * * 3
# 2 * * 2
# 4 * * 4
# 1 * * 1
```

results in decomposition in the order $\text{CPR}\rightarrow\text{M1}\rightarrow\text{PPI}\rightarrow\text{GNP}$.

LIST and CARDS

Notice that in the examples above, the supplementary cards tend to follow a simple pattern. The `LIST` and `CARDS` utility (both under `LIST` in the Reference Manual) are useful in handling batches of supplementary cards like this. For example, you could replace the examples above with:

```r
list ieqn = 1 2 3 4
errors(v=vsigma,steps=20) 4
cards ieqn * * ieqn
list ieqn = 3 2 4 1
errors(v=vsigma,steps=20) 4
cards ieqn * * ieqn
```
10.6 Using IMPULSE and ERRORS

Background

**IMPULSE** computes the responses of the system to particular initial shocks. It has a large array of options for specifying the shocks, but most of these (such as **PATHS** and **MATRIX**) are used more naturally with **FORECAST**. The VAR methodology works just with first period standard error shocks.

**ERRORS** uses the same information as **IMPULSE**, but produces its output in a different form. It decomposes the forecast error variance into the part due to each of the innovation processes. While **IMPULSE** will accept any form of innovation process, **ERRORS** requires orthogonalization, as the decomposition is meaningless without it.

You should always use **IMPULSE** and **ERRORS** together. For instance, you may note an interesting and unexpected response using **IMPULSE**. Before you get too excited, examine the variance decomposition with **ERRORS**. You may find that it has a trivial effect.

As noted earlier, you can use the **VAR (Forecast/Analyze)** wizard on the **Statistics** menu to analyze both the impulse responses and the variance decomposition.

Technical Information

If we look at the moving average representation of a vector time series:

\[
\begin{align*}
\mathbf{y}_t &= \mathbf{X}_t \beta + \sum_{s=0}^{\infty} \Psi^s \mathbf{v}_{t-s} = \mathbf{X}_t \beta + \sum_{s=0}^{\infty} \Psi^s \mathbf{G} \mathbf{v}_{t-s} , \\
\mathbf{E} \mathbf{u}_t \mathbf{u}_t' &= \Sigma \\
\mathbf{E} \mathbf{v}_t \mathbf{v}_t' &= \mathbf{I}
\end{align*}
\]

**IMPULSE** computes the \( \Psi^s \) or the \( \Psi^s \mathbf{G} \) (for orthogonalized innovations). These are organized as \( N^2 \) series, although you can do shocks to one innovation at a time.

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

\[
\begin{align*}
\sum_{s=0}^{K-1} \Psi^s \mathbf{G} \mathbf{v}_{t-s}
\end{align*}
\]

As \( \mathbf{v} \) has been assumed to be uncorrelated both across time and contemporaneously, the covariance matrix of the \( K \)-step forecast is

\[
\begin{align*}
\sum_{s=0}^{K-1} \Psi^s \mathbf{G} \mathbf{G}' \Psi^s
\end{align*}
\]

This doesn’t depend upon \( \mathbf{G} \), as long as \( \mathbf{G} \mathbf{G}' = \Sigma \). We can isolate the effect of a single component of \( \mathbf{v} \) by rewriting the sum as

\[
\begin{align*}
\sum_{s=0}^{K-1} \sum_{i=1}^{N} \Psi^s \mathbf{G} \mathbf{e}(i) \mathbf{e}(i)' \Psi^s = \sum_{i=1}^{N} \sum_{s=0}^{K-1} \Psi^s \mathbf{G} \mathbf{e}(i) \mathbf{e}(i)' \Psi^s , \text{ where } \mathbf{e}(i) \text{ is the } i\text{th unit vector.}
\end{align*}
\]

This decomposes the variance-covariance matrix of forecast errors into \( N \) terms, each of which shows the contribution of a component of \( \mathbf{v} \) over the \( K \) periods.
Graphing Impulse Responses

The moving average representation (MAR) of a model is simply the complete set of impulse responses. These are most usefully presented graphically. There are three logical ways to organize graphs of the MAR. The first two are demonstrated in Example 10.3, the last in Example 13.2.

- A single graph can have responses of all variables to a shock in one variable. If the variables are measured in different units, it is advisable to standardize the responses: if you divide the response of a variable by the standard deviation of its residual variance, all responses are in fractions of standard deviations.

  *When the variables are in comparable units and a comparison of actual values is important, it is better to graph unscaled responses.* For example, you would want to compare interest rates and the rate of inflation without scaling.

- A single graph can have responses of one variable to shocks in all the variables. There is no problem of scale in this case.

- You can use a matrix of small graphs, each with only a single response. This looks nicer, but is more difficult to set up, as you must be sure that all graphs showing the responses of a single variable use the same **MAXIMUM** and **MINIMUM** values. Otherwise, very small responses will be spread across the entire height of a graph box and look quite imposing.

Confidence Bands

Point estimates alone of impulse responses may give a misleading impression. You might note a response whose sign is unexpected. Is this truly interesting, or is it just a statistical fluke? This can be answered, in part, by examining the corresponding error decomposition. If the innovation you’re examining in fact explains a trivial amount of the variable, then it isn’t really meaningful.

But many responses can’t quite so easily be dismissed as uninteresting. **IMPULSE** produces a moving average representation from the point estimates of a VAR. Since the coefficients of the VAR aren’t known with certainty, neither are the responses. There are three principal methods proposed for computing confidence bands or standard errors for impulse responses:

1. Monte Carlo integration
2. Analytical derivatives
3. Bootstrapping

Despite the appealing sound of the phrase, “analytical derivatives” are based upon a linearization which becomes increasingly inaccurate as the number of steps grows and the response functions become increasingly non-linear. We have a very strong preference for the use of Monte Carlo integration, which is demonstrated in Examples 13.2. See Sims and Zha (1999) for a discussion of these issues.
Chapter 10: Vector Autoregressions

Interpreting the Decomposition of Variance

Decomposition of Variance for Series CANRGDPS

<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>0.011511805</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>0.013088464</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>0.014910398</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>

This is part of one of the tables produced by an **ERRORS** instruction for a six-variable VAR. There will be one such table for each endogenous variable.

The first column in the output is the standard error of forecast for this variable in the model. 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. However, the more interesting information is at the longer steps, where the interactions among the variables start to become felt. We have truncated this table to 10 lags to keep its size manageable, but ordinarily you should examine at least five years worth of steps.

The above table suggests the following:

- The three principal factors driving CANRGDPS are itself, USARGDPS and CANCD90D, which is a short interest rate.
- The importance of USARGDPS is fairly constant across the range. Because this variable was first in the ordering, it isn't clear (from examining just this one ordering) whether this is an artifact of the ordering only.
- Innovations in CANCD90D take almost six periods to have an effect but quickly become the prime mover.
- The other three variables (CANM1S, CANUSXSR and CANCPINF) have negligible explanatory power for CANRGNP.

If you want more information on how USARGDPS and CANCD90D affect CANRGDPS, you need to look at the impulse response functions.
Choosing Orderings

If you work with Choleski factorizations, the orderings that you should examine depend upon the set of questions you want to answer, and upon the structure of the covariance matrix of residuals.

- Variables that you don’t expect to have any predictive value for other variables should be put last: for instance, local variables in a model with national variables.
- By definition, the first variable in the ordering explains all of its one-step variance.
- The one-step variance will be nearly 100% due to own innovations if there is little correlation between the residuals of a variable and the residuals of variables that appear before it in the ordering.
- When there is substantial correlation among innovations in variables, the decomposition of one-step variance depends strongly on the order of factorization.

To determine whether a variable behaves exogenously, put the variable first in the ordering. The variance in an exogenous variable will be explained primarily by own innovations. The meaning of “primarily” depends upon the number of variables in the system: 50% is quite high in a six variable system. Remember that if the covariance matrix is nearly diagonal, the decomposition of variance will be fairly robust to changes of order.

Focusing on Two Variables

When there is high correlation between innovations in two variables, run a pair of decompositions with the two variables placed next to each other, changing only the positions of those two variables from one ordering to the next. Since the combined explanatory power of the two variables is independent of which one comes first, how the variance is split between them can be examined.

Usually, most of the variance will be attributed to whichever variable comes first. If this is true for both orderings, we can draw no conclusions. If one variable does much better when second than the other, you have evidence that variable is the causative factor, and the other moves with it closely. If most of the power is attributed to whichever variable is second, some linear combination of the two variables, perhaps the sum or difference, is the truly important factor. See, for instance, Litterman and Weiss (1983).

In the example shown for the decomposition of variance, there is the question of whether or not the influence of USARGDPS is mainly due to its placement in the ordering before CANRGDPS. That can be answered by switching the ordering around to put CANRGDPS earlier. This reordering pretty much wipes out the fraction of variance due to USARGDPS.
Example 10.3 Graphing an Impulse Response Function

This is the program IMPULSES.PRG, which computes and graphs impulse response functions in two different formats. For most cases, you can just use the procedure VARIRF which does all of this for you, with options for selecting which grouping method you want to use. However, we think it is helpful to see how VARIRF works, in case you want to produce your own variations.

If you’re interested in computing standard error bands or some other type of confidence limit on the estimates, see the examples in Chapter 13 and the MONTEVAR procedure.

```
open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpgdpcchs
canexpgdpdfs canm1s canusxsr usaexpgdpcch *
set logcangdp = log(canexpgdpcchs)
set logcandefl = log(canexpgdpdfs)
set logcanm1 = log(canm1s)
set logusagdp = log(usaexpgdpcch)
set logexrate = log(canusxsr)

compute neqn = 6
compute nlags = 5
compute nsteps = 24

system(model=canmodel)
variables logusagdp logexrate can3mthpcp $
     logcanm1 logcangdp logcandefl
lags 1 to nlags
det constant
end(system)

To help create publication-quality graphs, this sets longer labels for the series. These are used both in graph headers and key labels.

compute implabel=|| $ 
   "US Real GDP",$
   "Exchange Rate",$
   "Short Rate",$
   "M1",$
   "Canada Real GDP",$
   "CPI"||

estimate(noprint)
```
The procedure VARIRF does all of the steps shown below, but without requiring all of the detailed programming. It also lets you choose to have the graphs organized BYSHOCKS, BYVARIABLES or both.

@VARIRF(model=canmodel, steps=nsteps, vlabels=implabel, byshocks, byvariables)

And here is how you code this “manually”. First, create some arrays:

```plaintext
declare rect[series] impblk(neqn,neqn)
declare vect[series] scaled(neqn)
declare vect[strings] implabel(neqn)
```

These apply to the GRAPH instructions which are coming up later

```plaintext
list ieqn = 1 to neqn
smpl 1 nsteps
This computes the full set of impulse responses, which are in the series in IMPBLK. IMPBLK(i,j) is the response of variable i to a shock in j.

impulse(model=canmodel, result=impblk, noprint, steps=nsteps)
```

This loop plots the responses of all series to a single series. The response of a series is normalized by dividing by its innovation variance. This allows all the responses to a shock to be plotted on a single scale. Note that these graphs get a bit hard to read with more than five or six variables.

As this program will generate a dozen graphs in a bunch, the WINDOW option is used on the GRAPH instructions to give them descriptive labels in the WINDOW menu.

```plaintext
do i=1,neqn
  compute header="Plot of responses to "+implabel(i)
  do j=1,neqn
    set scaled(j) = (impblk(j,i))/sqrt(%sigma(j,j))
  end do j
  graph(header=header, key=below, klabels=implabel, number=0, window="to_"+implabel(i)) neqn
cards scaled(ieqn)
end do i
```

And this loop graphs the responses of a variable to all shocks. These don’t have to be normalized.

```plaintext
do i=1,neqn
  compute header="Plot of responses of "+implabel(i)
  graph(header=header, key=below, klabels=implabel, number=0, window="of_"+implabel(i)) neqn
cards impblk(i,ieqn)
end do i
```
Example 10.4 Structural Decomposition

The following (example cvmodel.prg) estimates two structural models for the six-variable Canadian VAR. The first of these interprets the orthogonal shocks as two real shocks, two financial market shocks and two nominal shocks. The loadings of these shocks onto the innovations produces a highly overidentified model: there are only six free parameters instead of the fifteen for a just identified model.

\[
\begin{bmatrix}
  u_u \\
  u_c \\
  u_r \\
  u_x \\
  u_m \\
  u_p \\
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 0 & \beta_{u1} & 0 & 0 & 0 \\
  \beta_{r1} & 1 & \beta_{cf1} & 0 & 0 & 0 \\
  0 & 0 & 1 & \beta_{r2} & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 & 0 \\
  \beta_{mr1} & 0 & 0 & 0 & 1 & \beta_{mn1} \\
  0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
  v_r_1 \\
  v_r_2 \\
  v_f_1 \\
  v_f_2 \\
  v_m_1 \\
  v_m_2 \\
\end{bmatrix}
\]

The second model is also overidentified, with nine parameters. This uses the other form of model, where non-orthogonalized shocks are mapped to orthogonalized ones.

\[
\begin{bmatrix}
  1 & 0 & \gamma_{ur} & 0 & 0 & 0 \\
  \gamma_{cu} & 1 & \gamma_{cu} & 0 & 0 & 0 \\
  0 & 0 & 1 & \gamma_{rx} & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 & 0 \\
  0 & \gamma_{mc} & \gamma_{mr} & 0 & 1 & \gamma_{mp} \\
  0 & \gamma_{pc} & \gamma_{pr} & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
  u_u \\
  u_c \\
  u_r \\
  u_x \\
  u_m \\
  u_p \\
\end{bmatrix}
= 
\begin{bmatrix}
  v_u \\
  v_c \\
  v_r \\
  v_x \\
  v_m \\
  v_p \\
\end{bmatrix}
\]

For both of these, an ERRORS instruction is used to compute the decomposition of variance.

open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpdpchs canexpdpdps canmls canusxsx usaexpdpch
* 
set logcangdp = log(canexpdpchs) 
set logcandefl = log(canexpdpdps) 
set logcanml = log(canmls) 
set logusagdp = log(usaexpdpch) 
set logexrate = log(canusxsx) 
*
system(model=canmodel)
variables logusagdp logcangdp can3mthpcp logexrate $
    logcandefl logcanm1
lags 1 to 4
det constant
end(system)
*
estimate(noprint)
*
dec frml[afrml bfrml]
nonlin uf1 cr1 cf1 rf2 mf1 mm2
frml  bfrml = |1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    cr1,1.0,cf1,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    0.0,0.0,1.0,rf2,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    mf1,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
compute uf1=cr1=cf1=rf2=mf1=mm2=0.0
This is estimated by using the genetic method first, then polishing the estimates with BFGS. In practice, you might want to repeat this several times to test whether there are global identification problems.
cvmodel(b=bfrml,method=bfgs,pmethod=genetic,piters=50,$
    factor=bfactor) %sigma
Because the shocks don’t really correspond one-to-one with the variables, the labels option is used on ERRORS to give them the desired labels.
errors(model=canmodel,factor=bfactor,steps=28,window="BModel",$
    labels=||"Real 1","Real 2","Fin 1","Fin 2","Nom 1","Nom 2"||)

nonlin rx ur cu cr pc pr mp mc mr
frml  afrml = |1.0,0.0,0.0,0.0,0.0,0.0|$
    cu,1.0,cr,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    0.0,0.0,1.0,rx,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    0.0,mc,mr,0.0,1.0,mp,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
    0.0,pc,pr,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0|$
compute ur=cu=cr=rx=mc=mr=mp=pc=pr=0.0
cvmodel(a=afrml,method=bfgs,pmethod=genetic,piters=50,$
    factor=afactor) %sigma
errors(model=canmodel,factor=afactor,steps=28,window="AModel")
10.7 Historical Decomposition

Background

**HISTORY** decomposes the historical values of a set of time series into a base projection and the accumulated effects of current and past innovations. You can see, for instance, whether movements in variable $x$ in 2001 were likely the result of innovations in variable $z$ a year earlier.

The historical decomposition is based upon the following partition of the moving average representation

$$y_{T+j} = \sum_{s=0}^{j-1} \psi_s u_{T+j-s} + \left[ X_{T+j} \beta + \sum_{s=j}^{\infty} \psi_s u_{T+j-s} \right]$$

The first sum represents that part of $y_{T+j}$ due to innovations in periods $T+1$ to $T+j$. The second is the forecast of $y_{T+j}$ based on information available at time $T$.

If $u$ has $N$ components, the historical decomposition of $y_{T+j}$ has $N+1$ parts:

- The forecast of $y_{T+j}$ based upon information at time $T$ (the term in brackets).
- For each of the $N$ components of $u$, the part of the first term that is due to the time path of that component.

Presenting Results

**HISTORY** generates a lot of series: $N(N+1)$ of them for a standard VAR. Graphs are the simplest way to present these. The example on the next page creates a separate page of graphs for each variable, with each page having a separate graph for the shocks to each of the other variables. Each of these subgraphs has the actual data, the forecast and the sum of the forecast and the effect of the residuals to that variable. This, for instance, shows the effect of the interest rate on Canadian GDP. We can see that almost the entire difference between actual GDP (the solid line) and the base forecast can be attributed to interest rate shocks.
Example 10.5 Historical Decomposition: Use of HISTORY

This example (HISTORY.PRG) does a historical decomposition of a six-variable Canadian model, which includes GDP for Canada and the U.S., the Canadian-U.S. exchange rate, M1, GDP deflator and a short interest rate. This is written to be generalized easily to other sets of variables. Other than the initial set up of variables, you just need to set the decomposition range (HSTART and HEND) and, perhaps, change the layout of the array of graphs.

```plaintext
open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpdpchds canexpdpds canm1s canusxsr usaexpdpch
*
set logcangdp = log(canexpdpchds)
set logcandefl = log(canexpdpds)
set logcanm1   = log(canm1s)
set logusagdp  = log(usaexpdpch)
set logexrate  = log(canusxsr)
*
compute hstart=2003:1
compute hend  =2006:4
*
compute modelsize=%modelsize(canmodel)
dec vect[int] depvar(modelsize)
dec vect[labels] varlabels(modelsize)
ewise varlabels(i)=%modellabel(canmodel,i)
*
estimate(noprint,noftests,resids=resids)
history(model=canmodel,add,results=history,from=hstart,to=hend)
do j=1,6
    spgraph(hfields=2,vfields=3,window=varlabels(j),$
    header="Historical Decomposition of "+varlabels(j))
do i=1,6
    graph(header="Effect of "+varlabels(i)) 3
    # depvar(j) hstart-4 hend
    # history(1,j)
    # history(1+i,j)
end do j
spgraph(done)
end do i
```
10.8 Cointegration and Error Correction Models

Background

In Section 6.11, we discussed how to test for cointegration. In this section, we describe how to impose it on a (VAR) model. The standard VAR model:

\[ y_t = \sum_{s=1}^{L} \Phi_s y_{t-s} + u_t \]

can always be rewritten in the form

\[ \Delta y_t = \sum_{s=1}^{L-1} \Phi_s^* \Delta y_{t-s} + \Pi y_{t-1} + u_t \]

where \( \Delta y_t \) is the first difference of the \( y \) vector (that is, \( y_t - y_{t-1} \)). If \( \Pi \) is zero, then the VAR can be modelled adequately in first differences. If \( \Pi \) is not zero, then, even if each component of \( y \) has a unit root, a VAR in first differences is misspecified.

If \( \Pi \) is full-rank, there is nothing to be gained by writing the system in form (2) rather than (1). The two are equivalent. The interesting case is where \( \Pi \) is non-zero but less than full rank. In that case, we can write the matrix as

\[ \Pi = \alpha \beta' \]

where \( \alpha \) and \( \beta \) are \( N \times r \) matrices (\( N \) is the number of components in \( y \) and \( r \) is the rank of \( \Pi \)). Note that the decomposition in (3) isn’t unique: for any \( r \times r \) non-singular matrix \( G \), it is also true that

\[ \Pi = (\alpha G)(\beta G^{-1})' \]

Where \( r \) is one, however, (3) is unique up to a scale factor in the two parts.

The instruction \texttt{ECT} allows you to estimate the VAR in the form (2) imposing a reduced rank assumption on the \( \Pi \) term. The \( \beta'y \) represent stationary linear combinations of the \( y \) variables. If the \( y \)'s themselves are non-stationary, they are said to be cointegrated. The instruction name (which is short for Error Correction Terms) comes from the fact that (2) is sometimes called the error correction form of the VAR. See, for instance, Chapter 6 of Enders (2003), Chapter 19 of Hamilton (1994).

Setting up the VAR

To estimate a VAR in error correction form, start out as if you were doing the standard form (1). That is, your variables aren’t differenced, and you count the number of lags you want on the undifferenced variables. Create the error correction equations, and use \texttt{ECT} to add them to the system. You don’t have to fill in the coefficients when setting up the VAR, but you at least need to have created the equations.

The VAR system works somewhat differently in error correction form. The regression output, coefficient vectors, and covariance matrices will be based upon the restricted
form. When the `MODEL` is used in a forecasting instruction such as `IMPULSE` or `FORECAST`, it is substituted back into form (1), so that it analyzes the original variables and not the differenced ones.

In general, you will use `ESTIMATE` to compute the coefficients for the model. You can use `KALMAN` (Section 10.13) to estimate the parameters sequentially, but the Kalman filter does not apply to the cointegrating vector $\beta$—it takes $\beta$ as given by the coefficients in the error correction equation(s), and estimates only the $\Phi_i^*$ and $\alpha$. If you are estimating these coefficients yourself (other than the least squares conditional on $\beta$ that `ESTIMATE` does), use the function `%MODELSETCOEFFS(model, coeffs)`. The coefficient matrix `coeffs` is a RECTANGULAR matrix. Each column gives the coefficients for an equation. In a given equation, the first coefficients are the $\Phi_i^*$. Note that there are $L-1$ lags for each endogenous variable, not $L$, since these equations are in differenced form. Next are the variables listed on the `DETERMINISTIC` instruction, if any. The final coefficients are the loadings $\alpha$ on the error correction terms.

### Setting the Cointegration Vectors

The cointegrating vectors $\beta$ can either be set from theory or estimated. If you want to set them from theory, use the `EQUATION` instruction to create the equation, and use the `COEFFS` option on `EQUATION` or the `%EQNSETCOEFFS` function to set the coefficients. For instance, in Example 6.8, we test whether Purchasing Power Parity is a cointegrating relationship. If (despite the results to the contrary in this dataset), we were to proceed with imposing this during estimation, we would do the following:

```plaintext
equation(coeffs=||1.0,-1.0,-1.0||) rexeq rexrate  
# pusa s pita  
system(model=pppmodel)  
variables pusa s pita  
lags 1 to 12  
det constant  
ect rexeq  
end(system)  
estimate
```

In this case, the error correction equation shows the desired linear combination of the endogenous variables. If, instead, the coefficients of the equilibrium equation are estimated by an auxiliary regression, the error correction equation will have one of the endogenous variables as its dependent variable. For the example above, you would just replace the `EQUATION` instruction (and supplementary card) with:

```plaintext
linreg(define=rexeq) pusa  
# s pita
```

If you’re estimating the cointegrating vector using a procedure which doesn’t normalize a coefficient to one (such as `JOHMLE`), define the equation without a dependent variable:

```plaintext
equation(coeffs=cvector) rexeq *  
# pusa s pita
```
Example 10.6 Error Correction Model

The following (example file ECT.PRG) analyzes a set of three interest rate variables, first testing for cointegration, then imposing it.

RATS does only a limited form of cointegration analysis. For a more complete treatment, the program CATS is available from Estima as an add-on. CATS provides more sophisticated tests for cointegrating rank, model diagnostics and tests on the structures of both the cointegrating vectors and their loadings (and much more).

```
cal(m) 1975:1
open data haverate.rat
data(format=rats) 1975:1 2001:6 ftbs3 ftb12 fcm7

Test for unit roots in the original variables

@dfunit(lags=6) ftbs3
@dfunit(lags=6) ftb12
@dfunit(lags=6) fcm7

Use @JohMLE to test the cointegrating rank and estimate the cointegrating vector corresponding to the largest eigenvalue. Note that it is far from obvious (a priori) that the cointegrating rank will be one. It might very well be zero, or, if there were a single stochastic trend linking the yields, then the cointegrating rank would be two.

Because the series don’t have a trend, the appropriate choice for the deterministic component is DET=RC, which doesn’t include a constant in the individual equations (where it would cause the series to drift because of the unit roots), but restricts it to the cointegrating vector. The components of the cointegrating vector produced by JohMLE with this choice will have four components: the three variables + the constant. If we had done the default DET=CONSTANT, it would only be a 3-vector. We would also include DET CONSTANT in the SYSTEM definition.

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

Set up the model with the error correction term

system(model=ectmodel)
variables ftbs3 ftb12 fcm7
lags 1 to 6
ect ecteq
end(system)

Estimate it and compute the decomposition of variance

estimate
compute sigma=%sigma
errors(model=ectmodel,steps=36)
```
10.9 VAR’s for Forecasting: The Bayesian Approach

Problems with Unrestricted Models

Forecasts made using unrestricted vector autoregressions often suffer from the overparameterization of the models. The number of observations typically available is inadequate for estimating with precision the coefficients in the VAR. This overparameterization causes large out-of-sample forecast errors. For instance, see Fair (1979) for a comparison of forecasting performance of several models that includes an unrestricted VAR.

One possible way to handle this is to use some criterion to choose “optimal” lag lengths in the model. Examples are the Akaike or Schwarz Criterion. Example 10.1 shows how these can be used for choosing an overall lag length. However, there are $N^2$ lag lengths which need to be chosen in a full model, which makes a complete search using such a criterion effectively impossible.

No method of applying direct restrictions (like PDL’s for distributed lags) seems reasonable. Vector autoregressions are, in effect, dynamic reduced forms, not structural relations, so the meaning of the values for coefficients is not obvious, and no “shape” constraint suggests itself.

A Bayesian Approach

The Bayesian approach to this problem is to specify “fuzzy” restrictions on the coefficients, rather than “hard” shape or exclusion restrictions. Shrinkage estimators such as this have long been suggested for dealing with multicollinearity and similar problems.

In his dissertation, Litterman suggests the following: the usual approach to degrees of freedom problems is to reduce the number of regressors, which in autoregressive models means using fewer lags. Dropping a lag forces its coefficient to zero. Rather than adopting an all-or-nothing approach, we “suggest” that coefficients on longer lags are more likely to be close to zero than shorter lags. However, we permit the data to override our suggestion if the evidence about a coefficient is strong. Formally, we implement this by placing on the long lags Normal prior distributions with means of zero and small standard deviations. This allows us to estimate the coefficients using Theil’s mixed estimation technique.

In a vector autoregression, we must concern ourselves not only with lags of the dependent variable, but also with lags of the other endogenous variables. Because of stability conditions, we have some pretty good information about the size of lag coefficients in a simple autoregression. However, it’s not as clear what the sizes of coefficients on other variables should be, and these depend, in part, on the relative scales of the variables involved.

Specification of a complete Normal prior on a VAR would be intractable because the covariance matrix of the prior would have dimensions $N^2 L \times N^2 L$. Instead, we have
put together a general form for the prior involving a few hyperparameters and, in effect, you estimate your model by choosing those.

You add the prior to your VAR by including the instruction `SPECIFY` in your system definition:

```plaintext
system(model=canusa)
variables usaml usatbill canml cantbill canusxr
lags 1 to 13
det constant
specify(type=symmetric,tightness=.15) .5
end(system)
```

### Standard Priors

In the discussion in this section, variable $j$ refers to the $j$th variable listed on the `VARIABLES` instruction and equation $i$ to the equation whose dependent variable is variable $i$.

The standard priors have the following characteristics:

- The priors put on the deterministic variables in each equation are non-informative (flat).
- The prior distributions on the lags of the endogenous variables are independent Normal.
- The means of the prior distributions for all coefficients are zero. The only exception is the first lag of the dependent variable in each equation, which has a prior mean of one, by default.

Because of these restrictions, the only information required to construct the prior is:

- the mean of the prior distribution for the first own lag in each equation.
- the standard deviation of the prior distribution for lag $l$ of variable $j$ in equation $i$ for all $i, j$ and $l$: denoted $S(i, j, l)$.

To simplify further the task of specifying the prior, the standard priors restrict the standard deviation function to the form:

$$
(1) \quad S(i, j, l) = \left\{ \gamma g(l) f(i, j) \right\} s_i \frac{s_j}{s_j} ; \quad f(i, i) = g(l) = 1.0
$$

where $s_j$ is the standard error of a univariate autoregression on equation $i$. We scale by the standard errors to correct for the different magnitudes of the variables in the system.

The part in braces is the tightness of the prior on coefficient $i, j, l$. It is the product of three elements:
Chapter 10: Vector Autoregressions

1. The overall tightness \([\gamma]\), which, because of the restrictions on the \(f\) and \(g\) functions, is the standard deviation on the first own lag.

2. The tightness on lag \(l\) relative to lag 1 \([g(l)]\).

3. The tightness on variable \(j\) in equation \(i\) relative to variable \(i\) \([f(i, j)]\).

The Prior Mean

By default, we use a mean of zero for the prior on all coefficients except the first own lag in each equation. The prior on this coefficient is given a mean of one. This centers the prior about the random walk process

\[
Y_t = Y_{t-1} + u_t
\]

This seems to be a reasonable choice for many economic time series. For example, there are theoretical reasons for various asset prices to follow random walks, and a series growing exponentially at rate \(\alpha\) can be represented by

\[
\log Y_t = \log Y_{t-1} + \alpha
\]

so its log follows a random walk with drift.

The only alternative which strongly suggests itself is a mean of zero on series which are likely to be close to white noise, such as a series of stock returns.

The MEAN and MVECTOR options of SPECIFY control the first own lag means. MVECTOR is more likely to be useful since it permits means to vary across equations:

\[
specify(mvector=||1.0,0.0,1.0,1.0,1.0||,type=symmetric) .5
\]

The MVECTOR option here puts a mean of 0.0 on the second equation and 1.0 on all the rest.

Lag Length and Lag Decay

Experience with VAR models has shown that it usually is better to include extra lags with a decaying lag prior rather than to truncate at an early lag and use the default prior. Unless you face severe data constraints, we would suggest using at least one year + one period of lags. Longer lags without any decay are a bad idea.

To tighten up the prior with increasing lags, use TYPE=HARMONIC or TYPE=GEOMETRIC with an appropriate value for DECAY. A HARMONIC with \(\text{DECAY}=1.0\) or \(\text{DECAY}=2.0\) commonly works well. GEOMETRIC tends to get too tight too fast.

Highly seasonal data are difficult to work with using these standard priors because you typically expect relatively large coefficients to appear on the lags around the seasonal. See Section 10.12 for some tips on dealing with seasonal data.
Chapter 10: Vector Autoregressions

Overall Tightness

The default is the relatively “loose” value of 0.20. Practice has shown that a reasonable procedure is to set the TIGHTNESS parameter to something on the order of .1 or .2; then, if necessary, you can tighten up the prior by giving less weight to the other variables in the system. Since TIGHTNESS controls directly the important own lags, setting it too small will force the own lags too close to the prior mean.

Prior Type

The prior type determines the relative tightness function \( f(i, j) \): the tightness on variable \( j \) in equation \( i \) relative to that on variable \( i \). The two types of priors (selected using the TYPE option) are the SYMMETRIC and the GENERAL.

TYPE=SYMMETRIC is the simplest prior and is the default. There is only one free hyperparameter; indicated with the other’s weight parameter on SPECIFY, it gives the relative weight \( (w) \) applied to all the off-diagonal variables in the system:

\[
(4) \quad f(i, j) = \begin{cases} 
1.0 & \text{if } i = j \\
\frac{1}{w} & \text{otherwise}
\end{cases}
\]

The symmetric priors generally are adequate for small systems: those with five or fewer equations. A combination of TIGHTNESS=.20 and \( w=0.5 \) is a common choice. As you push \( w \) to zero, the system approaches a set of univariate autoregressions—coefficients on all variables other than the own lags of the dependent variable and the deterministic part are forced to zero. The following is an example of SYMMETRIC:

```plaintext
system(model=smallmod)  
variables gnp m1 ipd tbill unemp bfi 
lags 1 to 4  
det constant  
specify(type=symmetric,tight=.2) .5  
end(system)
```

TYPE=GENERAL requires that you specify the entire \( f(i, j) \) function. Obviously, it is unrealistic to think of fine-tuning the prior by picking all of these independently. In fact, such a strategy simply transfers the problem with overparameterization from estimating too many equation coefficients to estimating too many hyperparameters. Instead, you should use a GENERAL prior in situations where

- the model is too large to apply safely a SYMMETRIC prior. The TIGHTNESS=.2 and \( w=.5 \) recommended above tends to be too loose overall for a system with six or more equations. However, making \( w \) much smaller will cut out too much interaction. Use a GENERAL which puts moderate weight on variables which you see as being important and low weight on those you believe to be less important.
the results of a `SYMmetric` show that you need to treat some equations more as univariate autoregressions than as part of the VAR. Use a `GENERAL` which is largely the same as the `SYMmetric`, but has small off-diagonal elements in the rows corresponding to these equations.

The \( f(i, j) \) function is input in one of two ways: by a `RECTangular` array (using the `MATRIX` option) or by supplementary cards.

```plaintext
system(model=ymrp)
variables gnp ml cpr ppi
lags 1 to 12
det constant
specify(type=general,tight=.2,decay=1.0)
# 1.0 0.5 0.5 0.5 0.1 1.0 1.0 0.1 0.1 1.0 1.0 0.1 $
 0.5 0.5 0.5 1.0
end(system)
```

or

```plaintext
declare rect priormat(4,4)
input priormat
  1.0 0.5 0.5 0.5
  0.1 1.0 1.0 0.1
  0.1 1.0 1.0 0.1
  0.5 0.5 0.5 1.0
specify(type=general,matrix=priormat,tight=.2,decay=1.0)
```

**Estimation Methods**

The VAR with a prior is estimated using the `ESTIMATE` instruction. This employs a variation on the mixed estimation procedure described in Section 5.12. It should be noted that, with this type of a prior, single equation techniques are not optimal except in the unlikely case that the residuals are uncorrelated. In the early 1980’s, when Bayesian VAR’s were introduced, system-wide techniques weren’t feasible for any but the smallest models. And it’s still the case that full system estimators, done properly (see, for instance, the `GIBBSVAR. PRG` program) can be done only for medium sized models, because of the size of the matrices which must be inverted.

The simpler calculations done by `ESTIMATE` (and `KALMAN`, Section 10.13) still are quite valuable. The gains from more careful estimation are likely to be small, since it is only the combination of a prior and a non-diagonal covariance matrix that produces any gain at all. Our suggestion would be that you develop the model using the basic techniques and switch to the more computationally intensive methods only once the model has been built.
10.10 Differences When Using a Prior

Differences with END(SYSTEM)

When you use a SPECIFY in setting up the system, the END(SYSTEM) instruction causes RATS to print a synopsis of the prior. For example:

```
Summary of the Prior...
Tightness Parameter 0.100000
Harmonic Lag Decay with Parameter 0.000000
Standard Deviations as Fraction of Tightness and Prior Means
Listed Under the Dependent Variable

<table>
<thead>
<tr>
<th></th>
<th>LOGCANGDP</th>
<th>LOGCANDEFL</th>
<th>LOGCANM1</th>
<th>LOGEXRATE</th>
<th>CAN3MTHPCP</th>
<th>LOGUSAGDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGCANGDP</td>
<td>1.00</td>
<td>0.50</td>
<td>0.01</td>
<td>0.01</td>
<td>0.20</td>
<td>0.50</td>
</tr>
<tr>
<td>LOGCANDEFL</td>
<td>0.50</td>
<td>1.00</td>
<td>0.01</td>
<td>0.01</td>
<td>0.20</td>
<td>0.50</td>
</tr>
<tr>
<td>LOGCANM1</td>
<td>0.50</td>
<td>0.50</td>
<td>1.00</td>
<td>0.01</td>
<td>0.20</td>
<td>0.50</td>
</tr>
<tr>
<td>LOGEXRATE</td>
<td>0.50</td>
<td>0.50</td>
<td>0.01</td>
<td>1.00</td>
<td>2.00</td>
<td>0.50</td>
</tr>
<tr>
<td>CAN3MTHPCP</td>
<td>0.50</td>
<td>0.50</td>
<td>0.01</td>
<td>0.01</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>LOGUSAGDP</td>
<td>0.50</td>
<td>0.50</td>
<td>0.01</td>
<td>0.01</td>
<td>0.20</td>
<td>1.00</td>
</tr>
<tr>
<td>Mean</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
```

Variables Defined by SPECIFY

%PRIOR  SPECIFY stores the matrix of weights and means in %PRIOR, an (N+1)xN array, arranged as printed above. By making changes to %PRIOR, you can alter a standard prior without going through a complete redefinition of the SYSTEM.

Differences with ESTIMATE

RATS estimates the system of equations using the mixed estimation technique described in Theil (1971). The differences between ESTIMATE for systems with a prior and for systems without a prior are:

- The degrees of freedom reported are not $T-K$, where $K$ is the number of regressors, but $T-D$, where $D$ is the number of deterministic variables. This is a somewhat artificial way to get around the problem that, with a prior, $K$ can exceed $T$.
- With the option CMOM=SYMMETRIC array for $X’X$, you can obtain the array $X’X$ of the regressors.
- With the option DUMMY=RECTANGULAR array of dummy observations, RATS saves the dummy observations used in doing the mixed estimation procedure in the specified array.
10.11 Choosing a Prior

Influence of Model Size

The larger is a model relative to the number of data points, the more important the prior becomes, as the data evidence on the individual coefficients becomes weaker. “Average” situations are models with 9 parameters per equation for 40 data points, 30 for 120, and 70 for 400. These models call for moderately tight priors. Substantially larger models for a given size require greater tightness through either:

- a lower value for TIGHTNESS, or
- the downweighting of the “other” variables, either through a tighter SYMMETRIC prior or through use of a GENERAL prior.

Objective Function

In searching over the parameters governing the prior, we need to have, formally or informally, an objective function. Because we are generating a forecasting model, the best forms for this are based upon forecast errors. Three have been used:

- Theil $U$ values (computed with the instruction THEIL), which can be used formally, by mapping them to a single value by a weighted average, or informally, by examining changing patterns in the values.
- likelihood function of the data conditional on the hyperparameters. This is a by-product of the Kalman filter procedure.
- log determinant of the covariance matrix of out-of-sample forecast errors.

The last two are discussed in Doan, Litterman and Sims. The third was used in that paper and is the most difficult to compute. Our preference is the informal use of Theil $U$ statistics.

In all cases, we calculate simulated “out-of-sample” forecasts within the data range. We do this by using the Kalman filter (Section 10.13) to estimate the model using only the data up to the starting period of each set of forecasts.

A Simple Procedure

A simple procedure which we have found to be effective is the following:

1. Run a system of univariate OLS models to get benchmark Theil U’s.
2. Run a system of univariate models with a standard value for TIGHTNESS.
3. Run a standard SYMMETRIC prior.

Based upon these, adjust the prior (switching to a GENERAL prior):

- If the Theil U’s in an equation are worse in 2 than in 1, loosen up on the own lags by setting the diagonal element to 1.5 or 2.0.
- If the Theil U’s in an equation are worse in 3 than in 2, tighten up on the other variables by reducing the off-diagonal elements.
Example 10.7 Using RUNTHEIL to pick a model

This example shows how to use the procedure RUNTHEIL (on the example file CANMODEL.PRG) to compute Theil U statistics for various priors. You need to change the italicized lines to accommodate your data set.

By setting Tightness to a large value (such as 2.00), you effectively eliminate the “Bayesian” part, and thus look at OLS. By setting Other to a very small value (such as .001), you effectively eliminate the “Vector” part, leaving a univariate model.

```plaintext
procedure runtheil
option choice type 1 symmetric general
option rect matrix
option vector mvector
option real tightness .1
option real other .5
option string window
*
local integer time
*
system(model=canmodel)
variables logcangdp logcandefl logcanm1 logexrate $
     can3mthpcp logusagdp
lags 1 to 4
det constant
specify(tight=tightness,type=type,matr=matrix,mvect=mvector) other
end(system)
*
theil(model=canmodel,setup,steps=12,to=2006:4)
estimate(noprint) * 1998:4
theil
do time=1999:1,2006:3
    kalman
    theil
end do time
theil(dump,window=window)
end
***********************************************************************
@runtheil(tightness=2.0,other=.001,window="Univariate OLS")
@runtheil(tightness=0.1,other=.001,window="Univariate BVAR")
@runtheil(tightness=0.1,other=0.5,window="Simple BVAR")
@runtheil(tightness=2.0,other=1.0,window="OLS VAR")
*
@runtheil(tight=.10,type=general,window="General")
# 1.00 0.50 0.50 0.50 0.50 0.50 $
 0.50 1.00 0.50 0.50 0.50 0.50 $
 0.01 0.01 1.00 0.01 0.01 0.01 $
 0.01 0.01 0.01 1.00 0.01 0.01 $
 0.20 0.20 0.20 2.00 0.20 0.20 $
 0.50 0.50 0.50 0.50 0.50 1.00
```

---

Chapter 10: Vector Autoregressions
Chapter 10: Vector Autoregressions

10.12 Selecting a Model

An Illustration

The selection of a Vector Autoregressive model for forecasting is more difficult than selection of an ARIMA or other univariate time series model. You have to make the following choices:

- which variables to include in the model
- lag length
- prior structure and hyperparameters

To look at a simple illustration, suppose that we are interested in the forecasts of the series, \textit{SALES}. Previous tests have shown that \textit{SALES} is closely related to national economic conditions. A first try at a model might be a two variable system with \textit{SALES} and \textit{GDP}. However, the model uses a one-step forecast of \textit{GDP} to compute the two-step forecast of \textit{SALES}. If the initial forecast of \textit{GDP} is poor, then the forecasts of \textit{SALES} derived from it are also likely to be unsatisfactory.

As a second attempt, you consider the addition of variables to improve the \textit{GDP} forecasts. However, a problem arises as you add more variables: it seems that there are always still more variables which you could add to improve the forecasts of the existing variables. Obviously, you cannot include in the system every variable which \textit{might} have an effect on \textit{SALES} (through some route). At some point, you must decide which variables to include and which to exclude.

Choosing a Set of Variables

Although you can incorporate quite a few variables in a system through an appropriate choice of prior, it is still a good idea to restrict yourself to relatively small systems (3 to 5 variables) if you want to choose a model to forecast a single series. You should think of a list of candidate series. Usually, there are some obvious choices, such as \textit{GDP} in the example above.

The instruction \texttt{ERRORS} can be very helpful in refining the forecasting model, especially when you have no strong prior information about which series will be important for the prediction of variables in question. There are two pieces of information it provides which are valuable:

The standard errors of forecast

You can use these for a quick comparison of the forecasting abilities of the VAR’s from several sets of possible variables. Check the computed standard errors for the variable(s) of greatest interest. The set that produces the lowest values should be regarded as the most promising.

The decomposition of variance

This can indicate which variables you might replace to lower the errors. A variable which explains a very low fraction of the target variable is a good
candidate for replacement. If you use the decomposition in this manner, remember to consider the possible effects of changes in the ordering.

**Using Partial VAR’s**

Often it is unrealistic to cast the forecasting model in the form of a full VAR. For instance, it is probably reasonable to assume that SALES (of a single company) has no real explanatory power for economy-wide variables, so a GDP equation could omit SALES entirely.

If we return to the situation on the previous page, a possible model would consist of

1. a single equation that explains SALES by lagged SALES, lagged GDP, and perhaps one or two other variables, and
2. a separate VAR system that forecasts GDP and the other variables.

We can combine the SALES equation with the other equations to form the forecasting model. This has several advantages over a VAR system which includes SALES:

- We can estimate the VAR subsystem using all the data available for its variables. This may be quite a bit longer than the data record for SALES.
- We can give special attention to the single equation for SALES, particularly if there is a seasonality problem (see below).

However, it is not possible to put a prior on the coefficients of the SALES equation using **SPECIFY**. Instead, you can use the procedure **MIXVAR** (provided with RATS on the file **MIXVAR.SRC**) to estimate this equation separately, then combine it with the VAR for the national variables. **MIXVAR**, in effect, estimates a single equation using a prior of the same type used in full VAR’s.

```plaintext
source mixvar.src
*
system(model=national)
variables orddurgd ipelect prime
lags 1 to 4
det constant
specify(type=symmetric) 1.0
end(system)
estimate
*
@mixvar(define=saleseq,numlags=2) sales
# orddurgd ipelect prime
forecast(model=national+saleseq,results=forecasts,steps=6,from=2007:3)
```
Dealing with Seasonality

Handling variables that exhibit strong seasonality can be somewhat tricky. There are several methods available, but none of them is guaranteed to work because seasonal effects can have very different forms.

- You can include seasonal dummies in the system using the `DETERM` instruction.
- You can model the VAR using seasonal differences (computed using `DIFFERENCE`), and use `MODIFY` and `VREPLACE` to rewrite the model prior to forecasting.

Including a long lag length allows the estimated coefficients to pick up the seasonal effect. However, you cannot use the `DECAY` factor with this method, since it will dampen the values of the important seasonal lags.

There are some types of non-diagonal priors (ones which don’t put independent distributions on the lags) which might help with this. These would mainly be implemented using “dummy observations”. See Section 10.13.4.
10.13 Using the Kalman Filter

Background

The Kalman filter is a fast, recursive algorithm for estimating and evaluating dynamic linear models. RATS has three instructions or sets of instructions which use it. **DLM** (Dynamic Linear Models), described in Section 12.2, is used for state-space models. These are usually small dynamic models which relate observed data to unobservable “states.” **RLS** (Section 5.7) uses the Kalman filter for Recursive Least Squares for a single linear model. This is generally used for examining the stability of a linear relationship.

For VAR’s, the “states” in the Kalman filter are the coefficients. The instruction **KALMAN** produces sequential estimates of these. **KALMAN** can be used to update coefficient estimates as the data set expands, or to estimate a model where coefficients are allowed to vary with time.

**KALMAN** works just one observation at a time, so it will generally be used in a loop. Because of the “step-at-a-time” design, you can do sequential calculations based upon each set of coefficients, such as producing forecasts or adjusting the model.

Assumptions

As mentioned above, the Kalman filter, as used by RATS for VAR’s, is a restricted form of the more general Kalman filter. It is specifically designed for estimating coefficients of a linear regression model, allowing for limited types of time-variation.

The following is the model we use. $\mathbf{\beta}_t$ is the vector of coefficients at time $t$:

- The measurement equation is $y_t = X_t \mathbf{\beta}_t + u_t$, 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 $\text{var}(\mathbf{v}_t) = \mathbf{M}_t$.
- $u_t$ and $\mathbf{v}_t$ are independent.
- $n_t$ and $\mathbf{M}_t$ are assumed to be known.

If $\mathbf{M}_t$ is equal to zero, then there is no time-variation.

Updating Formula

If we have an estimate of $\mathbf{\beta}_{t-1}$ using information through $t-1$ (denoted $\mathbf{\beta}_{t-1|t-1}$) and its covariance matrix $\Sigma_{t-1}$, then the updated estimate given $y_t$ and $X_t$ is

1. $\Sigma_{t|t-1} = \Sigma_{t-1} + \mathbf{M}_t$

2. $\Sigma_t = \Sigma_{t|t-1} - \Sigma_{t|t-1} X_t (X_t \Sigma_{t|t-1} X'_t + n_t)^{-1} X'_t \Sigma_{t|t-1}$

3. $\mathbf{\beta}_{t|t} = \mathbf{\beta}_{t-1|t-1} + \Sigma_{t|t-1} X_t (X_t \Sigma_{t|t-1} X'_t + n_t)^{-1} (y_t - X_t \mathbf{\beta}_{t-1|t-1})$
Initial Values
To use this algorithm, we need to supply the following information:

- $\beta_{0|0}$, the initial state vector
- $\Sigma_0$, the initial covariance matrix of the states
- $n_t$, the variance of the measurement equation
- $M_t$, the variance of the change in the state vector

10.13.1 Sequential Estimation
In the most common use of the Kalman filter in RATS:

- $M_t$ is 0
- $n_t$ is assumed to be the constant $\sigma^2$
- $\Sigma_0$ and $\beta_{0|0}$ are obtained using `ESTIMATE` through part of the sample.

This is a “fixed coefficients” setup. The true coefficient vector is assumed to stay the same throughout the sample, and the Kalman filter is used to estimate it using samples of increasing size.

If we look at the formulas on the last page, note that if we multiply $M_t$, $\Sigma_t$ and $n_t$ by the same constant, then that constant will drop out of the updating formula for $\beta$. We can take advantage of this by setting $n_t$ to 1. We will then have a $\Sigma$ that must be multiplied by $\sigma^2$ (which can be estimated separately using the residuals) to get the true covariance matrix. This makes $\Sigma$ analogous to $(X'X)^{-1}$ in a least-squares regression. `ESTIMATE` will initialize the Kalman filter and each `KALMAN` instruction will add one more observation to the sample.

This segment of code estimates the model through 1998:4, and does updates over the period from 1999:1 to 2006:3. Forecast performance statistics are compiled over that period using the instruction `THEIL`. Note that the estimates for a VAR without a prior will be the same whether you estimate through the whole period, or first estimate through a subperiod with the Kalman filter being used for the remainder. However, they won’t match when the system has a prior. This is because the prior is rescaled using statistics computed using the sample on the `ESTIMATE`.

```r
system(model=canmodel)
variables logcangdp logcandefl logcanm1 logexrate can3mthpcp logusagdp
lags 1 to 4
det constant
specify(tightness=.1) .5
end(system)
*
theil(model=canmodel,setup,steps=12,to=2006:4)
```
Recursive Residuals

The KALMAN instruction can also be used to compute recursive residuals, either for a VAR, or for a general linear regression. However, you will probably find the RLS instruction to be much more convenient for single equation models.

To compute recursive residuals with KALMAN, you first estimate the model over the first $K$ observations ($K = \text{number of regressors}$), and then use the Kalman filter to run through the remainder of the sample. The $T-K$ recursive residuals have the convenient property that, if the model is the Standard Normal Linear Model, these residuals are independent Normal. They are the series of normalized one-step Kalman Filter forecast errors:

\[
(4) \quad \frac{(y_t - X_t\beta_{t-1})}{\sqrt{n_t + X_t(\Sigma_{t-1} + M_t)X_t'}}
\]

The basic setup for a VAR model would be as follows:

\begin{verbatim}
compute nreg = number of regressors per equation
estimate(noprint) start     start+nreg-1

do time=start+nreg,end
    kalman(rtype=recursive,resids=recresids)
end do time
\end{verbatim}
10.13.2 Time-Varying Coefficients

Background

If you permit time-variation, you can’t initialize the Kalman filter using \texttt{ESTIMATE}. Instead, you need to start by setting presample values for $\beta$ and $\Sigma$, and then filter through the entire sample.

For very small models, you can set these as free parameters and optimize using \texttt{FIND}. However, this very quickly becomes infeasible. To run the simplest $K$ variable time-varying parameters estimation, you need to set (or estimate) $K$ (initial $\beta$) + $K(K+1)/2$ (initial $\Sigma$) + $K(K+1)/2$ (M matrix) + 1 (measurement equation variance) total parameters. With a mere 10 regressors, this is well over 100.

Fortunately, the initial values for $\beta$ and $\Sigma$ serve the same purpose as the mean and variance of a Bayesian prior. This suggests that, at least for VARs, we can just start with a standard prior.

We will be looking at estimating a single equation out of a two-variable VAR. See the example file \texttt{TVARYING.PRG} for a working example of the instructions in the next two sections. See also the procedure \texttt{TVARSET} which sets up an entire VAR using a structure similar to the one we describe here.

Initializing Coefficients

Use the instruction \texttt{ASSOCIATE} to initialize $\beta$ for an equation.

\begin{verbatim}
equation kalmeq gdq
  # constant gdq{1 to 4} fm1a{1 to 4}
declare vector b(9)
ewise b(i)=(i==2)
associate kalmeq b
\end{verbatim}

This sets a prior mean of 1 for $\text{GDPQ}\{1\}$ (the second regressor) and 0 for all other coefficients.

Initializing Variances

You use the instruction \texttt{KFSET} to supply $\Sigma_0$ and $n_t$. \texttt{KFSET} sets $n_t$ directly using one of the options \texttt{SCALE}, \texttt{CONSTANT} or \texttt{VECTOR}. List the $\Sigma$ matrices as parameters on \texttt{KFSET}. You need to set their initial values at some point before you execute a \texttt{KALMAN}.

Most priors for VAR’s have diagonal $\Sigma_0$ matrices, which cuts down dramatically on the number of “hyperparameters” which govern their values. The main point of the prior is to “tame” the coefficient estimates, as the severe collinearity among the regressors tends to make the data very weak at determining the precise contributions of the different lags.
Chapter 10: Vector Autoregressions

In Doan, Litterman and Sims (1984), we used a more complex variant of the following (continuing from above):

```
linreg gdpq
# constant gdpq{1 to 4}
compute gseesq=.9*%seesq
linreg fm1a
# constant fm1a{1 to 4}
compute mseesq=.9*%seesq
```

Set up the system. Use GSEESQ as the measurement equation variance.

```
system kalmeq
kfset(constant,noscale,likelihood=likely) xxx
# gseesq
tvarying tvx
end(system)
```

This uses a HARMONIC decay with DECAY=.5

The hyperparameters (PIx) below are the following:
- PI5 = Overall tightness. This corresponds to TIGHTNESS^2 on SPECIFY.
- PI2 = Relative tightness on other variables. This corresponds to w^2 on the symmetric prior.
- PI3 = Relative tightness on the constant. (No correspondence).
- PI7 = Relative tightness on time variation.

```
compute pi5 = .20**2 , pi2 = .5**2 , pi3 = 10000.0 , pi7=1.e-8
dim xxx(9,9) tvx(9,9)
compute xxx=%const(0.0)
compute xxx(1,1)=pi5*pi3*gseesq
do i=1,4
    compute decayfac=i**(2*.5)
    compute xxx(i+1,i+1)=pi5*decayfac
    compute xxx(i+5,i+5)=pi5*pi2*decayfac*gseesq/mseesq
end do i
compute tvx=pi7*xxx
```

Executing the Filter

Once you have gone through all the initialization steps, you are ready to begin Kalman filtering. The first step must be done using KALMAN with the STARTUP option. Remaining steps use just KALMAN.

```
do time=1960:1,1998:4
    if time==1960:1
        kalman(startup=time)
    else
        kalman
    end do time
```
Likelihood Function

For the model with random walk coefficient variation, conditional upon previous information, \( y_t \) is distributed Normally with

Mean \( X_t \beta_{t-\|t-1} \)

Variance \( \sigma_t^2 = n_t + X_t(\Sigma_{t-1} + M_t)X_t' \)

Ignoring constants, (minus) the log of the likelihood element for this observation is

\[
(5) \quad \log \sigma_t^2 + \frac{(y_t - X_t \beta_{t-\|t-1})^2}{\sigma_t^2}
\]

You can construct the sample likelihood by using the `LIKELIHOOD` option matrix on `KFSET`. This has dimensions 2 x (number of equations). The first element in each column is the cumulation of the first terms for an equation and the second element is the cumulation of the second terms.

As mentioned in Doan, Litterman and Sims (1984, p. 10), the filtering procedure above will produce the same series of coefficient estimates if \( n_t, \Sigma_0 \) and \( M_t \) are all multiplied by a constant. Concentrating out this “nuisance parameter” gives a pseudo-likelihood function for equation \( n \) which (disregarding constants) is

\[
-.5*(\text{LIKELY}(1,n)+\text{nobs}^*\text{LOG(LIKELY}(2,n)/\text{nobs}))
\]

where `LIKELY` is the `LIKELIHOOD` option matrix and `NOBS` is the number of observations.

While we could rely upon this by setting \( n_t = 1 \), when the constant multiplier for the variances just becomes \( \sigma^2 \) (which can be estimated as `LIKELY(2,n)/\text{nobs}`), we find that it’s simpler in the case of VAR’s to pick a reasonable value for \( n_t \) based upon the estimated variance from a preliminary regression. This is because we set \( \Sigma_0 \) directly based upon the standard VAR prior. The prior on the own lags is independent of the scale of the variable involved, and the priors on the lags of other variables are scaled up to take account of the scales. The nuisance parameter in the likelihood then becomes a correction factor to the variances.
10.13.3 More General Time-Variation
If we relax the random walk assumption for time-variation to allow the state vector to follow the process
\[ \beta_t = \mathbf{A}_t \beta_{t-1} + \mathbf{v}, \text{ with } \text{var}(\mathbf{v}) = \mathbf{M}_t \]
then the KF updating formulas change to
\[ S_t = A_t \Sigma_{t-1} A_t' + \mathbf{M}_t \]
\[ \Sigma_t = S_t - S_t X'_t (X_t S_t X'_t + n_t)^{-1} X_t S_t \]
\[ \beta_{t|t} = A_t \beta_{t-1|t-1} + S_t X'_t (X_t S_t X'_t + n_t)^{-1} (y_t - X_t A_t \beta_{t-1|t-1}) \]
The only difference between these formulas and those from before ((1) and (3)) is the addition of \( A_t \) to (7) and (9). Since the \( A \)'s only apply to \( \Sigma_{t-1} \) and \( \beta_{t-1|t-1} \), you can implement this in RATS by using \texttt{COMPUTE} to replace
\[ \cdot \Sigma_{t-1} \text{ by } A_t \Sigma_{t-1} A_t' \]
\[ \cdot \beta_{t-1|t-1} \text{ by } A_t \beta_{t-1|t-1} \]
This step needs to be done before each \texttt{KALMAN} instruction. To manipulate the coefficient vector, we set up the space for it ourselves using \texttt{ASSOCIATE(PERM)}.

Example
If we assume that the coefficient vector “shrinks” back towards the mean, following the process:
\[ \beta_t = \pi_8 \beta_{t-1} + (1 - \pi_8)(\text{mean vector}) + \mathbf{v}_t \]
we need to make the following alterations to the example in the preceding section:
\begin{verbatim}
dec vector b(9) bmean(9)
ewise b(i)=(i==2)
compute bmean=b
associate(perm) kalmeq b

and

do time=1960:1,1998:4
  if time==1960:1
    kalman(startup=time)
  else {
    compute b=pi8*b+(1-pi8)*bmean
    compute xxx=(pi8**2)*xxx
    kalman
  }
end do time
\end{verbatim}
This is a particularly simple example because $A_t$ is just $\pi_8$ times the identity.

This example uses the FIND command to estimate two of the “hyperparameters” used in the example TVARYING.PRG, from Section 10.13.2. Note the following:

- The IF inside the FIND loop is how you reject illegal values. Just set the value to %NA and skip the remaining calculation.
- We eliminated the PRINT on KALMAN since that would give us regression output for each test setting of the hyperparameters.

```plaintext
compute pi5 = .20**2, pi2 = .5**2, pi3 = 10000.
compute pi7 = 0.00000001, pi8 = 1.00
nonlin pi8 pi7
dec real loglikely
dim xxx(9,9) tvx(9,9)
fine(find) maximum loglikely
  if pi7<0.or.pi8>1.00 {
    compute loglikely=%na
    next
  }
  compute likely=%const(0.0)
  compute b=bmean
  compute xxx=%const(0.0)
  compute xxx(1,1)=pi5*pi3*gseesq
  do i=1,4
    compute decayfac=i**(2*.5)
    compute xxx(i+1,i+1)=pi5*decayfac
    compute xxx(i+5,i+5)=pi5*pi2*decayfac*gseesq/mseesq
  end do i
  compute tvx=pi7*xxx
  *
  do time=1960:1,1985:4
    if time==1960:1
      kalman(start=time)
    else {
      compute b=pi8*b+(1-pi8)*bmean
      compute xxx=(pi8**2)*xxx
      kalman
    }
  end do time
  compute nobs=1985:4-1960:1+1
  compute loglikely=-.5*(likely(1,1)+nobs*log(likely(2,1)/nobs))
end fine
```
10.13.4 Dummy Observation Priors

The priors handled using SPECIFY have the common property that the prior distributions are independent across coefficients. There are several ways of handling more general types of priors for VARs. You can apply single equation techniques with mixed estimation (Section 5.12). You can do system wide estimation using the ideas in the GIBBSVAR.PRG program.

If, however, you want to combine a non-diagonal prior with time variation, you need to be able to code the prior information up into the initial mean and covariance matrix used by KALMAN. While it’s possible to do this with matrix calculations, it will often be simpler to use KALMAN to do this. It has a special set of options for implementing “dummy observation” priors. These are priors that can written in the form:

\[ y_t = X_t \beta + v_t \]  

The options used on KALMAN to adjust the current coefficients and covariance matrix are:

- \( x = \) VECTOR of explanatory variables for dummy observation
- \( y = \) VECTOR of dependent variables for dummy observation
- \( v = \) VECTOR of equation variances

When KALMAN is applied with these, it doesn’t alter the likelihood matrix (10.13.2) or change the entry pointer.

Note that you can’t apply this until after you’ve initialized the coefficients and covariance matrices for the VAR, generally with some form of diagonal prior. If you provide an \( x \) option, but not a \( y \) option, it is assumed that the \( Y \) values are the ones predicted by current \( \beta \) for the input values of \( X \). Because the filter “error” is zero, there will be no change to the coefficients, but the covariance matrix of the coefficients will change. If you don’t provide a \( V \) option, the variance for the equations that were set up with KFSET are used.

For instance, the following is a dummy observation which says that the original prior coefficients are likely to be good predictors at the first data point (1949:3). This extracts into \( x_{dummy} \) the “X” vector of the VAR model at that data point. This is scaled by a relative value \( dumwght \). This, in effect, scales down the equation variances by \( dumwght \). If \( dumwght \) were one, the dummy observation would have the same level of information content as a standard observation. A higher value would mean it would be more precise than a typical observation.

```
compute xdummy = %eqnxvector(%modeleqn(sims9var,1),1949:3)*dumwght
kalman(x=xdummy)
```

See Sims (1993) for more on the use of dummy observation priors.
Chapter 10: Vector Autoregressions

10.14 Conditional Forecasts

Background

A conditional forecast is a forecast in which certain values during the forecast period are fixed in advance. The simplest type of conditional forecast is where time paths for the exogenous variables are input, as is usually done in simultaneous equations models.

When forecasts are conditional upon future values of endogenous variables, the procedure is more complicated. Consider the formula for the error in a $K$-step forecast (forecasting $t$ using $t-K$):

$$
\sum_{s=0}^{K-1} \Psi_s u_{t-s} = \Psi \sum_{s=0}^{K-1} s_s u_{t-s}
$$

When we constrain a future value of the process, we are setting a value for this forecast error, since subtracting the unconstrained forecast from the constrained value produces the error of forecast. We have thus put a linear constraint upon the innovations of the periods between $t$ and $t-K+1$.

With a one-step-ahead constraint, the forecast error in a variable consists of just one innovation since $\Psi_0 = I$ (non-orthogonal innovations). However, two steps out, the forecast error in a variable depends upon the first step innovations in all variables plus its own second step. There are now many ways to achieve the particular value. We have only a single constraint on a linear combination of $N+1$ variables.

Computing Conditional Forecasts

In calculating forecasts subject to constraints on the forecast error (1), it’s most convenient to switch to orthogonalized residuals, thus, the forecast error is

$$
\sum_{s=0}^{K-1} \Psi_s G v_{t-s} = \Psi \sum_{s=0}^{K-1} s_s G v_{t-s}
$$

where $G$ is a factor of the covariance matrix. By stacking the orthogonalized innovations during the forecast period into a vector, we can write the constraints in the form

$$
RV = r
$$

The conditional forecasts are then made by first computing the $V$ vector which minimizes $V'V$ subject to the constraint. The solution to this is

$$
V^* = R'(RR')^{-1}r
$$

The $V$’s are then translated into non-orthogonalized shocks, and the model is forecast with those added shocks.
Chapter 10: Vector Autoregressions

The CONDITION Procedure

This can all be done using the procedure CONDITION (on CONDITION.SRC). Before you execute it, you must define the VAR and estimate it. CONDITION can handle any number of constraints on single values of future endogenous variables. With the GENERAL option, you can use more general linear constraints on future values. Note that there is an older version of this procedure (called CNDITION) which does the same calculations, but uses parameters rather than options for most of the inputs.

@condition(options) nconstr
# series period value (one for each constraint)

or

@condition(general,options) nconstr
# vector with linear combination of endogenous variables
# entry value

nconstr Number of constraints

Supplementary Cards (without GENERAL, one for each constraint)
series Series constrained. This must be an endogenous variable of one of the equations.
period Period of the constraint.
value Target value for the constraint.

Supplementary Cards (with GENERAL, one pair for each constraint)
vector VECTOR or real numbers/expressions with linear combination of endogenous variables to constrain
period Period of the constraint.
value Target value for the constraint.

Options
model=model to forecast.
This should already be estimated.

from=starting period of forecasts [end of estimation + 1]
to=ending period of forecasts [last constrained period]
steps=number of forecast steps
STEPS can be used in place of the TO option if it’s more convenient.
The potential problem with the tighter constraints is that they might force some “whipsawing” innovations to hit all the values.
Example 10.8 Conditional Forecasting

This uses the same data set as Example 10.5, among others. This computes forecasts conditional on 5% year over year growth in the US GDP. This type of constraint is easy to handle because US GDP is in logs. The conditional and unconditional forecasts are then graphed for each series. The series \texttt{CONDFORE(1)},...,\texttt{CONDFORE(6)} have the conditional forecasts. The unconditional forecasts are computed and saved in \texttt{FORECAST(1)},...,\texttt{FORECAST(6)}.

It then does a set of 1000 simulations subject to the same constraint. The results of this for the Canadian GDP are analyzed on two graphs. Both of these show the .16 and .84 fractiles of the simulated values: these correspond roughly to one standard error. See Chapter 13 for more on the “bookkeeping” used to process the draws.

This is example \texttt{CONDITION.PRG}.

```plaintext
open data oecdsample.rat
calendar(q) 1981
data(format=rats) 1981:1 2006:4 can3mthpcp canexpgdpcchs $
  canexpgdpcs canmls canusxsr usaexpgdpc$ *
set logcangdp  = log(canexpgdpcchs)
set logcandefl = log(canexpgdpcs)
set logcanm1   = log(canmls)
set logusagdp  = log(usaexpgdpc)
set logexrate  = log(canusxsr)
*
* system(model=canmodel) 
variables logcangdp logcandefl logcanm1 $
  logexrate can3mthpcp logusagdp 
lags 1 to 4 
det constant 
specify(tightness=.15) 0.5 
end(system) 
estimate 
* 
compute fstart=2007:1,fend=2009:4 
* 
@condition(model=canmodel,steps=12,results=condfore) 2
# logusagdp 2007:4 logusagdp(2006:4)+.05
# logusagdp 2008:4 logusagdp(2006:4)+.10
forecast(model=canmodel,results=forecasts,from=fstart,to=fend) 
* 
do i=1,6
  compute [label] l=%modellabel(canmodel,i)
  graph(header="Forecasts of "+l,window=1,)$ 
    key=below,klabels=||"Unconditional","Conditional"||) 2
    # forecasts(i)
    # condfore(i)
end do i
```
This saves 1000 simulated values of Canadian Real GDP from the distribution constrained to 5% real growth of US Real GDP. It uses a series of vectors of dimension 1000; one for each time period during the forecast range.

```
dec series[vect] can

gset can fstart fend = %zeros(1000,1)
do draws=1,1000
   @condition(simulate,model=canmodel,steps=12,results=simfore) 2
   # logusagdp 2007:4 logusagdp(2006:4)+.05
   # logusagdp 2008:4 logusagdp(2006:4)+.10
   do t=fstart,fend
      compute can(t)(draws)=simfore(1)(t)
   end do t
end do draws
```

Compute the 84th and 16th percentiles of the distribution

```
set upper fstart fend = %fractiles(can(t),||.84||)(1)
set lower fstart fend = %fractiles(can(t),||.16||)(1)
```

```
graph(key=below,$
header="Canadian Real GDP with High Growth of US GDP",$
klabels=||"Unconditional","Conditional","84%ile","16%ile"||) 4
# forecasts(1)
# condfore(1)
# upper / 3
# lower / 3
```

Transform the GDP numbers into average (annualized) growth rates from FSTART-1.

```
set fgrowth fstart fend = $
   400.0*(forecasts(1)-logcangdp(fstart-1))/(t-fstart+1)
set cgrowth fstart fend = $
   400.0*(condfore(1)-logcangdp(fstart-1))/(t-fstart+1)
set ugrowth fstart fend = $
   400.0*(upper-logcangdp(fstart-1))/(t-fstart+1)
set lgrowth fstart fend = $
   400.0*(lower-logcangdp(fstart-1))/(t-fstart+1)
```

```
graph(key=below,$
header="Canadian GDP Average Growth with High Growth of US GDP",$
klabels=||"Unconditional","Conditional","84%ile","16%ile"||) 4
# fgrowth
# cgrowth
# ugrowth / 3
# lgrowth / 3
```
Chapter 11
Simultaneous Equations

We have not emphasized simultaneous equations in designing RATS over the years, leaving the field of the “big model” to specialized software. However, RATS can easily handle models of a more modest size, providing estimation and forecasting techniques.
11.1 Estimation

Preliminary Transformations

Most simultaneous equations models require a sizable set of preliminary transformations of the variables: differences, averages, logs, percentage changes, etc. When these involve lags, some data points are lost. Fortunately, you do not need to keep track of this information: RATS handles it by marking as missing any entry which it cannot compute. For instance, in this example, the three constructed variables are actually defined over different periods.

\[
\text{set addshort} = rs + rs(1) + rs(2) + rs(3) \\
\text{set diffyd} = yd - yd(1) \\
\text{set diffrate} = rl - rs
\]

Instrument Lists

The estimation instructions (LINREG, AR1, SUR, NLLS and NLSYSTEM) take their instruments from the list maintained by the instruction INSTRUMENTS. With a small model, you can just set the list once and estimate the model. With a larger model, the available instrument list may be too large to be used in its entirety, as it will exhaust your degrees of freedom. Thus, during estimation, you may need to change it from one equation to the next. A combination of INSTRUMENT instructions with the DROP and ADD options will let you make small changes:

\[
\begin{align*}
\text{instrument(drop)} & \quad rw \ rw(1) \\
\text{instrument(add)} & \quad xz \ zw
\end{align*}
\]

Drops RW and RW(1) from the list
Adds XZ and ZW to the list

Autocorrelated Errors

To correct for first-order autocorrelated errors, use AR1 with the INST option. If you choose METHOD=CORC, RATS uses Fair’s (1970) procedure. This requires, for consistency, that you include as instruments the lags of the variables involved in the regression. The alternative is METHOD=HILU, which does not have this requirement.

With the speed of current computers, there is no practical advantage to using the “Cochrane–Orcutt” style iterative procedures. Particularly in simultaneous equations, it is quite possible to have the function being optimized show multiple peaks when graphed against the autoregressive parameter. The search method of HILU will make it much more likely that you will find the best estimates. In fact, in the example model in this chapter, we found a difference between the two techniques, and altered the model as a result.

A common criticism of simultaneous equations models where many of the equations have to be estimated with autocorrelation corrections is that much of the explanatory power of the “model” actually comes not from the structural relationship, but from the autoregressive errors. If your original equations are showing very low Durbin-Watsons, it would probably be a good idea to rethink them as dynamic equations.
Chapter 11: Simultaneous Equations

Three Stage Least Squares

You can estimate the system by three stage least squares (3SLS) using either **SUR** or **NLSYSTEM** with the **INST** option.

To use **SUR**, you must do the following:

1. Define the equations in the system using a set of **EQUATION** instructions.
2. Set up the instruments list using **INSTRUMENTS**.
3. Estimate the model using **SUR(INST)**.

To use **NLSYSTEM**, you need to

1. Set the list of parameters to be estimated using the instruction **NONLIN**. If you might be doing a good deal of experimenting with your equations, it would be a good idea to set up a separate **PARMSET** for each equation, and then combine them when you estimate.
2. Create formulas for the equations with **FRML** instructions.
3. Set up the instruments list using **INSTRUMENTS**.
4. Estimate the model with **NLSYSTEM(INST)**.

**NLSYSTEM** and **SUR** should produce almost identical results for any linear model which has no restrictions on the parameters.

Limited Information Maximum Likelihood (LIML)

LIML actually was used before the now much more popular 2SLS as a single equation (limited information) estimator for simultaneous equation models. For a description of it, see, for instance, Greene (2003). While it has some theoretical advantages over 2SLS, it doesn’t appear to provide superior estimates. It can be done with RATS using the procedure **LIML** on **LIML.SRC**. Just as with **LINREG(INST)** for 2SLS, set the instrument list in advance using **INSTRUMENTS**. The syntax is then familiar:

```plaintext
@liml depvar start end
# list of explanatory variables
```

Full Information Maximum Likelihood (FIML)

RATS has no specific instruction to do FIML for simultaneous equations. However, for small systems, you can implement FIML using the instruction **NLSYSTEM**. This assumes that you have Normal residuals with an unrestricted covariance matrix. You would set everything up as you would for 3SLS, except that, instead of using the **INSTRUMENTS** option, you include the option **JACOBIAN=FRML** for the Jacobian. For the linear model $Y \Gamma = X \beta + u$, this will provide a formula which computes $|\Gamma|$. If you’re interested in doing this, see the example which follows.
## Chapter 11: Simultaneous Equations

### Example 11.1 Simultaneous Equations Estimation

We will be demonstrating the various estimation techniques using a small model from Pindyck and Rubinfeld (1998), p. 390. Their model is

\[
C_t = \alpha_C + \beta_C Y_t + \gamma_C C_{t-1} + u_{Ct}
\]

\[
I_t = \alpha_I + \beta_I (Y_{t-1} - Y_{t-2}) + \gamma_I Y_t + \delta_I R_{t-4} + u_{It}
\]

\[
R_t = \alpha_R + \beta_R Y_t + \gamma_R (Y_t - Y_{t-1}) + \delta_R (M_t - M_{t-1}) + \epsilon_R (R_{t-1} + R_{t-2}) + u_{Rt}
\]

\[
Y_t \equiv C_t + I_t + G_t
\]

The data set consists of the following variables, quarterly from 1950:1 to 1985:4.

- **CONS (C)** = real personal consumption
- **INVEST (I)** = real gross domestic investment
- **GNP (Y)** = real GNP net of exports and imports
- **GOVT (G)** = real government purchases of goods and services
- **MONEY (M)** = M1
- **RATE (R)** = 90 day Treasury bill rate

The variables \((Y_t - Y_{t-1})\), \((M_t - M_{t-1})\) and \((R_t + R_{t-1})\) are created from these.

We will end up using a slightly different form for the investment equation. Ours will add an \(I_{t-1}\) term to the second equation above. This is done because an estimate of the autocorrelation coefficient on the residuals in the equation above comes in very close to one.

This example (`SIMULEST.PRG`) demonstrates two-stage least squares, two-stage least squares with AR1 errors, three-stage least squares using both **SUR** and **NLSYSTEM** and full-information maximum likelihood. For the forecasting examples later in the chapter, we have created a “source” file called `PRSETUP.SRC` which estimates the model using 2SLS, and performs all other operations which must be done before doing forecasts.

```plaintext
open data prsmall.xls
cal(q) 1947
data(format=xls,org=obs) 1947:1 1988:1

set ydiff = gnp-gnp{1}
set rsum  = rate+rate{1}
set mdiff = m-m{1}
```

*First, we estimate the equations by 2SLS.*

```plaintext
instruments constant cons{1} ydiff{1} gnp{1} govt mdiff $
rate rate{4}$
```
Pindyck and Rubinfeld note the low Durbin-Watson on the investment equation. This is re-estimated using AR1. Here, we try both the Cochrane–Orcutt method (actually Fair’s procedure) and the Hildreth–Lu method.

```
instruments constant cons{1 2} ydiff{1 2} gnp{1} $
govt{0 1} mdiff{0 1} rate{0 to 5}
```

```
ar1(method=hilu,inst) invest 1950:1 1985:4
# constant ydiff{1} gnp rate{4}
ar1(method=corc,inst) invest 1950:1 1985:4
# constant ydiff{1} gnp rate{4}
```

Hildreth–Lu gives a very different result from Cochrane–Orcutt, with a Rho effectively of 1.0. We therefore replace P&R’s investment equation with one redone to provide better dynamic behavior. The investment equation is altered by adding lagged investment to the explanatory variables. The complete model is then re-estimated with an updated set of instruments.

```
instruments constant cons{1} ydiff{1} gnp{1} invest{1} $
govt mdiff rate rate{4}
```

```
linreg(inst,frml=conseq) cons 1950:1 1985:4 residc
# constant gnp cons{1}
linreg(inst,frml=investeq) invest 1950:1 1985:4 residi
# constant invest{1} ydiff{1} gnp rate{4}
linreg(inst,frml=rateeq) rate 1950:1 1985:4 residr
# constant gnp ydiff mdiff rsum{1}
```

**LIML estimation uses the LIML procedure**

```
@liml cons 1950:1 1985:4
# constant gnp cons{1}
@liml invest 1950:1 1985:4
# constant invest{1} ydiff{1} gnp rate{4}
@liml rate 1950:1 1985:4
# constant gnp ydiff mdiff rsum{1}
```
We now estimate the model using 3SLS with the instruction SUR

```r
equation consleq cons
  # constant gnp cons{1}
equation investleq invest
  # constant invest{1} ydiff{1} gnp rate{4}
equation rateleq rate
  # constant gnp ydiff mdiff rsum{1}

sur(inst, iterations=10) 3 1950:1 1985:4
  # consleq residc
  # investleq residi
  # rateleq residr
```

We next estimate the model using 3SLS with the instruction NLSYSTEM. (Not necessary here but would be if there were some non-linearities).

```r
nonlin(parmset=structural) c0 c1 c2 i0 i1 i2 i3 i4 r0 r1 r2 r3 r4
frml consnl cons = c0+c1*gnp+c2*cons{1}
frml investnl invest = i0+i1*invest{1}+i2*ydiff{1}+
  i3*gnp+i4*rate{4}
frml ratenl rate = r0+r1*gnp+r2*ydiff+r3*mdiff+r4*rsum{1}
nlsystem(inst, parmset=structural, cvout=v) 1950:1 1985:4 $
  consnl investnl ratenl
```

Finally, we estimate the model by FIML. This adds to the standard likelihood for a multivariate Normal model, the log det of the Jacobian of the transformation from residuals to endogenous variables. This is done using the Jacobian option, which gives the determinant. In this case, this actually would reduce to 1–C1–I3, but we write out the full expression.

```r
frml jacobian = %det(||1.0 ,0.0 ,0.0 , -c1|$
  0.0 ,1.0 ,0.0 , -i3|$ 
  0.0 ,0.0 ,1.0 ,0.0|$
  -1.0,-1.0 ,0.0 ,1.0||)
nlsystem(parmset=structural, cvout=v, jacobian=jacobian,$
  title="FIML Estimates") 1950:1 1985:4 consnl investnl ratenl
```
Chapter 11: Simultaneous Equations

11.2 Setting Up a Model for Forecasting

The MODEL Data Type

A MODEL is a special data type in RATS which describes a collection of equations or formulas for forecasting purposes. For a simultaneous equations model, this is usually created with the instruction GROUP. The MODEL data type is also used in VAR analysis (Chapter 10). In that case, the SYSTEM instruction is the standard creator.

There is no hard limit on the number of equations in a model. However, if you have a 100+ equation model, you may be better off with a more specialized program.

There are a number of functions and operations which you can apply to the MODEL data type. For instance, the “+” operator applies to models to combine two smaller models into a larger one. The functions %MODELSIZE(model) returns the number of equations and %MODELDEPVARS(model) returns a VECTOR of INTEGERS which lists the dependent variables of the equations.

Creating Formulas

A model which is linear in its endogenous variables can be forecast using methods similar to those for VAR’s and ARIMA models. However, our emphasis will be on more general models which could have non-linear aspects, such as mixed logs and levels, or even explicitly non-linear equations. Your model can include linear equations defined using the DEFINE option of an instruction like LINREG or BOXJENK, but any non-linearities have to be handled using FRMLs. A FRML is a description of a (possibly) non-linear relationship.

There are five ways to create the FRMLs for use in simultaneous equations:

- **LINREG(FRML=formula to define)** or **AR1(FRML=formula to define)** defines a formula from the estimated equation.

- **FRML** defines formulas with free parameters (specified by NONLIN) to be estimated by NLLS or NLSYSTEM. Or, with fixed parameters, it allows you to input directly a structural equation, if, for instance, you estimated it outside of RATS.

- **FRML(EQUATION=equation to convert)** or **FRML(LASTREG)** converts an already estimated (linear) equation to a formula.

- **FRML(IDENTITY)** creates an identity.

- **EQUATION(FRML=formula to define)** associates the named formula with the equation being defined. If the equation is estimated (for instance with SUR or LINREG(EQUATION=equation)) the formula is automatically redefined. This association is broken if you ever change the equation in such a way that something changes besides its coefficients.
Chapter 11: Simultaneous Equations

The GROUP Instruction

You put a model together using the instruction GROUP. This creates a model from a set of formulas or equations. Each endogenous variable must be on the left side of one and only one formula. See the discussion below for tips on bringing your model into compliance. If you are doing any type of simulation which adds shocks to the equations, it will also be necessary to list identities last.

The GROUP instruction also allows you to specify the series which are to receive the forecasted values. This is done by appending >>series to a formula on the list. If you are planning to do simulations with Normally distributed shocks, you will also provide the covariance matrix for those shocks on GROUP using the CV option.

The models created with GROUP can be “added” using the notation model1+model2, which appends model 2 to model 1. This allows you to define separate “sector” models and combine them for forecasting.

There are quite a few functions for manipulating the MODEL data type, but most of those apply mainly to vector autoregressions or similar types of multivariate time series models.

Swapping Endogenous Variables

Because RATS uses a Gauss–Seidel algorithm to solve models, each endogenous variable must be on the left side of one and only one formula. If you estimate an equation with an exogenous variable on the left (such as a money demand equation), or a pair of equations with the same variable on the left, you must rearrange an equation to make it compatible with GROUP. If the equation is linear and is estimated by LINREG, AR1 or SUR, you can accomplish this by using MODIFY and VREPLACE with SWAP:

```
linreg(inst,define=mtemp) money
# constant gnp rate
modify mtemp
vreplace money by rate swap
```

This rewrites the equation with SWAP, rather than MONEY, as the dependent variable. You can see the results by using the PRINT option on VREPLACE.

Definitional Identities

If you have a model with mixed logs and levels, or other transformations of endogenous variables, you need to add to your model identities which define them. Usually, these will just mimic the SET instructions used to make the transformations:

```
set loggnp = log(gnp)
frml(identity) loggnpid loggnp = log(gnp)
```

If you don’t do this, the forecasting procedure will treat LOGGNP and LOG(GNP) as unrelated variables.
11.3 Solving Models: Various Topics

Gauss–Seidel Algorithm

RATS solves models using the Gauss–Seidel algorithm. This is a simple, and usually effective, algorithm which requires no complex calculations. Simply put, it solves

\[
\begin{align*}
y_1 &= f_1(y_2, y_3, \ldots, y_n) + u_1 \\
y_2 &= f_2(y_1, y_3, \ldots, y_n) + u_2 \\
& \vdots \\
y_n &= f_n(y_1, y_2, \ldots, y_{n-1}) + u_n
\end{align*}
\]

by setting \(y_1\) using the first equation, using initial values for the other \(y\)'s, then setting \(y_2\) using the second equation, using the initial values of \(y_3\) to \(y_n\) and the just computed value for \(y_1\), etc. From this, it's clear why each endogenous variable can be on the left side of one and only one equation.

This process continues until convergence, which means that the \(y\)'s change little from one iteration to the next. Convergence is not guaranteed, however, even if a solution exists, and even if the system is linear. For instance, the system below won't converge as written. It will converge if the equations are renormalized, and it will also converge with the use of a damping factor of .5 or less—see the next page.

\[
\begin{align*}
y_1 &= y_2 + 5 \\
y_2 &= -2y_1 + 3
\end{align*}
\]

Note the following:

- The initial values for the \(y\)'s come from the data series themselves, if available; otherwise the last historic value is taken. In multiple step forecasts, the step \(k\) solution initializes step \(k+1\).
- Convergence is considered to be achieved when all endogenous variables satisfy

\[
\min\left( \left| \frac{y_k - y_k^0}{y_k^0} \right|, \left| y_k - y_k^0 \right| \right) < \varepsilon
\]

where \(y_k^0\) is the value from the previous iteration. That is, the percentage change is less than \(\varepsilon\) for larger values, and the absolute change is less than \(\varepsilon\) for smaller ones (less than 1.0). You control \(\varepsilon\) with the \texttt{CVCRIT} option on the instructions: \texttt{FORECAST}, \texttt{STEPS}, \texttt{THEIL} or \texttt{SIMULATE}.

These instructions also have the options \texttt{ITERS=number of iterations}, and \texttt{DAMP=damping factor} which apply to the Gauss–Seidel solution procedure.
Chapter 11: Simultaneous Equations

Achieving Convergence

If you have problems getting convergence, there are several adjustments which you can make. Obviously, if a (non-linear) system has no solution, nothing will work.

- Increase the \( \text{ITER} \) option on the \text{FORECAST}, \text{STEPS}, \text{THEIL}, or \text{SIMULATE} instruction from the default of 50. This can help if the procedure is converging, but just moving very slowly.
- Reorder the equations. This is probably a hit-or-miss strategy, since it may be hard to identify the source of the poor behavior.
- Use the \text{DAMP} option. \( \text{DAMP} = \lambda \) causes RATS to use

\[ y_k = (1 - \lambda) y_k^0 + \lambda (f_k(...) + u) \]

a weighted average of the old and new values. The default is \( \lambda = 1 \). Taking a value of \( \lambda < 1 \) will slow the solution process, but a small enough value of \( \lambda \) will eliminate the explosive behavior of systems like the one on the previous page.

*If you’re getting NA’s for some or all of your forecasts, it is almost certainly because you have some exogenous variables which aren’t defined during your forecast period.* See the “Out-of-Sample Forecasts” segment below.

Out-of-Sample Forecasts

To compute forecasts out-of-sample, you need to make some assumptions about the future of the exogenous variables. There are two ways to handle these:

- Add to the model simple equations, such as autoregressions, to forecast them.
- Set paths for them over the forecast period, using \text{SET} for systematic paths and \text{DATA} otherwise.

You can, of course, combine these two, treating different variables differently. This short example (\text{SIMULFOR.PRG}) uses the first method, while the add factor example later in the section uses the second. Our example has the slight complication that \text{MDIFF}, not \text{MONEY} itself, is needed in the interest rate equation, thus requiring an additional identity to define this to the model.

```
source prsetup.src
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)
```
Multipliers

You compute multipliers by running the model with and without a set of changes to
the paths of the exogenous variables. Unless the model is linear (this one is), the
computed multipliers apply only to the particular context—they will change when
you solve the model at a different period, or with different assumptions on the other
exogenous variables.

You must compute the two sets of forecasts and subtract them to get the multipliers,
which can involve some rather complicated bookkeeping. The easiest way to do this
is with the RESULTS option on FORECAST which creates a VECTOR of SERIES for the
results. The DO loop in this example loops over the number of equations in the model
and creates labels of “M_dependent variable” for the multipliers.

This is example SIMULMUL.PRG.

source prsetups.src
smpl 1984:1 1985:4
forecast(model=prsmall,results=base)
compute govt(1984:1)=govt(1984:1)+1.0
increase govt
forecast(model=prsmall,results=mults)
compute govt(1984:1)=govt(1984:1)-1.0
reset govt
do i=1,%rows(mults)
   set mults(i) = (mults(i)-base(i))/2.0
   labels mults(i)
      # 'M_'+%label([series]%modeldepvars(prsmall)(i))
end do i
print(picture="*.###") / mults

Historical Simulations

You can use the instruction THEIL to compute statistics on the accuracy of the model
using the historical data. Because this solves the model for each time period you loop
over, it can take a long time with a big model.

This looks at forecasts for 1 to 4 steps ahead over the period from 1982:1 to 1985:4.
The TO=1985:4 on THEIL(SETUP,...) indicates the end of the historical data.

This is example SIMULTHE.PRG.

source prsetups.src
theil(setup,model=prsmall,steps=4,to=1985:4)
do time=1982:1,1985:4
   theil time
end do time
theil(dump,window="Forecast Performance")
Add Factors

Add factors adjust the solution of the model by altering one or more intercepts, or, equivalently, adding a non-zero error term to one or more equations. You can implement add factoring in RATS using either the SHOCKS or INPUT options of FORECAST. INPUT (which uses a supplementary cards) is more convenient for small models and SHOCKS (which uses a VECTOR) is better for larger ones. If you need to add factor more than one period, use the PATHS option.

This example (SIMULADD.PRG) does three forecasts, bumping the first period interest rate by 0.5 and 1.0 points for the second and third. (Adding x to the interest rate (3rd) equation adds x to RATE, since R_t appears only in that equation. Generally, the effect of a change will not be quite as predictable).

```
source prsetup.src
smpl 1986:1 1988:4
set m 1986:1 1988:4 = m{1}*1.01
set govt 1986:1 1988:4 = govt{1}*1.008
set mdiff 1986:1 1988:4 = m-m{1}
declare real fudge
dofor fudge = 0.0 0.5 1.0
    forecast(model=prsmall,input,print)
    # 0.0 0.0 fudge
end dofor
```

A similar setup using SHOCKS would use (following the SET MDIFF)

```
declare vector shocks(3)
compute shocks=%const(0.0)
declare real fudge
dofor fudge = 0.0 0.5 1.0
    compute shocks(3)=fudge
    forecast(model=prsmall,shocks=shocks,print)
end dofor
```
Chapter 12
Special Models and Techniques

This chapter describes some special techniques and models. The lead section is on ARCH and related models. Section 12.2 covers dynamic linear (state-space) models and 12.3 Markov switching models. 12.4 deals with non-parametric regression, 12.5 linear and quadratic programming and 12.6 covers neural networks.
12.1 ARCH and Related Models

ARCH (AutoRegressive Conditional Heteroscedasticity), GARCH, and related models have become very popular. Thanks to the flexible maximum likelihood estimation capabilities of RATS, it has proven to be an excellent tool for estimating standard ARCH and GARCH models, as well as the many complex variants on them.

The instruction **GARCH** can handle most of the more standard ARCH and GARCH models. However, because refinements on these models are being developed, it’s still useful to know how to do the calculations using more basic instructions.

In this section, we briefly introduce the basic theory underlying these models, and demonstrate the standard programming set ups required to estimate them. See Enders (2003), Tsay (2002), Hamilton (1994), or Campbell, Lo and MacKinlay (1997) for more information.

Much of the discussion in this section is based upon an article for the RATSletter written by Rob Trevor of Macquarie University. Our thanks to Rob for his many contributions to the RATS community over the years.

Autoregressive conditional heteroscedasticity (ARCH) was proposed by Engle (1982) to explain the tendency of large residuals to cluster together. A very general form for an ARCH or GARCH model is

\[
\begin{align*}
  y_t &= X_t \beta + u_t \\
  h_t &= \text{var}(u_t) = h(u_{t-1}, u_{t-2}, \ldots, u_{t-q}, h_{t-1}, h_{t-2}, \ldots, h_{t-p}, X_t, X_{t-1}, \ldots, X_{t-k}, \alpha)
\end{align*}
\]

where \( \alpha \) is a set of unknown parameters for the \( h \) function. Assuming the residuals are Normally distributed, the log likelihood term for entry \( t \) (omitting additive constants, although they’re included in output from **GARCH**) takes the form:

\[
\begin{align*}
  -\frac{1}{2} \log h_t - \frac{1}{2} (y_t - X_t \beta)^2 / h_t
\end{align*}
\]

where \( h_t \) is \( h(\cdot) \) evaluated at \( t \). This has a standard regression model for the mean of the series, and a separate model for the variance. We’ll refer to these as the “mean model” and “variance model.” These can be, and often are, analyzed separately. The parameters for (1), for instance, can be estimated consistently by least squares; they just won’t be efficient if they don’t take into account the heteroscedasticity.

In many cases the mean model is just the intercept, and that is the default for the **GARCH** instruction.
12.1.1 Standard Models

ARCH Models

The simplest “variance model” is the ARCH(q) model:

\[ h_t = c_0 + a_1 u_{t-1}^2 + a_2 u_{t-2}^2 + \ldots + a_q u_{t-q}^2 \]  

Assume that the dependent variable is the daily appreciation of the Deutsche Mark vs the dollar, and the model to be estimated is an ARCH(6). You can estimate this using GARCH by:

```
garch(q=6) / dlogdm
```

The GARCH instruction is designed to handle univariate or multivariate models, so the syntax has the estimation range first (hence the /) followed by the list of dependent variable series.

GARCH Models

Even Engle in his original paper ran into some problems implementing the ARCH model with real world data. Volatility seems to be quite a bit more persistent than can be explained by an ARCH(1) model, where variance depends only upon last period’s residual. It’s possible to add additional lags of the squared variance, as was done in the example above, but unconstrained estimates of the lags will often show some negative coefficients. Engle used four lags, but had to constrain the shape of the lag distribution.

Bollerslev (1986) proposed the GARCH model as an alternative. In a GARCH model, the variance term depends upon the lagged variances as well as the lagged (squared) residuals. This allows for persistence in volatility with a relatively small number of parameters. The variance model for the standard GARCH(p,q) is

\[ h_t = c_0 + a_1 u_{t-1}^2 + a_2 u_{t-2}^2 + \ldots + a_q u_{t-q}^2 + b_1 h_{t-1} + b_2 h_{t-2} + \ldots + b_p h_{t-p} \]  

To estimate a GARCH(1,1) on the same series as above, use

```
garch(p=1,q=1) / dlogdm
```
Chapter 12: Special Models

Exponential Models

Nelson (1991) introduced a number of refinements on the GARCH model, which we'll examine separately in this section, as you can choose them separately on the **GARCH** instruction. The first was to model the log of the variance, rather than the level. This avoids any problem that could arise because of negative coefficients in standard ARCH and GARCH models, and gives rise to an EGARCH model. The form for the variance that we use in RATS is

\[
\log h_t = c_0 + a_1 |u_{t-1}|/\sqrt{h_{t-1}} + a_2 |u_{t-2}|/\sqrt{h_{t-2}} + \ldots + a_q |u_{t-q}|/\sqrt{h_{t-q}} + \\
\quad b_1 \log h_{t-1} + \ldots + b_p \log h_{t-p}
\]

A few things to note about this. First, it doesn’t include “asymmetry” effects, where positive and negative \(u\)’s have different effects. That will be described a bit later.

Nelson also used an alternative distribution to the normal, which is discussed in the next subsection.

EGARCH models are often parameterized using

\[
|u_{t-i}|/\sqrt{h_{t-i}} = E \left( |u_{t-i}|/\sqrt{h_{t-i}} \right),
\]

which, by definition, has mean 0. The simplification used in (6) affects only the constant term in that formula.

To estimate a GARCH(1,1) with an exponential variance model, use

```
garch(p=1,q=1,exp) / dlogdm
```

IGARCH Models

The IGARCH model is also due to Nelson (1990). This constrains the GARCH model coefficients to sum to one, that is,

\[
a_1 + a_2 + \ldots + a_q + b_1 + \ldots + b_p = 1
\]

To estimate an IGARCH model, include the **I=NODRIFT** or **I=DRIFT** option on **GARCH**. **I=NODRIFT** imposes the constraint (8) and zeros out the constant \((c_0)\) in the variance equation. **I=DRIFT** also imposes (8) but leaves the constant free.

To estimate an IGARCH (1,1) model (without variance drift), use

```
garch(p=1,q=1,i=nodrift) / dlogdm
```
12.1.2 Fat-Tailed Distributions

In many cases the assumption of conditional normality cannot be maintained. RATS offers two choices as alternatives: the Student-t and the Generalized Error Distribution (GED). You choose which to use with the `DISTRIB` option, which has choices `NORMAL` (the default), T and GED. Both the Student-t and GED have a shape parameter, which determines their kurtosis, and a scale parameter, which determines the variance given the shape parameter. Since ARCH or GARCH models a variance, the t or GED density selects the scale to give that variance. For instance, the GED has density function

\[
\frac{\exp\left(-\frac{|x|}{b}\right)^{c/2}}{b\left(2^{c+1}\right)\Gamma(1+c/2)}
\]

where \(c\) is the shape and \(b\) the scale. The variance, given \(b\) and \(c\), is

\[
2^c b^2 \Gamma(3c/2) / \Gamma(c/2)
\]

Given a variance from the model and the value of the shape parameter, (10) is solved for \(b\), and (the log of) (9) is used in maximum likelihood estimation.

Both densities have the normal as a special case. For the GED, that’s with shape parameter \(c = 1\), for the t, it’s with infinite degrees of freedom. The GED family includes both fat-tailed densities (\(c > 1\)) and thin-tailed ones (\(c < 1\)). The t with finite degrees of freedom is always fatter tailed than the normal. Note that, because the t has no variance if the degrees of freedom are less than or equal to 2, the density function (as rescaled by RATS) won’t be defined for those values.

You can either set a value for the shape parameter using the option `SHAPE`, or have it estimated, which is the default if you don’t include `SHAPE`. Continuing with our example, the following two instructions would estimate a GARCH(1,1) with a t distribution for the errors. In the first, the degrees of freedom are pegged at 5, while in the second, they’re estimated.

\[
\begin{align*}
garch(p=1,q=1,distrib=t,shape=5) & / dlogdm \\
garch(p=1,q=1,distrib=t) & / dlogdm
\end{align*}
\]
12.1.3 Asymmetry

It has long been recognized that equity returns exhibit asymmetrical conditional variance behavior, that is, that positive values of the residuals have a different effect than negative ones. That won’t be captured by any of the models above, since the residual always enters the variance as a square or absolute value. The EGARCH model from Nelson (1991) adds an extra term to (6) to provide for this; an EGARCH(1,1) would have the variance evolving according to

\[ \log h_t = c_0 + a_1 |u_{t-1}| / \sqrt{h_{t-1}} + b_1 \log h_{t-1} + d_1 u_{t-1} / \sqrt{h_{t-1}} \]

There are many equivalent ways to introduce the asymmetric effect into the model; we choose this one to maintain a similar form to that used in other models you can estimate with GARCH. Note that, with this parameterization, a negative value of \( d_1 \) means that negative residuals tend to produce higher variances in the immediate future.

An analogous change to the standard GARCH model was proposed by Glosten, et. al. (1993) and is known as GJR, after the originators. Again, looking at a GARCH(1,1), the GJR variance model is

\[ h_t = c_0 + a_1 u_{t-1}^2 + b_1 h_{t-1} + d_1 u_{t-1}^2 I_{u_{t-1} < 0} (u_{t-1}) \]

where \( I \) is an indicator function, in this case, for \( u > 0 \). With this formulation, a positive value of \( d_1 \) means negative residuals tend to increase the variance more than positive ones.

To estimate an EGARCH and a GARCH with asymmetric effects:

- \texttt{garch(p=1,q=1,exp,asymmetric) / dlogdm}
- \texttt{garch(p=1,q=1,asymmetric) / dlogdm}
12.1.4 The Mean Model

Up to now, we’ve ignored the model for the mean. All of the examples above have used only an intercept. There is one model for the mean that is even simpler than this, where the dependent variable is already thought to be a mean zero process. To estimate the model under that assumption, use the option NOMEAN.

For a less trivial model, include the option REGRESSORS and add a supplementary card with the explanatory variables for the mean. Include CONSTANT if it’s one of the explanatory variables. For instance, the following will do a GARCH(1,1) with an AR(1) process for the mean:

```
garch(p=1,q=1,regressors) / dlogdm
# constant dlogdm{1}
```

If you want an ARMA model for the mean, use lags of %MVGAVGE in your list of regressors for any moving average terms. For instance, the same model, but with an ARMA(1,1) is

```
garch(p=1,q=1,regressors) / dlogdm
# constant dlogdm{1} %mvgavge{1}
```

Pre-sample values for the moving average process are set to zero. The residuals are generated recursively conditioned on that.

ARCH-M, GARCH-M

ARCH-M models (Engle, Lilien, Robins (1987)) generalize the ARCH model by allowing a function of the variance to enter the regression function itself. The most common form of this nowadays uses the variance itself as a regressor. You can estimate an ARCH-M or GARCH-M model by using the special name %GARCHV among the regressors. This refers to the variance generated by the model. It behaves like a series, so you can include lags of it with the standard {...} notation. To add the current variance to an AR(1)-GARCH(1,1) model:

```
garch(p=1,q=1,regressors) / dlogdm
# constant dlogdm{1} %garchv
```

The EQUATION option

You can also use the option EQUATION to indicate both the dependent variable and the mean model. For instance,

```
boxjenk(ar=1,ma=1,constant,define=arma11) dlogdm
```

```
garch(p=1,q=1,equation=arma11)
```

estimates an ARMA(1,1) model first by least squares, then with GARCH errors.
12.1.5 ARCH-X, GARCH-X Models
If you add other explanatory variables to your variance model besides those formed from the lagged residuals and lagged variances, you get an ARCH-X or GARCH-X model. To do this, use the option XREGRESSORS and add a supplementary card listing the extra explanatory variables in the variance equation. (For these, don’t include CONSTANT, which will always be in the variance model). If you need both regressors for the mean model and for the variance model, put the supplementary card for the mean model regressors first. For instance, suppose that you have a dummy variable for the Mondays, and you think that the variance is systematically different on Mondays.

```
garch(p=1,q=1,regressors,xreg) / dlogdm
# constant dlogdm{1}
# monday
```

12.1.6 Estimation Options
The models are all estimated using maximum likelihood. Because the normal is a special case of both the t and the GED, the log likelihood used by GARCH includes all the integrating constants so the function values produced under different choices for the distribution will be comparable.

These types of models have a technical issue that needs to be addressed in estimating them: the variance is generated recursively and the pre-sample values of it are unobservable, and, unfortunately, the estimates can be somewhat sensitive to the choice made for these. For the basic ARCH model, it’s possible to start the estimation at entry \( q+1 \), as the variance depends only upon the lagged \( u \)'s, which are (in general) computable given the set of parameters. If you wish to do this, use the option CONDITION. However, that isn’t available for a GARCH model, because the required lag of \( h \) can’t be computed no matter how many data points are used.

By default, the GARCH instruction handles both the pre-sample lagged squared \( u \)'s and the lagged variances by setting them equal to the unconditional estimates of the variance: the one obtained from a least squares estimate of the mean model. If you wish to use a different value, you can include the option PRESAMPLE=pre-sample variance.

The GARCH instruction provides four estimation methods, which you select with the METHOD option. The choices are BFGS, BHHH, SIMPLEX and GENETIC, all of which are described in Chapter 7. The default is BFGS. BHHH is a common choice, since the model is in the form that can be estimated using that; however, that isn’t necessarily a good choice as the main estimation method, as the curvature far from the maximum can be quite different from that calculated by BHHH. If you want the BHHH covariance matrix, it’s better to use BFGS as a preliminary method, then switch to BHHH for a final iteration once it’s converged. The derivative-free methods (SIMPLEX and GENETIC) are very useful in multivariate GARCH models, but aren’t quite as important with the univariate ones. However, if you’re having convergence problems
with BFGS or BHHH, you can use the PMETHOD with SIMPLEX or GENETIC and PITERS options to refine the initial guesses.

An alternative to using one of the non-normal distributions is to obtain quasi-Maximum Likelihood estimates by using the log likelihood function from the conditional normal specification. As this is not the true likelihood, it requires calculating a robust estimate of the covariance of the parameter estimates. The BFGS algorithm with the ROBUSTERRORS option produces the required calculation:

\[
garch(method=bfgs,robust,other \ options) \ldots
\]

Under fairly weak conditions, the resulting estimates are consistent even when the conditional distribution of the residuals is non-normal (Bollerslev and Wooldridge (1992)). The covariance matrix calculated by RATS is not numerically identical to the one referred to in these papers, but is asymptotically the same. Calculations similar to those in RATS have been used by others (for example, McCurdy and Morgan (1991)).

### 12.1.7 Output

The output from a GARCH instruction is similar to that from a LINREG, but with fewer summary statistics; for instance, no $R^2$. The coefficients are listed in the order:

1. Mean model parameters in the standard order for regressions. If you use the default mean model, the coefficient will be labeled as Mean. If you used the REGRESSORS option, these will be labeled as they would for any other regression.
2. Constant in the variance equation (labeled as C).
3. “ARCH” (lagged squared residuals) parameters, in increasing order of lag (labeled as A, or A(lag) if you used more than one lag)
4. “GARCH” (lagged variance) parameters, if any, in increasing order of lag (labeled as B or B(lag))
5. Asymmetry coefficients if any (labeled as D).
6. XREGRESSORS variables, labeled as they would in any other regression.

This is also the order in which the coefficients appear in the %BETA vector defined by GARCH, and are the order for the INITIAL vector, if you want to use that.

### RESIDS and HSERIES options

For a univariate model, you can get the residuals and series of variance estimates using the RESIDS and HSERIES options. (Different options are needed for multivariate models). The following, for instance, will produce in the series USTANDARD the standardized residuals from a GARCH(1,1).

\[
garch(p=1,q=1,resids=u,hseries=h) / dlogdm
set ustandard gstart gend = u/sqrt(h)
\]

GARCH also defines the %RESIDS series when you estimate a univariate model.
Chapter 12: Special Models

12.1.8 Using MAXIMIZE

While the GARCH instruction can handle a wide range of standard GARCH models, there will always be recent refinements which don’t fit into its framework. In most cases, these can be estimated using MAXIMIZE by making adaptations to one of a few basic models. We’ve found that it’s easiest to do this if the key parts of the log likelihood are computed by separate FRMLs. Thus, there are seven steps used in specifying ARCH, GARCH, and like models in RATS:

1. use the NONLIN instruction to specify the parameters to be estimated
2. use FRML to specify the conditional mean(s)
3. use FRML to specify the conditional variance(s)
4. use FRML to specify the log likelihood using the mean and variance formulas
5. set the initial values of the parameters
6. set the pre-sample values of the residuals and the conditional variance
7. use the MAXIMIZE instruction to compute the estimates.

Using separate PARMSETs for the mean and variance models is also a good practice. See Section 7.6 for more on that.

The following program segment implements a GARCH(1,1) model, with a variance function of \( h_t = c + au_{t-1}^2 + bh_{t-1} \). The values for the residuals, the squared residuals and the variances are generated (recursively) and saved into the series U, UU and H. The contents of these series will change with each function evaluation. The use of the UU series simplifies some of the calculations, since the model is generating its expected value.

It’s important to note that when you use MAXIMIZE, you must provide an explicit start for the estimation range. Because of the recursive nature of the calculation, MAXIMIZE can’t easily figure out which entry is the first one computable. In this case, the first entry is 3, allowing for one lag due to the AR(1), and another because the input series was generated as a log first difference.

```
linreg dlogdm
# constant dlogdm{1}
frml(lastreg,vector=beta) meanf
nonlin(parmset=meanparms) beta
*
  set uu = %seesq
  set h  = %seesq
  set u  = 0.0
*
nonlin(parmset=garchparms) c a b
  compute c=%seesq,a=.05,b=.05
  frml varf = c+a*uu{1}+b*h{1}
  frml logl = (u(t)=dlogdm-meanf),$
              (uu(t)=u**2),(h(t)=varf(t)),%logdensity(h,u)
maximize(parmset=meanparms+garchparms) logl 3 *
```
If you try to estimate a GARCH model using **MAXIMIZE** and get the error message:

**Missing Values And/Or SMPL Option Leave No Usable Data Points**

your estimation range probably starts too soon for your data. Check the range of your data and adjust the **MAXIMIZE** instruction appropriately. If you still have problems, you can test the settings of your formulas with (for instance),

```plaintext
set testlog start end = logl(t)
print start end testlog u h
```

where `start` and `end` are the starting and ending dates you want to use for your estimation. See if the printed values make sense. If you see NA’s (missing values) in some or all of the series, work your way back and check the calculations in question to find where they’re failing. In a GARCH model, once there’s a single bad data point, all later time periods will also be dropped since the variance term isn’t computable.

If the **MAXIMIZE** seems to work, but you’re losing observations unexpectedly, do the combination of **SET** and **PRINT** above. Check your data to make sure there isn’t a missing value. If nothing turns up, do the same thing before the **MAXIMIZE** (right after setting your initial guess values). RATS will throw out any data points which produce NA’s given the initial parameter settings. So, if you see NA’s in the output, you may need to change your initial values.

The GARCH model has one numerical problem not shared by ARCH. It is possible for the variance to “explode” for certain values of the parameters. In estimating the unknown parameters, RATS may examine some of these bad values with uncertain consequences. We have two suggestions for dealing with this:

1. Use **PMETHOD=SIMPLEX** for a small number of preliminary iterations to (in effect) improve your initial estimates. Use one of the others as your main estimation method.

2. Replace the **VARF** formula with the following, which forces an NA value if the variance gets near an overflow:

   ```plaintext
   frml garchvar = c + a*uu{1} + b*%ovcheck(h{1})
   ```
12.1.9 Out-of-Sample Variance Forecasts

For the simple GARCH(1,1) model, the variance evolves according to

\[ h_t = c + a h_{t-1} + b u_{t-1} \]  

(13)

The forecast of \( h_{t+k} \) given information at time \( t-1 \) (denoted \( \Omega_{t-1} \)) can be derived as

\[ E(h_{t+k} \mid \Omega_{t-1}) = c + a E(h_{t+k-1} \mid \Omega_{t-1}) + b E(u_{t+k-1}^2 \mid \Omega_{t-1}) \]

\[ = c + (a + b) E(h_{t+k-1} \mid \Omega_{t-1}) \]  

(14)

(as long as \( k > 0 \)) which uses the fact that the expectation of the squared residuals is \( h \) by definition. Thus, multiple step forecasts can be obtained recursively. Note that the coefficient on the lagged term in the forecasting equation is the sum of the coefficients on the lagged variance term and the lagged squared residuals. When \( k = 0 \), we would be able to use sample values for both \( h_{t-1} \) and \( u_{t-1}^2 \).

Even though there are parallels in the structure, forecasting an ARCH or GARCH model is more complicated than forecasting an ARMA model, because the out-of-sample “residuals” can’t simply be set to zero, as they enter as squares. The following does the forecasts by defining two formulas: one produces a recursive estimate of the variance, the other produces the expected value of the squared residuals. Out-of-sample, these are, of course, the same. They differ, however, in their values at the end of the sample, where the lagged \( uu \) series are the sample values of the squared residuals, while the lagged \( HT \) series are the estimated variances. This estimates a GARCH(1,1) model and forecasts 120 periods from 1992:1 on. We need both the \texttt{HSERIES} and \texttt{RESIDS} options on the \texttt{GARCH} instruction, since both pieces are needed in constructing the forecasts.

\begin{verbatim}
garch(p=1,q=1,hseries=ht,resids=at) / sp500
Create the historical uu series
set uu = at**2
Copy the coefficients out of the \%beta vector
compute vc=%beta(2),vb=%beta(4),va=%beta(3)
frml hfrml  ht = vc+vb*ht(1)+va*uu(1)
frml uufrml uu = ht
group garchmod hfrml>>ht uufrml>>uu
forecast(model=garchmod,from=1992:1,steps=120)
\end{verbatim}

While this type of recursive calculation works for the simple ARCH and GARCH models, it doesn’t work for an \texttt{EGARCH} model. Forecasts for those have to be done by simulation or bootstrapping; methods described in chapter 13.
Example 12.1 ARCH and GARCH Models

This (example GARCHUV.PRG) is based upon an example from Verbeek (2004). It estimates a variety of ARCH and GARCH models. Many of these have already been discussed earlier in the section.

```
open data garch.asc
data(format=free,org=columns) 1 1867 bp cd dm jy sf *
set dlogdm = 100*log(dm/dm{1})

Estimates ARCH(6), GARCH(1,1) and EGARCH(1,1) with asymmetry. Variances for each are saved.
garch(p=0,q=6,hseries=hh06) / dlogdm
garch(p=1,q=1,hseries=hh11) / dlogdm
garch(p=1,q=1,exp,asymmetric,hseries=hha) / dlogdm

Graph the estimated standard errors for the three models
set h06 = sqrt(hh06)
set h11 = sqrt(hh11)
set hha = sqrt(hha)
graph(key=below,klabels="ARCH6","EGARCH11","GARCH11") 3
# h06 1770 *
# hha 1770 *
# h11 1770 *

Estimate a GARCH-M model
garch(p=1,q=1,regressors) / dlogdm
# constant %garchv

Estimate a GARCH(1,1) with an AR(1) mean model
garch(p=1,q=1,reg) / dlogdm
# constant dlogdm{1}

Same model estimated using MAXIMIZE
linreg dlogdm
# constant dlogdm{1}
frml(lastreg,vector=beta) meanf
nonlin(parmset=meanparms) beta *
set uu = %seesq
set h = %seesq
set u = 0.0 *
nonlin(parmset=garchparms) c a b
compute c=%seesq,a=.05,b=.05
frml varf = c+a*uu{1}+b*h{1}
frml logl = (u=dlogdm-meanf),(uu(t)=u**2),$
   (h(t)=varf(t)),%logdensity(h,u)
maximize(parmset=meanparms+garchparms) logl 3 *
```
12.1.10 Multivariate GARCH Models

The main problem with extending univariate models to a set of variables is that the covariance matrix has to be positive definite at each time period in order for the likelihood to be defined. Even if the variance of each equation stays positive, if the cross terms stray out of bounds for just one data point, a set of parameters gives an undefined function value. Also, the number of parameters can grow quite quickly as you increase the number of variables, and the estimation time grows as well. Except for some highly restricted parameterizations, the practical limit is three or four variables. For our example, we’ll be working with a set of three exchange rates of Japan, France and Switzerland versus the U.S. dollar, with a bit over 6000 daily observations.

A first attempt at an extension to several variables would model each variance or covariance term separately:

\[
H_{ij}(t) = c_{ij} + a_{ij} u_i(t-1) u_j(t-1) + b_{ij} H_{ij}(t-1)
\]

This is generally known as the multivariate GARCH(1,1) (or diagonal VECH) and is what the \texttt{GARCH} instruction will estimate if you do no other options besides picking \( p \) and \( q \). Now, the \texttt{HSERIES} and \texttt{RESIDS} options described before for univariate models won’t do for a multivariate model, as we need a covariance matrix and a vector of residuals at each entry. Their counterparts for multivariate GARCH are \texttt{HMATRICES} and \texttt{RVECTORS}. \texttt{HMATRICES} provides a \texttt{SERIES} of \texttt{SYMMETRIC} arrays and \texttt{RVECTORS} returns a \texttt{SERIES} of \texttt{VECTORS}. There is also an \texttt{MVHSERIES} option which gives the covariance matrix as a \texttt{SYMMETRIC} of \texttt{SERIES}, which is more useful in some situations.

\texttt{garch(p=1,q=1,iters=200,hmatrices=hh) / xjpn xfra xsui}

While this is the most “straightforward” extension of a univariate model, the lack of connection among the variance terms is implausible, and it may be hard for the parameter estimates to stay clear of regions where the covariance matrix is on the verge of being deficient rank at some data points.

The BEK formulation (Engle and Kroner (1995)) directly imposes positive definiteness on the variance matrix:

\[
H(t) = C'C + A'v(t)u'(t)A + B'H(t-1)B
\]

As each term is positive semi-definite by construction, this will avoid bad regions. However, while positive-definiteness is assured, it comes at the cost of a poorly behaved likelihood function. With all the parameters entering through quadratic forms, they aren’t globally identified: changing the signs of all elements of \( C, B \) or \( A \) will have no effect on the function value. In addition, \( C \) can have only \( n(n+1)/2 \) free parameters. \texttt{GARCH} parameterizes it to be lower triangular.
Starting from a standard set of initial guess values, it’s hard for one of the hill-climbing methods to make good progress. More than with any of the other multivariate models, BEK will benefit from a preliminary refinement using the simplex. You get the BEK (sometimes written BEKK) model by adding the option MV=BEK.

\[
garch(p=1, q=1, mv=bek, method=bfgs, iters=200, \$
pmethod=simplex, piter=10) / xjpn xfra xsui
\]

In addition, RATS offers the Vech model, which allows complete interaction among the terms

\[
(17) \quad vech(H(t)) = C + Avech(u(t-1)u(t-1)') + Bvech(H(t-1))
\]

The \textit{vech} operator takes a symmetric matrix and returns a vector with only its lower triangle. In RATS, symmetric arrays are stored by rows, so that’s how the coefficients are arranged. You implement this with the option MV=VECH. However, this has a large number of free parameters (even in a bivariate case), and it’s clearly unwieldy with more than two variables. Model (15) is sometimes known as a diagonal Vech because it is (17) with \(A\) and \(B\) constrained to be diagonal.

\textbf{Restricted Correlation Models}

The remaining methods all use \textit{GARCH} models for the variances, but generate the covariances in a more restricted fashion. The simplest of these is \textit{MV=DIAG}. This estimates separate univariate \textit{GARCH} models on each dependent variable. The “model” for the covariances between variables is that they are all zero. This allows you to do “two-step” procedures, which model the correlations based upon the standardized residuals from the univariate models. Using \textit{GARCH} to handle all the variables simultaneously (rather than doing separate univariate \textit{GARCH} instructions) ensures that they are estimated over a common range.

\[
garch(p=1, q=1, mv=diag, hmatrices=hd, r vectors=rd) / xjpn xfra xsui
\]

The next step up in complexity is the \textit{Constant Correlation} specification, which you can estimate with the option MV=CC. The covariances here are given by

\[
(18) \quad H_{ij}(t) = R_{ij} \sqrt{H_{ii}(t) H_{jj}(t)}
\]

where the off-diagonal (lower triangular) elements of \(R\) are estimated parameters.

\[
garch(p=1, q=1, mv=cc) / xjpn xfra xsui
\]

While this generally has a well-behaved likelihood function, and can handle a bigger set of variables than the more fully parameterized models, it does have the drawback of requiring the correlation to be constant. In some applications, time-varying correlations are essential. Engle (2002) proposed a method of handling this which he dubbed \textit{Dynamic Conditional Correlations}. This adds two scalar parameters which govern a “\textit{GARCH}(1,1)” model on the covariance matrix as a whole.
(19) \( \mathbf{Q}_i = (1-a-b)\mathbf{Q}_0 + au_{i-1}u'_{i-1} + b\mathbf{Q}_{i-1} \)

where \( \mathbf{Q}_0 \) is the unconditional covariance matrix.

However, \( \mathbf{Q} \) isn’t the sequence of covariance matrices. Instead, it is used solely to provide the correlation matrix. The actual \( \mathbf{H} \) matrix is generated using univariate GARCH models for the variances, combined with the correlations produced by the \( \mathbf{Q} \).

(20) \( \mathbf{H}_{ij}(t) = \mathbf{Q}_{ij}(t)\sqrt{\mathbf{H}_{ii}(t)\mathbf{H}_{jj}(t)/\sqrt{\mathbf{Q}_{ii}(t)\mathbf{Q}_{jj}(t)}} \)

These last three methods (DIAG, CC and DCC) all require models for the variances of the individual processes. The default is a standard GARCH model with each having the same number of GARCH parameters. You can make two other choices for this. The first is VARIANCES=EXP, which gives the multivariate E-GARCH model. The individual variance models are E-GARCH models as described in equation (6) (page 418) without the asymmetry term. (Asymmetric effects are permitted as described below). The other is VARIANCES=VARMA. This was proposed by Ling and McAleer (2003). The variance terms here take the form (for a 1,1 model):

(21) \( \mathbf{H}_{ii}(t) = c_{ii} + \sum_j a_{ij}u_j(t-1)^2 + \sum_j b_{ij}\mathbf{H}_{jj}(t-1) \)

This allows large shocks in one variable to affect the variances of the others. The following instruction estimates a VARMA-GARCH model with constant correlations.

\[ \text{garch(p=1,q=1,mv=cc,variances=varma,pmethod=simplex,piters=5,}$ \]
\[ \text{method=bfgs,iters=200) / xjpn xfra xsui} \]

Forecasting

There are two different schemes for calculating out-of-sample forecasts. The STANDARD and BEK models can be converted into special cases of the VECH, which can be forecast using a multivariate extension of (15). The CC and DIAG models use (15) to forecast the variances, which are then converted to a full covariance matrix by using the estimated correlation pattern. Use the MVGARCHFore procedure to do the calculations. Note that there are quite a few combinations of options which do not allow simple recursive forecasts. For instance, the DCC model, or VARIANCE=EXP are excluded from the ones that MVGARCHFore can handle.

Fat-Tails

To allow for a fat-tailed distribution, you can choose the DISTRIB=T option which uses a multivariate Student distribution. (The GED doesn’t generalize to multivariate processes.) As with univariate models, you can choose a fixed degrees of freedom with SHAPE=degrees of freedom, or, if you don’t use that option, the degrees of freedom will be estimated.
The Mean Model

As with the univariate GARCH models, the default for the mean model is the constant. In the output, these are shown as \texttt{MEAN(1)}, ... , \texttt{MEAN(n)}. Again, as in the univariate case, you can estimate the models assuming mean zero processes by adding the option \texttt{NOMEAN}.

To specify a more complicated mean model, you need to create a \texttt{MODEL} variable, generally using \texttt{GROUP}. Because the equations in the model include the information about the dependent variable, you don’t need to list dependent variables on the \texttt{GARCH} instruction. Suppose that we want AR(1) models for each of the three variables.

\begin{verbatim}
equation(constant) jpneq xjpn 1
equation(constant) fraeq xfra 1
equation(constant) suieq xsui 1
group ar1 jpneq fraeq suieq
garch(p=1,q=1,model=ar1,mv=dcc,pmethod=simplex,piter=10,$
method=bfgs,iters=200,trace)
\end{verbatim}

GARCH-M

In order to do GARCH-M with a multivariate model, you have to plan ahead a bit. On your \texttt{GARCH} instruction, you need to use the option \texttt{MVHSERIES=SYMM[SERIES]} which will save the paths of the variances (example: \texttt{MVHSERIES=HS}). Your regression equations for the means will include references to the elements of this array of series. Since those equations need to be created in advance, you need to create the \texttt{SYMM[SERIES]} first as well. Here’s an example that includes as explanatory variables in each equation all the covariances which involve the residuals from an equation.

\begin{verbatim}
dec symm[series] hhs(3,3)
do i=1,3
  do j=1,i
    set hhs(i,j) gstart gend = 0.0
  end do j
end do i
* equation jpneq xjpn
  # constant hhs(1,1) hhs(1,2) hhs(1,3)
equation fraeq xfra
  # constant hhs(2,1) hhs(2,2) hhs(2,3)
equation suieq xsui
  # constant hhs(3,1) hhs(3,2) hhs(3,3)
* group garchm jpneq fraeq suieq
garch(model=garchm,p=1,q=1,mvhseries=hhs) gstart gend
\end{verbatim}
Asymmetry

By using the ASYMMETRIC option, you can add asymmetry effects to multivariate GARCH models. The precise form of this will depend upon the variance model that you choose. For models where only the variances are directly governed by a GARCH model (DIAG, CC or DCC), each variance just has a term added as in (12).

For the other models, define

\[ v(t-1) = u(t-1) \odot I_{u>0}(u(t-1)) \]

where \( \odot \) denotes the elementwise (Hadamard) product of the vectors. This will be a copy of \( u \) with the positive elements zeroed out. With this, for the standard MV-GARCH model, the asymmetry terms in a GARCH(1,1) add the following to the formula for \( H_j(t) \):

\[ d_jv_j(t-1)v_j(t-1) \]

For the VECH, the added term is

\[ D_{vech}(v(t-1)v(t-1)') \]

and for the BEK, it's

\[ D'v(t-1)v(t-1)'D \]

Using MAXIMIZE

If you want to estimate a GARCH model which doesn't fit into the forms given here, you can use MAXIMIZE instead. The setup for this follows the same steps as it does for a univariate model, but all the steps are more complicated because of the need to deal with matrices rather than scalars. The task is usually easier if you write it specifically for two variables. If you need to allow for three or more, it makes more sense to write it very generally (as is done here) rather than expanding out terms.

This shows the code for estimating the standard GARCH(1,1) model on the three variables used in Example 12.2. For most variations, all you would typically need to do is to have an alternative FRML for HF (near the bottom of the next page). In this case, HF is easy to write as a simple FRML evaluating to a SYMMETRIC array, but remember that, if the calculation is more complicated, you can use a FUNCTION.

\[ \text{compute } gstart=2, gend=6237 \]
\[ \text{compute } n=3 \]
dec vect[series] y(n) u(n)
dec vect[frml] resid(n)
dec series[symm] h uu
*
set y(1) = xjpn
set y(2) = xfra
set y(3) = xsui

This is specific to a mean-only model. It sets up the formulas (the &i are needed in
the formula definitions when the FRML is defined in a loop), and estimates them
using NLSYSTEM. This both initializes the mean parameters, and computes the
unconditional covariance matrix. If you want more general mean equations, the
simplest way to do that would be to define each FRML separately.

dec vect b(n)
nonlin(parmset=meanparms) b
do i=1,n
   frml resid(i) = (y(&i)-b(&i))
end do i
nlsystem(parmset=meanparms,resids=u) gstart gend resid
compute rr=%sigma

The paths of the covariance matrices and UU’ are saved in the SERIES[SYMM]
names H and UU. UX and HX are used to pull in residuals and H matrices.

declare symm hx(n,n)
declare vect ux(n)

These are used to initialize pre-sample variances.

gset h  * gend = rr
gset uu  * gend = rr

This is a standard (normal) log likelihood formula for any multivariate GARCH
model. The difference among these will be in the definitions of HF and RESID.
The function %XT pulls information out of a matrix of SERIES.

declare frml[symm] hf
*
frml logl = $
   hx = hf(t) , $
   %do(i,1,n,u(i)=resid(i)) , $
   ux = %xt(u,t) , $
   h(t)=hx, uu(t)=%outerxx(ux) , $
   %logdensity(hx,ux)

Standard GARCH(1,1)

dec symm vcs(n,n) vas(n,n) vbs(n,n)
compute vcs=rr,vbs=%const(0.05),vas=%const(0.05)
nonlin(parmset=garchparsms) vcs vas vbs
frml hf = vcs+vbs.*h{1}+vas.*uu{1}
maximize(parmset=meanparms+garchparsms,pmethod=simplex,piters=10,$
   method=bfgs,iters=400) logl gstart gend
Example 12.2 Multivariate GARCH

This is the program GARCHMV.PRG. It estimates a GARCH(1,1) on a three variable system of exchange rate returns using a variety of models. It does a graph of the time-varying correlations from the simple MV-GARCH against constant correlations. The final use of the GARCH instruction estimates AR(1) mean models for each of the variables.

In addition to the lines of the program shown here, there is also code demonstrating estimation of model using MAXIMIZE rather than GARCH for both the standard model and the constant correlation.

```
open data g10xrate.xls
data(format=xls,org=columns) 1 6237 usxjpn usxfra usxsui *
set xjpn = 100.0*log(usxjpn/usxjpn{1})
set xfra = 100.0*log(usxfra/usxfra{1})
set xsui = 100.0*log(usxsui/usxsui{1})

These demonstrate the five basic MV-GARCH models. This saves the estimated covariance matrices for the simple MV-GARCH, and the covariance matrices (which will be diagonal) and residuals for the diagonal model.

garch(p=1,q=1,iter=200,hmatrices=hh) / xjpn xfra xsui
garch(p=1,q=1,mv=be,method=bfgs,iter=200,$
    pmethod=simplex,piters=20) / xjpn xfra xsui
garch(p=1,q=1,mv=diag,hmatrices=hd,rvectors=rd) / xjpn xfra xsui
garch(p=1,q=1,mv=cc) / xjpn xfra xsui
garch(p=1,q=1,mv=dcc,method=bfgs,trace) / xjpn xfra xsui

Compute the covariance matrix of the standardized residuals from the diagonal GARCH

set z1 = rd(t)(1)/sqrt(hd(t)(1,1))
set z2 = rd(t)(2)/sqrt(hd(t)(2,2))
set z3 = rd(t)(3)/sqrt(hd(t)(3,3))
vcv(matrix=cc)
# z1 z2 z3

Compute the correlations from the MV-GARCH

set rho12 = hh(t)(1,2)/sqrt(hh(t)(1,1)*hh(t)(2,2))
set rho13 = hh(t)(1,3)/sqrt(hh(t)(1,1)*hh(t)(3,3))
set rho23 = hh(t)(2,3)/sqrt(hh(t)(2,2)*hh(t)(3,3))
```
Graph the time varying correlations with a grid line at the correlation from the standardized residuals of the diagonal model.

```plaintext
graph(header="Correlation of JPN with FRA",vgrid=||cc(1,2)||)  # rho12
graph(header="Correlation of JPN with SUI",vgrid=||cc(1,3)||)  # rho13
graph(header="Correlation of FRA with SUI",vgrid=||cc(2,3)||)  # rho23
```

**AR(1) models for each dependent variable**

```plaintext
equation(constant) jpneq xjpn 1
equation(constant) fraeq xfra 1
equation(constant) suieq xsui 1
```

```plaintext
group ar1 jpneq fraeq suieq
```

```plaintext
garch(p=1,q=1,model=ar1,mv=dcc,pmethod=simplex,piter=20,$
   method=bfgs,iters=200,trace) / xjpn xfra xsui
```

**Forecasting variances**

```plaintext
garch(p=1,q=1,iters=200,hmatrices=hh,rvectors=us) / xjpn xfra xsui
```

```plaintext
@MVGarchFore(steps=100) hh us
```

Compute correlations into the forecast period, and graph them along with some of the final values from the actual sample. The GRID option puts a vertical line at the separation between actual data and forecasts.

```plaintext
set rho12 6238 6337 = hh(t)(1,2)/sqrt(hh(t)(1,1)*hh(t)(2,2))
set rho13 6238 6337 = hh(t)(1,3)/sqrt(hh(t)(1,1)*hh(t)(3,3))
set rho23 6238 6337 = hh(t)(2,3)/sqrt(hh(t)(2,2)*hh(t)(3,3))
```

```plaintext
graph(header="Correlation of JPN with FRA",vgrid=||cc(1,2)||,grid=(t==6237))  # rho12 6100 6337
graph(header="Correlation of JPN with SUI",vgrid=||cc(1,3)||,grid=(t==6237))  # rho13 6100 6337
graph(header="Correlation of FRA with SUI",vgrid=||cc(2,3)||,grid=(t==6237))  # rho23 6100 6337
```
12.2 Dynamic Linear (State Space) Models

Background

Dynamic linear models (DLM’s) form a very broad class which can describe most of the models commonly used in time series work. The basic assumptions are:

1. \[ X_t = A_t X_{t-1} + Z_t + F_t w_t, \] and
2. \[ Y_t = c_t' X_t + v_t. \]

The \( X \)'s are the states, and (1) is known as the state equation, transition equation or system equation. It describes how the vector of states evolves over time. (2) is the measurement or observation equation. It relates the observable \( Y \) to the (typically) unobservable states. \( w_t \) and \( v_t \) are shocks to the transition and measurement errors. They are usually assumed to be independent, mean zero and Normally distributed. Through most of this discussion, we’ll assume that \( F=I \). The \( F \) matrix comes in handy when you have many more states than shocks.

This is much more general than is usually required. For instance, \( Y \) is typically a single variable (the series being modelled). In most cases, the \( c \) and \( A \) matrices are time-invariant, and \( Z \) is rarely needed.

However, it is also more general than it first appears. Although (1) has the form of a one-lag vector autoregression, it can be used for a much broader range of models by state augmentation. For instance, the AR(2) process

3. \[ x_t = \varphi_1 x_{t-1} + \varphi_2 x_{t-2} + w_t, \]

can be converted to the state equation

4. \[ X_t = \begin{bmatrix} x_t \\ x_{t-1} \end{bmatrix} = \begin{bmatrix} \varphi_1 & \varphi_2 \\ 1 & 0 \end{bmatrix} X_{t-1} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} w_t. \]

Non-linear dynamic models can often be approximated in a linear form, and models with non-Gaussian errors can be approximated by ones with the simpler Gaussian errors.

A DLM for a series can be built directly from a reduced form description of the process, or it can be constructed in parts. In general, there are many equivalent representations for a model. Consider, for instance,

5. \[ y_t = 2 y_{t-1} - y_{t-2} + \varepsilon_t. \] One DLM for this is

6. \[ X_t = \begin{bmatrix} y_t \\ y_{t-1} \end{bmatrix}, \quad A = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}, \quad w_t = \begin{bmatrix} \varepsilon_t \\ 0 \end{bmatrix}, \quad c' = [1 \ 0], \quad v_t = 0 \]
Another can be derived from
\[ T_t = T_{t-1} + \varepsilon_t, \quad y_t = y_{t-1} + T_t \]
where \( T_t \) is a “local” trend rate. The DLM which describes this is
\[
\begin{align*}
X_t &= \begin{bmatrix} y_t \\ T_t \end{bmatrix}, \\
A &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \\
w_t &= \begin{bmatrix} \varepsilon_t \\ \varepsilon_t \end{bmatrix}, \\
c' &= \begin{bmatrix} 1 & 0 \end{bmatrix}, \\
v_t &= 0
\end{align*}
\]
Another example is the first order moving average process
\[ y_t = \varepsilon_t + \theta \varepsilon_{t-1}. \]
Two DLM’s for this are
\[
\begin{align*}
X_t &= \begin{bmatrix} \varepsilon_t \\ \varepsilon_{t-1} \end{bmatrix}, \\
A &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \\
w_t &= \begin{bmatrix} \varepsilon_t \\ 0 \end{bmatrix}, \\
c' &= \begin{bmatrix} 1 & \theta \end{bmatrix}, \\
v_t &= 0, \quad \text{and}
\end{align*}
\]
\[
\begin{align*}
X_t &= \begin{bmatrix} y_t \\ \varepsilon_t \end{bmatrix}, \\
A &= \begin{bmatrix} 0 & \theta \\ 0 & 0 \end{bmatrix}, \\
w_t &= \begin{bmatrix} \varepsilon_t \\ \varepsilon_t \end{bmatrix}, \\
c' &= \begin{bmatrix} 1 & 0 \end{bmatrix}, \\
v_t &= 0
\end{align*}
\]

**The Methods**

The instruction **DLM** analyzes these models. It offers a choice of five methods, controlled by the **TYPE** option. To simplify notation, let \( X_{t|s} \) and \( \Sigma_{t|s} \) represent the expected value and the variance of \( X_t \) given information through time \( s \).

1. **TYPE=FILTER** computes \( X_{t|t} \), the expectation given past and present.
2. **TYPE=SMOOTH** computes \( X_{t|T} \), the expectation given all data.
3. **TYPE=SIMULATE** simulates values for \( X_t \) and \( Y_t \) assuming Normal distributions for the shocks.
4. **TYPE=CSIMULATE** simulates values for \( X_t \) conditional on the observed \( Y_t \) values.
5. **TYPE=CONTROL** solves a linear-quadratic control problem.

**TYPE=CONTROL** requires a slightly different form of state equation:
\[
X_t = A_t X_{t-1} + B_t U_t + F_t w_t
\]
where \( U_t \) are the controls. Optimal control solves
\[
\min_{\{U_t\}} E \left( X'_0 Q_0 X_0 + \sum_{t=1}^{T} \{X'_t Q_t X_t + U'_t R_t U_t\} \right)
\]
Chapter 12: Special Models

Getting Information into DLM

DLM has quite a few inputs: at a minimum, the A, c and y matrices, the variances for v and w, and there are often others. Some of those (such as the y) will almost certainly be changing with time, while others will be fixed matrices, and some, while they are fixed over time, might depend upon parameters that you’re trying to estimate, and thus won’t be known in advance. These all go into DLM using options with names based upon the notation given in (1) and (2); the A matrix is input with the A option, c with the C option, etc. The options for the two variance matrices are SV and SW. In general, you can give the values for these in the most natural way. For instance, if you want the series LOGGDP to be your Y, the option would be Y=LOGGDP. If c is the vector [1,0,0], the option could read C=| 1.0, 0.0, 0.0 |. If you have just one observable, the c matrix can be either or row or column vector.

DLM will automatically detect whether the input information is time-varying so you don’t have to take special steps as you might with other state-space software.

Getting Information from DLM

DLM has many uses and generates quite a bit of information. You can, for instance, get the estimated states and their variances, predicted observables and prediction errors and variances, state shock estimates and variances, and more. Most of this is retrieved as one of a SERIES of matrices. For instance, the estimated states are a SERIES[VECTOR]—at each time period analyzed by DLM there’s a complete VECTOR. In most cases, you will be interested in the time path of only one or two components of this. You can pull items out easily with SET instructions. For instance, if STATES is the name of your output, the following will pull the first component out of this:

set xstate = states(t)(1)

Note that you have to use the two sets of subscripts: the (t) is used to pick the entry out of the SERIES of VECTORS and the (1) pulls the first element out of that vector. The variances of the states (and most other variances) are produced as a SERIES of SYMMETRICS. Again, SET is generally the easiest way to pull information out of this. For instance, if SIGMAS is the name assigned to the series of covariance matrices, to get the standard deviation of the first state, do

set xstddev = sqrt(sigmas(t)(1,1))
Kalman Filter
The Kalman filter is used to solve the filtering problem, and is also used as a step in the other methods. Let $\mathbf{M}_t = \text{var}(w_t)$ and $\mathbf{N}_t = \text{var}(v_t)$. The Kalman filter is the following four equations:

\begin{align*}
(14) & \quad \mathbf{X}_{t|t-1} = \mathbf{A}_t \mathbf{X}_{t-1|t-1} + \mathbf{Z}_t \\
(15) & \quad \mathbf{\Sigma}_{t|t-1} = \mathbf{A}_t \mathbf{\Sigma}_{t-1|t-1} \mathbf{A}_t' + \mathbf{M}_t \\
(16) & \quad \mathbf{\Sigma}_{\ell t} = \mathbf{\Sigma}_{t|t-1} - \mathbf{c}_t (\mathbf{c}'_t \mathbf{\Sigma}_{t|t-1} \mathbf{c}_t + \mathbf{N}_t)^{-1} \mathbf{c}'_t \mathbf{\Sigma}_{t|t-1} \\
(17) & \quad \mathbf{X}_{\ell t} = \mathbf{X}_{t|t-1} + \mathbf{c}_t (\mathbf{c}'_t \mathbf{\Sigma}_{t|t-1} \mathbf{c}_t + \mathbf{N}_t)^{-1} (\mathbf{Y}_t - \mathbf{c}'_t \mathbf{X}_{t|t-1})
\end{align*}

The first two of these apply whether or not data are observable at $t$. The last two are only used if $\mathbf{Y}_t$ exists. If it doesn’t, $\mathbf{X}_{\ell t} = \mathbf{X}_{t|t-1}$ and $\mathbf{\Sigma}_{\ell t} = \mathbf{\Sigma}_{t|t-1}$.

Kalman Smoother
The Kalman smoother starts by Kalman filtering through the entire data set. At the end of the data set, the Kalman filter has computed $\mathbf{X}_{T|T}$ and $\mathbf{\Sigma}_{T|T}$. The smoothed estimates of the states and their variances are then computed by a backwards recursion as described in Durbin and Koopman (2002). The smoothed states are the ones that are most often used when you’re trying to extract components, like seasonals or trend cycles.

Simulations
Standard simulations draw random (Normal) shocks for the $\mathbf{w}$ and $\mathbf{v}$ to produce simulated values of the states and $\mathbf{y}$. Conditional simulations draw states and disturbances from the (reduced rank) distribution which produces the observed $\mathbf{Y}$. This is very useful as a step in Bayesian analysis of dynamic models. See Durbin and Koopman (2002) for more information.

Missing Values
A missing data point can be represented by either a $\mathbf{c}_t$ or $\mathbf{Y}_t$ which has a missing value. You can also use a SMPL option to skip data points. Note that the missing time periods are still considered part of the overall sample. The Kalman filter and smoother will estimate the state vector at those time periods and optimal control will generate a control value for them.

If $\mathbf{Y}$ has more than one component, it’s possible that there will be data points where some are observed and some are not. Those observations are handled (within DLM) by reducing the $\mathbf{Y}$, $\mathbf{c}$ and $\mathbf{\Sigma}_c$ matrices to include only the elements that correspond to the $\mathbf{Y}$ values that are observed.
Chapter 12: Special Models

Caution

Almost every book which covers this topic uses its own collection of names for the components. This is a minor inconvenience. More significant is that some texts use different time subscripts in the state equation, something like:

\[(18) \quad \xi_{t+1} = A_t \xi_t + w_t \]

The two systems are related by

\[(19) \quad \xi_t = X_{t-1} \]

The Kalman filter will produce slight timing differences in the state vector. This is usually a minor problem, since the state estimates from Kalman filtering are themselves of little interest. The two systems will give the same results for the likelihood (and thus for parameter estimates), for Kalman smoothing and for simulations or forecasts of the observables.

In trying to match up results done under the two systems, one thing to note is that DLM wants the pre-sample information to be \(X_{00}, \Sigma_{00}\) while the alternative specification needs \(X_{10}, \Sigma_{10}\) (in our notation). In general, these \(aren’t\) the same: see equations (14) and (15). Use the option PRESAMPLE=X1 to have your input initial values interpreted as the latter.

Initial States

With the (possible) exception of optimal control, one other piece of information is needed for the calculations: the initial distribution of the states. This requires a mean and a variance. If the state vector has a stationary distribution, it’s possible to use the unconditional distribution. For instance, if we take expected values in the (time-invariant) transition equation

\[(20) \quad X_t = AX_{t-1} + Z + w_t \]

we get

\[(21) \quad EX = AX + Z, or \quad EX = (I - A)^{-1}Z \]

(if there is no Z component, the presample mean will just be zero, which is the default). If we take variances (assuming the variance of \(w\) is constant), we get the equation:

\[(22) \quad \Sigma_X = A \Sigma_X A' + \Sigma_{w} \]

You can compute the solution \(\Sigma_X\) to this yourself using the RATS function \%PSDINIT(A, \Sigma_{w})\). DLM also has the option PRESAMPLE=ERGODIC which will compute solutions for both the presample mean and variance.
More commonly, however, even a model with time-invariant matrices isn’t stationary so there is no stationary distribution for the initial state. For instance, if $A$ has a unit eigenvalue, the inverse in (21) doesn’t exist and there is no solution to (22). This has usually been handled in one of three ways:

1. The early observations are ignored, until a sufficient number have been processed to resolve the non-stationarity. If you’re trying to estimate the free parameters in a DLM, you can do this with the option `CONDITION=initial periods`.

2. Big finite variances are used for the non-stationary components. The problem with this is that for a single basic model, “big” may need to vary considerably from one application to the next. If the variances are too large, you may lose all precision on the stationary part of the distribution.

3. The problem is expanded into a full sample which can be computed with the help of a $T \times T$ inverse (thus not using DLM).

RATS offers a fourth alternative: the exact diffuse calculations described in Koopman (1997) and Durbin and Koopman (2002). In this, the prior variance has two parts: finite and infinite (diffuse). We describe how to set this up below.

**Exact Diffuse Initial Conditions**

To implement the exact diffuse Kalman filter, include the option `EXACT`. With each observation, it does a two-part Kalman filter, which typically will reduce the rank of the diffuse part by one (or the number of observables). Once the rank of the diffuse part hits 0, the standard Kalman filter takes over.

Without any further options, this takes the initial state as mean 0 with variance $kI$ where $k$ is “infinite.” A more complicated situation is where the state space model is partly non-stationary and partly stationary. The stationary part is a combination of mean and covariance matrix as with a standard stationary model, while the diffuse part has just a covariance matrix (with mean zero). To input these directly, you can use the options `X0` and `SX0` for the stationary mean and variance and `SH0` for the diffuse covariance matrix. However, unless the state representation is shrewdly designed (to, for instance, partition the states into non-stationary and stationary), figuring out what these matrices are can be rather messy. DLM provides the $G$ option to do this calculation. With this, you provide an $r \times n$ matrix $G$ which transforms the states to stationarity. DLM modifies the state vector to

$$
Y_t' = LY_t, L = \begin{bmatrix} G \\ H \end{bmatrix}, \text{ where } GH' = 0.
$$

This can always be constructed if $G$ is full row rank. The new state equation is

$$
Y_t^* = LAL^{-1}Y_{t-1}^* + LZ + Lw
$$
If the $G$ matrix has been chosen correctly, $LAL^{-1}$ will be block lower triangular where the top left $r \times r$ corner has no unit roots. The stationary mean and variance for that subvector of $Y_t^*$ can be computed as described before. This is transformed back to the $Y$ space producing a reduced rank covariance matrix. The diffuse part of the variance can be taken as

\[(25) \quad I - G'(GG)^{-1}G\]

**Optimal Control**

The linear-quadratic optimal control problem is solved by recursion. See, for instance, Bertsekas (1976). Suppose the state vector going into period $T$ is $X_{T-1}$. The only remaining cost to be incurred is

\[(26) \quad E(X'_T Q_T X_T + U'_T R_T U_T)\]

Substitute $X_T = A_T X_{T-1} + B_T U_T + w_T$ and take expected values. Since $w_T$ has expected value 0 (conditional on the past shocks to the system), all terms involving it drop out except $E(w'_T Q_T w'_T)$. But that term doesn't interact with $U_T$, and so can be ignored as far as the optimization problem is concerned. This is known as the \textit{certainty-equivalence principle}: the backwards solution can be done as if there is no shock in the transition equation.

The solution to minimizing the remaining cost given $X_{T-1}$ is

\[(27) \quad U_T = \left\{ -(B'_T Q_T B_T + R_T)^{-1} B'_T Q_T A_T \right\} X_{T-1}\]

With $U_T$ now solved out as a linear function of $X_{T-1}$, the time $T$ terms of the cost (26) can be written as a quadratic form in $X_{T-1}$. The problem from $T-1$ on can be solved using the same method, producing $U_{T-1}$ as a linear function of $X_{T-2}$. The control laws are generated recursively until the start of the sample. At that point, we will have a \textit{cost-to-go functional} which gives the total cost as a quadratic form in $X_0$. The Kalman filter is then applied to generate the estimated state vectors. The transition equation for the Kalman filter will be

\[(28) \quad X_t = A_t X_{t-1} + B_t L_t X_{t-1} + w_t\]

where $L_t$ are the matrices in the control laws

\[(29) \quad U_t = L_t X_{t-1}\]
Estimating Parameters

The moving average model (9) depends upon the parameter $\theta$. If $\theta$ is known, DLM can be used for prediction or control. If $\theta$ isn’t known, DLM can be used to estimate it. Assuming normal disturbances, the Kalman filter generates the following distribution for the observables:

\[ Y_t | Y_1, ..., Y_{t-1} \sim N(c'_t X_{\theta t-1}, c'_t \Sigma_{\theta t-1} c_t + N_t) \]

The log likelihood function for the full sample is thus (omitting additive constants, although those will be included in the values reported by DLM)

\[ -(1/2) \sum_t \log |c'_t \Sigma_{\theta t-1} c_t + N_t| - (1/2) \sum_t (Y_t - c'_t X_{\theta t-1})'(c'_t \Sigma_{\theta t-1} c_t + N_t)^{-1} (Y_t - c'_t X_{\theta t-1}) \]

This is the objective function used for Kalman filtering (and smoothing). Whether you estimate parameters or not, DLM will set the variables %LOGL and %FUNCVAL to this or a similar value as discussed later. The objective function for TYPE=CONTROL is the recursively generated cost functional—DLM will put that value into %FUNCVAL.

To estimate a model, use NONLIN to set up a PARMSET with the free parameters that you want to estimate. You have a choice of four estimation methods: SIMPLEX, GENETIC, BFGS and GAUSSNEWTON. The first two are described in Section 7.3 and the other two in 7.2. BFGS and GAUSSNEWTON are the only ones which can compute standard errors for the coefficients. However, they can only do that correctly when applied to TYPE=FILTER or TYPE=SMOOTH. The objective function in TYPE=CONTROL isn’t likelihood-based, so the second derivative estimates don’t help form an estimator for the covariance matrix.

Variance Scale Factors

There are three variance terms in the model: $\Sigma_w$, $\Sigma_v$ and $\Sigma_{00}$. By default, DLM calculates the likelihood assuming that these are known (or being conditioned upon). However, as with linear regressions, it’s more common that the variance in the measurement equation isn’t known. The simplest way to handle this is to assume that all three variance (matrices) are known up to a single (unknown) constant multiplier: this is often a reasonable assumption since, if $c$ is fixed, the scale of the $X$’s will have to adjust to match the scale of $Y$ anyway.

There are two options available for estimating this multiplier: it can be concentrated out, or it can be given a prior distribution. This is controlled by the option VARIANCE. This scale factor is usually, though not always, the unknown variance in the measurement equation, hence the option name. VARIANCE=CONCENTRATED (the older option SCALE can be used in place of it) requests that this value be concentrated out, while VARIANCE=CHISQUARED gives it an (inverse) chi-squared prior. If, as is typical, this variance is the variance in the measurement equation, you use the option SV=1. Note well: if you choose either one of these, all the estimated variance matrices will
need to be scaled by the estimated variance. The default for the VARIANCE option is VARIANCE=KNOWN, which means that all the variances are to be used as given.

With VARIANCE=CONCENTRATED, if we write $N_t = \lambda n_t$, the log likelihood element can be rewritten as

$$-\frac{1}{2} \sum_i \log |c_i' (\lambda \Sigma_{ij-1}) c_i + \lambda n_i| - \frac{1}{2} \sum_i (Y_i - c_i' X_{ij-1})' (c_i' (\lambda \Sigma_{ij-1}) c_i + \lambda n_i)^{-1} (Y_i - c_i' X_{ij-1})$$

where $\lambda$ is that unknown scale factor. $\lambda$ can be isolated to give

$$-\frac{\text{rank}}{2} \log \lambda - \frac{1}{2\lambda} \sum_i (Y_i - c_i' X_{ij-1})' (c_i' \Sigma_{ij-1} c_i + n_i)^{-1} (Y_i - c_i' X_{ij-1}) - \text{terms without } \lambda$$

where rank is the sum of the ranks of the $c_i' \Sigma_{ij-1} c_i + n_i$. This will just be the number of observations if you have one observable and no missing values. The maximizing value for $\lambda$ is

$$\hat{\lambda} = \frac{1}{\text{rank}} \sum_i (Y_i - c_i' X_{ij-1})' (c_i' \Sigma_{ij-1} c_i + n_i)^{-1} (Y_i - c_i' X_{ij-1})$$

DLM sets the variable %VARIANCE to this value. Substituting this in gives the concentrated likelihood (again, omitting additive constants)

$$-\frac{\text{rank}}{2} \log \hat{\lambda} - \frac{\text{rank}}{2} - \frac{1}{2} \sum_i \log |c_i' \Sigma_{ij-1} c_i + n_i|$$

which is what DLM will maximize if you are estimating parameters. You should get the same results by estimating all the variances directly. However, the likelihood function is often very flat in the common scale factor, so the full set of variances may be close to being perfectly correlated. If this happens, you can run into problems with the optimization routine stalling out short of the maximum.

With VARIANCE=CHISQUARED, you need to provide two pieces of information in order to complete the prior: the “mean” (PSSCALE=prior scale) and the degrees of freedom (PDF=prior degrees of freedom). When you Kalman filter through the data, the estimates of the variance will also be updated. If you need to track that, you can use the options SIGHISTORY=series of estimated variances and DFHISTORY=series of degrees of freedom of variance estimates. When you use this option, the conditional distribution of $Y$ is a $t$, not a Normal, and the likelihood is computed using the $t$ density. See West and Harrison (1997) if you’re interested in this approach.
Conditioning on Early Values

If you have a non-stationary model, the initial state vector is usually represented by a diffuse prior. If, instead of using the exact diffuse prior, you use large finite values, the likelihood could be dominated by the first few observations, where the variance is high. If you do this, you can use the option \texttt{CONDITION=initial periods} to indicate the number of observations which should \textit{not} be incorporated into the likelihood function. You still include these in the estimation range; the likelihood actually used will just condition on them. Typically, you would condition on the number of states, but it might be a smaller number if the initial state vector is non-stationary only in a reduced number of dimensions.

This estimates the model shown below. This concentrates out the variance of $\varepsilon$, so $\Sigma_{0|0} = I$ (this is a fully stationary model) and $\Sigma_{\nu}$ has a 1 in the top left corner.

$$
\begin{align*}
X_t &= \begin{bmatrix} \varepsilon_t \\ \varepsilon_{t-1} \end{bmatrix}, \\
A &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \\
w_t &= \begin{bmatrix} \varepsilon_t \\ 0 \end{bmatrix}, \\
c' &= \begin{bmatrix} 1 & \theta \end{bmatrix}, \\
v_t &= \begin{bmatrix} 0 \\ \nu_t \end{bmatrix}
\end{align*}
$$

\begin{verbatim}
nonlin theta
compute [rect] a=||0.0,0.0|1.0,0.0||
compute [symm] sw=||1.0|0.0,0.0||
*
compute theta=0.0
dlm(method=bfgs,a=a,c=||1.0,theta||,y=ydiff,sx0=%identity(2),$ 
   sw=sw,var=concentrate) 1959:2 2001:1
\end{verbatim}
Example 12.3 Hodrick-Prescott Filter

The Hodrick-Prescott (1980) filter has been used extensively to extract a growth component from macroeconomic series, leaving the cyclical element. This is available in RATS using the \texttt{FILTER} instruction with the option \texttt{TYPE=HP}. This shows how it can be computed using exact initial Kalman smoothing.

The series is modelled as

\begin{align*}
(1) \quad y_t &= g_t + u_t, \text{ where the unobservable growth component } g \\
(2) \quad g_t &= 2g_{t-1} - g_{t-2} + \varepsilon_t
\end{align*}

This is almost identical to the model at the beginning of the section, except for a “measurement error” in the form of the irregular component. A DLM for this is

\begin{align*}
(3) \quad X_t &= \begin{bmatrix} g_t \\ g_{t-1} \end{bmatrix}, \quad A = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}, \quad \varepsilon_t = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad w_t = \varepsilon_t
\end{align*}

HP’s recommendation is to use Kalman smoothing. This is an example of a non-stationary state space model. It’s a particularly simple case for the exact initial Kalman smoother because the initial state is fully non-stationary (that is, no non-trivial linear combination of the initial state vector is stationary), so all that is needed is to use the \texttt{EXACT} option. The \texttt{X0} and \texttt{SX0} options aren’t necessary since there is no finite part of the initial distribution.

Note that the design of the HP filter pegs a value for the ratio between the variances of $u_t$ and $\varepsilon_t$ (1600 being the common choice for quarterly data). If this were an attempt to model the complete dynamics of $y$, we would set this up to estimate the (relative) variances. That, however, is not the point. The HP filter is not “modelling” $y$; it’s trying to extract $g$, and $g$ is supposed to be a smooth curve. By making the variance in (1) high relative to that in (2), the short-term movements will be kept out of the growth curve.

The model (2) is a special case of a local trend model. There are two procedures for setting up basic DLMs which are often used for decomposing time series data. \texttt{@LOCALDLM} sets up a local level or local trend model. \texttt{@SEASONALDLM} sets up a seasonal model. These produce the appropriate $A$, $C$ and $SW$ matrices. If you have a model with both trend and seasonal components, you can “add” the models. This requires a diagonal concatenation of the $A$ and $SW$ matrices and a vertical concatenation of $C$ matrix.

\begin{verbatim}
@LocalDLM(type=trend) at ct
@SeasonalDLM(type=additive) as cs
compute a=at~\as
compute c=ct~~cs
\end{verbatim}
This example is provided in the file `HPFILTER.PRG`.

```plaintext
cal(q) 1959:1
open data haversample.rat
data(format=rats) 1959:1 2006:4 gdph
log gdph / lgdp

This uses the built-in HP filter in the FILTER instruction

filter(type=hp,tuning=1600.0) lgdp / hpfilt

compute lambda = 1600.0

As relative variances are all that matter, SW is pegged at 1 with SV being LAMBDA.

compute [rect] a = ||2.0,-1.0|1.0,0.0||
compute [vect] c = ||1.0,0.0||
compute [rect] d = ||1.0|0.0||

* dlm(a=a,c=c,y=lgdp,sv=lambda,d=d,sw=1.0,exact,type=smooth)  $
  1959:1 2006:4 estates$
set hpsmooth = estates(t)(1)

graph(window="HP filter") 2
  # lgdp
  # hpsmooth

Direct solution of the system of equations from the first order conditions for the minimization problem. Most of the matrix is set using the FMATRIX instruction, as, except for the first and last few rows, the rows are 4th difference operators.

compute nobs=2006:4-1958:4
dec rect hp(nobs+2,nobs+2)
fmatrix hp 1 1
  # -2 -1 0 1 2
  # 1.0 -4.0 6.0 -4.0 1.0
compute hp(1,1)=1.0,hp(1,2)=-2.0,hp(1,3)=1.0
compute hp(2,1)=-2.0,hp(2,2)=5.0,hp(2,3)=-4.0,hp(2,4)=1.0
compute hp(nobs+2,nobs+2)=1.0,hp(nobs+2,nobs+1)=-2.0,$
  hp(nobs+2,nobs)=1.0
compute hp(nobs+1,nobs+2)=-2.0,hp(nobs+1,nobs+1)=5.0,$
  hp(nobs+1,nobs)=-4.0,hp(nobs+1,nobs-1)=1.0
dec vect yy(nobs+2) dd(nobs+2)
ewise yy(i)=%if(i<=nobs,lgdp(nobs+1-i),0.0)
ewise dd(i)=%if(i<=nobs,1.0,0.0)
compute gg=%solve(lambda*hp+%diag(dd),yy)

* set hpexact 1959:1 2006:4 = gg(nobs+1-t)```
Example 12.4 Estimating a State-Space Model

A direct competitor with the GARCH model for modeling the variances of financial time series is the stochastic volatility model. In its simplest form, this is represented by

(1) \[ y_t = \varepsilon_t \sqrt{h_t} \]

(2) \[ \log h_t = \gamma + \varphi \log h_{t-1} + w_t \]

where \( \varepsilon_t \sim N(0,1), w_t \sim N(0,\sigma^2) \). Estimating the free parameters of this model is somewhat tricky because the variances \( h_t \) aren’t observable and are subject to the random shock \( w_t \). (2) is a perfectly good state equation for \( \log h_t \), but (1) isn’t in the proper form for a measurement equation. However, if we square and log (1), we get

(3) \[ \log \gamma^2 = 1 \times \log h_t + \log \varepsilon_t^2 \]

which would be usable, except that \( \log \varepsilon_t^2 \) isn’t Normally distributed. It is, however, a distribution with a known mean and variance, so the model’s parameters can be estimated by quasi-maximum likelihood using DLM. This is example SV.PRG.

open data xrates.xls
data(format=xls,org=columns) 1 946 usxuk usxger usxjpn usxsui

\[ MEANX2=\text{mean of log chi-square}, \quad VARX2=\text{variance of log chi-square} \]

compute meanx2=%digamma(0.5)-log(0.5)
compute varx2 =%trigamma(0.5)

Gamma and Phi are very highly correlated when Phi is near 1, which makes estimation by general hill-climbing procedures somewhat tricky. Instead, we reparameterize this to use \( \text{Gammax}=\Gamma(1-\Phi) \) in place of \( \Gamma \).

set dlogp = log(usxuk{0}/usxuk{1})
diff(center) dlogp / demean
nonlin phi sw gammax
set ysq = log(demean**2)-meanx2

Get initial guess values from ARMA(1,1) model

boxjenk(ar=1,ma=1,constant,noprint) ysq

PHI is taken directly as the AR(1) coefficient The variance SW is backed out by matching first order autocorrelations in the MA term. GAMMA is chosen to reproduce the mean of the YSQ series

compute phi=%beta(2),$
sw=-phi*varx2*(1+%beta(3)**2)/%beta(3)-(1+phi**2)*varx2
compute sw=%if(sw<0,.1,sw)
compute gammax=%mean

Estimate the unconstrained model

dlm(method=bfgs,sw=sw,sv=varx2,y=ysq,type=filter,c=1.0, 
$ sx0=sw/(1-phi**2),x0=gammax,a=phi,z=gammax*(1-phi)) 2 * states
12.3 Markov Switching Models

Markov switching models for time series have become popular lately, largely as a result of the work of James Hamilton (see Chapter 22 of Hamilton (1994)).

To start, let’s look at a simple Markov chain. Suppose that there are two regimes. The probability of moving from state 1 to state 2 is $P_{12}$, and for moving from state 2 to state 1 is $P_{21}$, both unknown. In this simple case, the regime is known at each time period. Let the state be shown by the dummy variable REGIME which will be 1 for state 1 and 0 for state 2.

This is a simple enough situation that the maximum likelihood estimate of $P_{12}$ is just the fraction of the times that the system is in state 1 and moves to state 2. However, we’ll illustrate how to set this up with MAXIMIZE, as more complex examples require only minor modifications.

```plaintext
nonlin p12 p21
frml markov = $
    
    rp1 = p21*(1-regime{1})+(1-p12)*regime{1},$
    rp2 = (1-p21)*(1-regime{1})+p12*regime{1},$
    pstar = rp1/(rp1+rp2),$
    %if(regime,log(pstar),log(1-pstar))
maximize markov
```

$RP_1$ and $RP_2$ are the relative probabilities of states 1 and 2 at a given time period. $PSTAR$ converts these into the actual probability of state 1. In this case, it is trivially either $P_{21}$ or $1-P_{12}$.

Now we will look at a situation where the regime is independent across time and unknown, but where an observable variable $y$ is governed by different processes in the two regimes:

In regime 1, $y_t = a_{10} + a_{11}x_{1t} + u_{1t}$

In regime 2, $y_t = a_{20} + a_{21}x_{2t} + u_{2t}$

Assume that, conditional on $x_1$ and $x_2$, $u_1$ and $u_2$ are distributed independently both contemporaneously and across time.

Let $p$ be the (unknown) unconditional probability of the process being in state 1 at any time period. The likelihood comes from the formula

$$f(y| \text{parameters}) = p f(y| \text{parameters}, \text{state 1}) + (1 - p) f(y| \text{parameters}, \text{state 2})$$

Bayes’ formula gives the post-data probability of being in state 1 as

$$\frac{p f(y| \text{parameters}, \text{state 1})}{f(y| \text{parameters})}$$
A (set of) formulas to estimate this model is:

```plaintext
nonlin p a10 a11 a20 a21 sigma
frml reg1 = y-a10-a11*x1
frml reg2 = y-a20-a21*x2
frml mixing = $
  f1=%density(reg1{0}/sigma)/sigma, $
  f2=%density(reg2{0}/sigma)/sigma, $
  rp1=f1*p, rp2=f2*(1-p), $
  pstar=rp1/(rp1+rp2), log(rp1+rp2)
```

We now combine the two types of models. The state may shift from period to period following the Markov model from the first example. If \( p^* \) is our estimate of the probability of being in state 1 based upon data through \( t \), then the (pre-data) probability of state 1 in period \( t+1 \) is

\[
p_{21}(1-p^*) + (1-p_{12})p^*
\]

The set up for this model is:

```plaintext
nonlin p12 p21 a01 a02 a11 a12 sigma
frml markov = $
  f1 = %density(reg1{0}/sigma)/sigma, $
  f2 = %density(reg2{0}/sigma)/sigma, $
  rp1 = f1*(p21*(1-pstar{1}) + (1-p12)*pstar{1}), $
  rp2 = f2*((1-p21)*(1-pstar{1})+p12*pstar{1}), $
  pstar = rp1/(rp1+rp2), $ \log(rp1+rp2)
```

There are several technical problems that you might face in using these types of models:

- If one of the states is fairly rare, the parameters in its regime equation may be poorly estimated. Because the regime equation is also important in computing the probabilities of each state, the estimates of \( p^* \) may shift quite a bit from one iteration to the next, causing very slow convergence.

- The global identification of the regimes may be fairly weak. For instance, if the explanatory variables \( X_1 \) and \( X_2 \) are the same (or are absent), the two states could be switched without affecting the model. This is not a severe problem if you understand it, since it means simply that there are two global optima. However, if there isn't a clear enough distinction between the two regimes, the estimates could collapse to show (in effect) a single regime. Trying to compute transition probabilities between two states which are indistinguishable is a fool's errand, but one that MAXIMIZE will still attempt.

In short, use the TRACE option on MAXIMIZE and take a very careful look at your results before writing a thirty page paper about them.
MARKOV.SRC functions

The MARKOV formula does a lot of work: it evaluates the density functions under the two states, re-estimates the value of $p^*$ and returns the log likelihood. And it’s specifically written for a two state model. If you move to three states, the number of free parameters in the transition matrix goes up to six, and with four, to twelve. To allow for more than two states, it makes sense to switch to matrix operations and break the calculations into manageable pieces. We now describe a set of three functions that are on MARKOV.SRC.

The representation for the transition matrix which simplifies the calculations most is an $(n-1) \times n$ matrix, where $P(i,j)$ is the probability of moving to state $i$ from state $j$. The missing $P(n,j)$ is just one minus the sum of the other elements in column $j$.

The function %MCSTATE($P$, $PSTAR$) takes such a transition matrix, combined with the current estimate of $p^*$, and returns the pre-data revision to the probabilities. Post-multiplying $P$ by $p^*$ gives the probabilities of states 1,...,$n-1$; and the probability of state $n$ is just one minus the sum of the other probabilities.

function %mcstate p pstar
type vect %mcstate pstar
type rect p
local vect pstub
local integer i
compute pstub=p*pstar
dim %mcstate(%cols(p))
ewise %mcstate(i)=%if(i<=%rows(p),pstub(i),1-%sum(pstub))
end

%MSUPDATE($F$, $PSTAR$, $FPT$) returns the updated state probabilities given the vector of likelihoods (levels, not logs) of the states in $F$ and the prior probabilities of the states in the vector $PSTAR$. $FPT$ is a REAL returned by the function which gives $F \cdot P^*$. If $F$ is a vector of likelihoods given the states, $FPT$ will be the overall likelihood for this entry.

function %msupdate f pstar fpt
type vect %msupdate f pstar
type real *fpt
local vector fp
compute fp=f.*pstar
compute fpt=%sum(fp)

If the $F$'s come in zero, keep the old PSTAR. The parameters will get rejected anyway since the LOG($FPT$) will be NA.

if %minvalue(pstar)<0.00
   compute %msupdate=pstar,fpt=%na
else
   if fpt>0 ; compute %msupdate=fp/fpt
   else ; compute %msupdate=pstar
end
%MCERGODIC takes the transition probability matrix (parameterized as described above) and returns as a VECTOR the ergodic (stationary) distributions of the states. In most applications, this will be the appropriate initialization for $p^*$. To feed this into the PSTAR vector, use the option STARTUP=(PSTAR=%MCERGODIC(p)) on the MAXIMIZE which estimates the model. This gives a calculation which is done before each function evaluation.

```
function %mcergodic p  
  type vect %mcergodic  
  type rect p  
  *  
  local rect a  
  local integer i j n  
  *  
  compute n=%cols(p)  
  dim a(n+1,n)  
  ewise a(i,j)=%if(i>n,1,(i==j)-%if(i<n,p(i,j),1-%sum(%xcol(p,j))))  
  compute %mcergodic=%xcol(inv(%innerxx(a))*tr(a),n+1)  
end
```

To complete the model, you need to be able to generate the vector of likelihoods needed by %MSUPDATE. The following could be used to estimate the final model described above. Our suggestion is that you create a FUNCTION to do this calculation. That will make it easier to make changes to the basic model. The return value of the FUNCTION should be a VECTOR with size equal to the number of states. Remember that these are the likelihoods, not the log likelihoods. If it's more convenient doing the calculation in log form, make sure that you exp the results before you return.

```
function SimpleRegimeF time  
  type vector SimpleRegimeF  
  type integer time  
  dim SimpleRegimeF(2)  
  compute f(1)=%density(reg1(time)/sigma)/sigma)  
  compute f(2)=%density(reg2(time)/sigma)/sigma)  
end
```

```
compute n=2  
dec rect p(n-1,n)  
dec vect f(n) pstar(n)  
nonlin p a01 a02 a11 a12 sigma  
frml markov = f=SimpleRegimeF(t),$  
    (pstar=%msupdate(f,%mcstate(p,pstar),fpt),log(fpt))  
maximize(startup=(pstar=%mcergodic(p))) markov start end
```

The final line in the definition of the MARKOV formula will look the same regardless of the structure of the model. It's the definition of the “F” vector that changes from model to model.
While these three functions are often useful, the calculations for some models don’t easily fit into this framework. For instance, Example 12.5 has a large set of states with a very sparse (that is, mostly zero) transition matrix, so it does the same types of calculations but uses specialized methods to avoid a lot of multiplies by zero.

State Probability Smoothing

The sequence of \( P_{STAR} \) vectors gives the “filtered” probabilities of the states, that is, the probability that the data are in each state at time \( t \) given information only through time \( t \). The smoothed estimates of the probabilities are the computed using all the data. Computing these requires keeping the entire history of the \( P_{STAR} \) (the output from \%MSUPDATE), plus the entire history of the pre-data adjusted probabilities (output from \%MCSTATE), then doing an extra sweep through the data at the end. See Hamilton (1994) for technical details. To do this, add the following to the set up:

```plaintext
dec series[vect] pt_t pt_t1 psmooth
iset pt_t = %zeros(nstates,1)
iset pt_t1 = %zeros(nstates,1)
iset psmooth = %zeros(nstates,1)
```

adjust the MARKOV formula to

```plaintext
frml markov = f=SimpleRegimeF(t),$
pt_t1=%mcstate(p,pstar),pt_t=pstar=%msupdate(f,pt_t1,fpt),log(fpt)
```

and add

```plaintext
@%mssmooth p pt_t pt_t1 psmooth
```

after the model has been estimated. \%MSSMOOTH is also on MARKOV.SRC.

Models with Lags

The model which we’ve been using so far has an observation equation which depends upon the current state, not lagged states. The only dynamic connection among states comes through the Markov switching process for them. The analysis gets more complicated if the observation equation also depends upon lagged states. There have been several proposals for Markov switching models for GDP growth (or, more generally for Markov switching VAR’s) in increasing order of complexity:

\[
\begin{align*}
(4) & \quad x_t = \alpha(S_t) + \varphi_1 x_{t-1} + \ldots + \varphi_p x_{t-p} + u_t \\
(5) & \quad x_t = \alpha(S_t) + \varphi_1(S_t)x_{t-1} + \ldots + \varphi_p(S_t)x_{t-p} + u_t \\
(6) & \quad x_t - \mu(S_t) = \alpha + \varphi_1(x_{t-1} - \mu(S_{t-1})) + \ldots + \varphi_p(x_{t-p} - \mu(S_{t-p})) + u_t \\
(7) & \quad x_t - \mu(S_t) = \varphi_1(x_{t-1} - \mu(S_{t-1})) + \ldots + \varphi_p(x_{t-p} - \mu(S_{t-p})) + (1 - L)u_t
\end{align*}
\]
In (4) and (5) the autoregression changes with the state. In (4), only the intercept changes, while in (5), it’s the entire equation. Either of these can be handled using the methods described so far. The “RegimeF” function just needs to compute the residual and likelihood in each state.

(6) is the Hamilton switching model. With \( n = 2 \), there are high mean growth and low mean growth periods with a state-invariant autoregression linking them to the data. The measurement equation depends upon current and \( p \) lagged values of the states. This is most easily handled by expanding the number of “states” to \( n^{p+1} \) to cover all the combinations that could occur in (6). The augmented states at time \( t \) have the underlying states at \((t,t-1,\ldots,t-p)\). The transition probabilities between most of the augmented states is zero, since we can only move from \((S_t,S_{t-1},\ldots,S_{t-p})\) to \((S_{t+1},S_t,\ldots,S_{t-p+1})\) if \( S_t,\ldots,S_{t-p+1} \) remain the same.

(7) is the model from Lam (1990). While apparently trivially different from (6) (differing just with the \((1-L)\) in the error term), that makes the likelihood at \( t \) dependent upon the states stretching all the way back to the start of the data set since \( u \) is unobservable. While it’s possible to estimate (7) by maximum likelihood, it’s quite a complicated process. Models like this are more often estimated using Markov Chain Monte Carlo procedures, such as Gibbs sampling (Section 13.7).

**MSVARSETUP.SRC**

The procedure `MSVARSetup` sets up a Markov switching VAR (or autoregression if there is only one variable). It creates matrices for the state transition probabilities (\( P \)), the means or intercepts (\( MU \)) and the lag coefficients (\( PHI \)) and defines several functions and formulas needed for estimating the model. See below for an example.

```r
@MSVARSetup( options )
# list of dependent variables

Options

lags=# of VAR lags [1]
states=# of states [2]
switch=[means]/intercepts

switch=MEANS is the Hamilton type model (6) where the mean of the series is state-dependent. switch=INTERCEPTS has the intercept term in each VAR equation being state dependent (model 4).

@msvarsetup(lags=1)
# dusa djap dfrg duk dcan daus
nonlin(parmset=msparms) p
nonlin(parmset=varparms) mu phi sigma
@msvarinitial
compute gstart=1962:1,gend=1991:4
maximize(parmset=varparms+msparms,start=(pstar=MSVARInit()),$ reject=%minvalue(tp)<0.0,method=bfgs,iters=400) msvarf gstart gend
```
Example 12.5 Hamilton Switching Model

This example estimates Hamilton’s switching model for GDP growth (Hamilton (1994), Chapter 22). What makes this tricky is that, although there are just two basic states (expansion and contraction), the use of lags creates 32 combinations of these. Because the calculation of the likelihood is quite complicated, a function is used to compute the probability for an entry (the argument is the entry number).

This type of model is often simplified (considerably) by having the equation determined solely by the contemporaneous state of the Markov process rather than the combination of states through all the lags.

The page references in the comments are from Hamilton. This is example HAMILTON.PRG.

cal(q) 1951:1
open data gnpdata.prn
data(format=prn,org=columns) 1951:1 1984:4
*  
set g = 100*log(gnp/gnp{1})
graph
# g

Hamilton’s GNP growth model has two possible states at each period (interpreted as expansion and contraction). With one lag plus the current, there are 4 possible combinations of states when analyzing a given time period.

compute nlags=4
compute nstates=2**(nlags+1)
dec vect mu(2) phi(nlags)
dec vect tp(5)

nonlin mu phi p11 p22 sigma

To speed matters up, create a lookup table which gives the mapping from the coding for the states into the binary choice for each lag. The current state is 1 for 1–16 and 2 for 17–32. The first lag is 1 for 1–8 and 17–24 and 2 for 9–16 and 25–32, etc.

dec rect[integer] lagstate(nstates,nlags+1)
ewise lagstate(i,j)=1+mod((i-1)/2**(nlags+1-j),2)

We also create a mapping from state to state which will pick out the transition probability. Transit will be a number from 1 to 5. 1 is the most common value, which represents a zero transition probability. For instance, if we’re in \{1,1,2,2,1\}, the only states which can be obtained are \{1,1,1,2,2\} and \{2,1,1,2,2\}. The transition probability to the first of these would be P11, and for the second it would be 1–P11. Given our coding, it’s easy to tell whether a state can be obtained from another, since the final four spots of the new state have to be the same as the lead four spots of the old state.
dec rect[integer] transit(nstates,nstates)
ewise transit(i,j)=fix(%if(%mod(i-1,2**nlags)==(j-1)/2,$
  2*lagstate(i,1)+lagstate(j,1)-1,1))
*

dec vect reg(nstates) f(nstates) p(nstates) $
fp(nstates) pstar(nstates)

The mask series is used to sum up the probability of being in state 1 at any point
in time. The PROB1 series will hold the result of this calculation.

dec vect mask(nstates)
ewise mask(i)=(i<=nstates/2)
set prob1 = 0.0

We create a special purpose function to do a one period update of the Markov
chain. We can’t use one of the standard functions described in the RATS manual
because we don’t have a full 32x32 transition matrix, instead using the lookup
table into a smaller one.

function MarkovProb time
type real MarkovProb
type integer time
*
local integer i j
local real u
local real sp fpt

do i=1,nstates
  compute u=g(time)-mu(lagstate(i,1))
  do j=1,nlags
    compute u=u-phi(j)*(g(time-j)-mu(lagstate(i,j+1)))
  end do j
  compute reg(i)=u
end do i
*
ewise f(i)=%density(reg(i)/sigma)/sigma

do i=1,nstates
  compute sp=0.0
  do j=1,nstates
    compute sp=sp+tp(transit(i,j))*pstar(j)
  end do j
  compute p(i)=sp
end do i
compute fp=f.*p
compute fpt=%sum(fp)
compute pstar=fp*(1.0/fpt)
compute prob1(time)=%dot(mask,pstar)
compute MarkovProb=fpt
end
*
frml ggrowth = log(MarkovProb(t))
PSTAR (the probability distribution of the states at the previous period) needs to be initialized with each function evaluation, as it gets updated at each period. We also copy the transition probabilities into the TP vector. The initial distribution is set equal to the ergodic probabilities of the states, given the current parameters. The solution method is shown on page 684 of Hamilton.

The “A” matrix from 684 has 1–the transition probs in the top NxN, with a row of 1’s below that.

```r
function MarkovInit
  type vector MarkovInit
  *
  local rect a
  local integer i j
  *
  compute tp=||0.0,p11,1-p22,1-p11,p22||
  dim a(nstates+1,nstates)
  ewise a(i,j)=%if(i>nstates,1.0,(i==j)-tp(transit(i,j))
  compute MarkovInit=%xcol(inv(tr(a)*a)*tr(a),nstates+1)
end
```

Note that the states aren’t globally identified. We would get exactly the same likelihood if state 1 were contraction and 2 were expansion. The choice of initial values for mu and the optimization method can alter whether, in the process of estimation, the states “trade places.”

```r
compute p11=.85,p22=.70
compute mu(1)=1.0,mu(2)=0.0
linreg g
# constant g{1 to 4}
compute phi=%xsubvec(%beta,2,5)
compute sigma=%seesq
*
maximize(start=(pstar=MarkovInit()),method=bfgs) ggrowth nlags+2 *
```

To create the shading marking the recessions, create a dummy series which is 1 when the RECESSIONS series is 1, and 0 otherwise. (RECESSIONS is 1 for NBER recessions and -1 for expansions).

```r
set contract = recessq==1
*
spgraph(vfields=2)
graph(header="Quarterly Growth Rate of US GNP",shade=contract)
# g %regstart() %regend()
set prob2 = 1-prob1
graph(style=polygon,shade=contract,$
  header="Probability of Economy Being in Contraction",)
# prob2 %regstart() %regend()
spgraph(done)
```
Example 12.6 SWARCH (Switching ARCH)

Switching ARCH/GARCH models were first proposed by Hamilton and Susmel (1994) as a possible alternative to the standard ARCH and GARCH models. Instead of persistence in volatility being a function of the size of past residuals and the variance, it comes through a Markov model for two or more volatility states.

Technically, this is much easier to handle for ARCH than GARCH, so that’s what this does. Besides that, the persistence that the lagged variance term in GARCH provides is handled in this model by the Markov process.

This estimates a three state Markov model. There are several possible ways to model the effect of the different states on volatility: the one we choose here multiplies the constant term in the variance equation by the state dependent multiple. The variances are all relative to that in state 1, so the “HV” vector has one less element than the number of states.

This is example SWARCH.PRG.

```plaintext
open data g10xrate.xls
data(format=xls,org=columns) 1 6237 usxjpn

Convert to percent daily returns
set x = 100.0* log(usxjpn/usxjpn{1})

These set the number of states, and the number of ARCH lags
source markov.src
compute nstates=3
compute q=2
P is the matrix of transition probabilities
dec rect p(nstates-1,nstates)

HV will be the relative variances in the states. This will be normalized to a relative variance of 1 in the first state, so it’s dimensioned N-1.
dec vect hv(nstates-1)

This will be the vector of ARCH parameters
dec vect a(q)

We have three parts of the parameter set: the mean equation parameters (here, just an intercept), the ARCH model parameters, and the Markov switching parameters.
nonlin(parmset=meanparms) mu
nonlin(parmset=archparms) a0 a
nonlin(parmset=msparms) p hv *
dec vect pstar(nstates)
UU and U are used for the series of squared residuals and the series of residuals.
clear uu u
```
ARCHStateF returns a vector of likelihoods for the various states at time given residual E. The likelihoods differ in the states based upon the values of HV, where the intercept in the ARCH equation is scaled up by HV. Again, the elements of HV are offset by 1 since they’re normalized with the first state at 1.

```r
function ARCHStateF time e
type vector ARCHStateF
type real e
type integer time
*
local integer i j
local real vi
dim ARCHStateF(nstates)
do i=1,nstates
    compute vi=a0*%if(i>1,hv(i-1),1)
do j=1,q
        compute vi=vi+a(j)*uu(time-j)
end do i
compute ARCHStateF(i)=%if(vi>0,$
    %density(e/sqrt(vi))/sqrt(vi),0.0)
end do i
end
```

As is typically the case with Markov switching models, there is no global identification of the states. By defining a fairly wide spread for HV, we'll hope that we'll stay in the zone of the likelihood where state 1 is low variance, state 2 is medium and state 3 is high.

```r
compute hv=||10,100||
compute p=||.8,.2,.0|.2,.6,.4||
stats x
compute mu=%mean,a0=%variance,a=%const(0.05)
set uu = %variance
```

The log likelihood function recursively calculates the vector PSTAR of estimated state probabilities. The first step at each T is to compute the mean and update the U and UU series. The likelihoods in the states and computed by the ARCHStateF function, and then %MCSTATE and %MSUPDATE update the PSTAR vector and compute the likelihood. We use a START formula on maximize to start PSTAR at the ergodic probabilities.

Because we need 2 lags of UU, the estimation starts at 3.

```r
frml logl = u(t)=(x-mu),uu(t)=u(t)**2,f=ARCHStateF(t,u(t)),$
pstar=%msupdate(f,%mcstate(p,pstar),fpt),log(fpt)
maximize(start=(pstar=%mcergodic(p)),method=bhhh,pmethod=simplex,$
piters=5,parmset=meanparms+archparms+msparms) logl 3 *
```
12.4 Non-Parametric Regression and Density Estimation

The Tools

DENSITY does kernel density estimation. It can estimate the density function and, for the differentiable kernels, the derivative of the density.

NPREG is for the non-parametric regression of one series on another. Its design is quite similar to DENSITY for the kernel-based regressions (Nadaraya-Watson). There’s also an option for LoWeSS (Locally WEighted Scatterplot Smoother), which isn’t used as much in econometrics.

Both NPREG and DENSITY have a GRID=INPUT/AUTOMATIC option. When you’re using them just to get a graph of the shape, the GRID=AUTOMATIC usually works fine. Use GRID=INPUT when you need either to better control the grid, or, if, as in Example 12.7, you need to evaluate at existing data in order to use the output as part of a more involved calculation. (The “grid” doesn’t really have to be a grid in those cases. There’s actually no computational reason for it to be organized in any particular way).

Bandwidths

The default bandwidth on DENSITY

\[(.79 \text{IQR})N^{-\frac{1}{5}}, \text{ where IQR=interquartile range, N=number of observations}\]

has certain optimality properties in larger samples (see the discussion in Pagan and Ullah (1999)), but you might find it to be too small in some applications. The bandwidth used can be obtained after the instruction (either DENSITY or NPREG) from the variable $%EBW$. You can adjust off of that, if you’re concerned about using too narrow a bandwidth.

Example

```
garch(p=1,q=1,hseries=h,resids=u) / xjpn
set ustandard = u/sqrt(h)
set xgrid 1 12000 = -6.0+.001*t
density(type=gauss,grid=input) ustandard / xgrid fgrid
set ngrid = %density(xgrid)
scatter(style=line) 2
# xgrid fgrid
# xgrid ngrid
```

estimates a GARCH(1,1) model, and a density function for the standardized residuals, using a very fine input grid running over \([-6,6]\). The computed density is graphed against the standard normal density.
Example 12.7 Non-Parametric Regression

This is from Pagan and Ullah (1999, p. 248). The data set shows state by state (total) expenditures for travel, along with population and (total) income. The regression is expenditure on income. It seems reasonable to assume that the variance of the residuals is somehow related to population. This first does weighted least squares assuming variance proportional to population squared, then estimates a flexible scedastic function using \texttt{NPREG}. Two parametric functions are also tried. This is example \texttt{NPREG.PRG}.

\begin{verbatim}
open data travel.csv
data(format=prn,org=columns) 1 51 pop income exptrav *
linreg exptrav / resids
  # constant income

  Assumed scedastic function is POP**2
set popsq = pop**2
linreg(spread=popsq) exptrav
  # constant income

  Nonparametric estimator
set ressqr = resids**2
npreg(grid=input,type=gaussian) ressqr pop / pop vpopnp
linreg(spread=vpopnp,${
  title="Semiparametric Weighted Least Squares") exptrav
  # constant income

  Parametric estimates of alternative scedastic functions
linreg ressqr
  # popsq
prj vpopsq
linreg ressqr
  # constant pop popsq
prj vpopquad

  The data set is already sorted by POP, so STYLE=LINES will work
scatter(style=lines) 3
  # pop vpopsq
  # pop vpopquad
  # pop vpopnp

  This reveals that a big problem is one truly massive outlier
scatter(style=lines,overlay=dots,ovsame) 4
  # pop vpopsq
  # pop vpopquad
  # pop vpopnp
  # pop ressqr
\end{verbatim}
Example 12.8 Adaptive Kernel Estimation

This is also an example from Pagan and Ullah (1999, p 250). It does an adaptive kernel estimator. For the regression equation

\( y_i = X_i \beta + u_i, \)

if the density of \( u \) is the (unknown) \( f \), the derivative of the log likelihood with respect to \( \beta \) is

\( -\sum X_i (f'(u_i) / f(u_i)) \equiv -\sum X_i \psi_i \)

The adaptive kernel estimator is a two-step estimator which starts with OLS and uses kernel estimates of

\( \psi_i = f'(u_i) / f(u_i) \)

The second step is to compute the change in \( \beta \) that would push (2) towards zero. A “method of scoring” step would take this step as (minus) the information matrix times the gradient. Assuming that \( u \) and \( X \) are independent, the information matrix can be estimated as

\( (1/T) \sum \psi_i^2 \sum X_i'X_i \)

which can be computed using MCOV with the NOZUDEP option. If you’re not willing to assume that, you could use an OPG estimate as

\( \sum X_i'\psi_i^2X_i \)

This is computable using MCOV with ZUDEP, and is needed at any rate to estimate the covariance matrix.
This is example ADAPTIVE.PRG.

```plaintext
open data housing.csv
data(format=prn,org=columns) 1 59 price age aircon baths bedrms $
   cond corner culd dish fence firepl floors garage irreg lajolla $
   lndry patio pool rooms sprink sqft view yard

set lsqft = log(sqft)/log(10)
set lyard = log(yard)/log(10)

Estimate linear regression with White standard errors
linreg(robusterrors) price / resids
# sqft yard pool lajolla baths firepl irreg sprink view lsqft $
lyard constant

Adaptive kernel estimator. We need to compute the density at the given data
points, so GRID=INPUT is used with the RESIDS providing the evaluation
points.
density(type=gauss,derives=f1,grid=input) resids / resids fu
set psi = f1/fu

The information matrix is computed by using MCOV with the NOZUDEP option
as u and X are assumed to be independent.
mcov(lastreg,matrix=ibxx,nozudep) / psi

The same but with ZUDEP is needed for the center of the “sandwich”. We also
need the gradient, which will be %NOBS * the mean vector of the products of psi
with the regressors.
mcov(lastreg,matrix=ib,meanvector=mv) / psi
compute d=mv*%nobs

Display the regression output with the adjusted beta and sandwich estimator of
the covariance matrix.
linreg(create,lastreg,form=chisquared,$
title="Adaptive Kernel Estimator",$
coeffs=%beta-inv(ibxx)*d,covmat=inv(ibxx)*ib*inv(ibxx))
```
12.5 Linear and Quadratic Programming

The instruction \texttt{LQPROG} provides quick and efficient solutions to linear and quadratic programming problems.

Constraints

\texttt{LQPROG} minimizes either a linear or quadratic function in \( x \) subject to

1. \( A_e x = b_e, \ A_i x \leq b_i, \ A_g x \geq b_g \)

2. \( x_i \geq 0 \) for all \( i = 1, \ldots, n\text{neg} \)

where, by design, \( b_e \geq 0, \ b_i \geq 0, \ b_g \geq 0 \). A constraint with a naturally negative value of \( b \) needs to be sign-flipped.

The constraints are input using matrices \( A \) and \( b \) constructed as follows:

\begin{equation}
A = \begin{bmatrix}
A_e \\
A_i \\
A_g
\end{bmatrix}, \ b = \begin{bmatrix}
b_e \\
b_i \\
b_g
\end{bmatrix}
\end{equation}

In other words, equality constraints must be listed first, followed by those inequality constraints which are \( \leq \) a non-negative constant, and finally those which are written as \( \geq \) a non-negative constant. You input the constraints to \texttt{LQPROG} using the options \texttt{A=A matrix} and \texttt{B=B vector}. You have to indicate the number of equality constraints with \texttt{EQUAL=number of equalities}, and the number of \( \geq \) constraints with \texttt{GE=number of non-negativity constraints}. The default for each of those options is 0, so all the input constraints are \( \leq \) by default.

For linear programming, all the \( x \)'s are constrained to be non-negative, that is, \( n\text{neg} \) in (2) is the number of variables. In contrast, a quadratic program can handle unbounded arguments. You have to arrange the elements of \( x \) so the ones which are forced non-negative come first. The option \texttt{NNEG=number of non-negative x's} can be used if there are some that are constrained to be non-negative and some unbounded. By default, \textit{all} are constrained to be non-negative.
Linear Programming

**LQPROG** can solve linear programming problems of the form:

(4) minimize $c^T x$

subject to the constraints described on the previous page. You input the cost vector with option \( C=C \ vector \).

In this example, a firm is trying to allocate its advertising budget between magazine and television. Television costs more, but reaches a larger audience. The magazine readership, while smaller, has superior demographics. They want to allocate their budget in the most cost-effective manner given goals of meeting or exceeding total exposures and exposures for subgroups.

(5) minimize $40m + 200t$

subject to $4m + 40t \geq 160$ total exposures

$3m + 10t \geq 60$ high income

$8m + 10t \geq 80$ age group

$m,t \geq 0$

The input matrices for this problem are

(6) $A = \begin{bmatrix} 4 & 40 \\ 3 & 10 \\ 8 & 10 \end{bmatrix}$, $b = \begin{bmatrix} 160 \\ 60 \\ 80 \end{bmatrix}$, $c = \begin{bmatrix} 40 & 200 \end{bmatrix}$

The input constraints are all $\geq$, so we need the output \( GE=3 \). We can solve the problem with the following program:

```rats
dec rect a
dec vect b
compute a=||4.0,40.0|3.0,10.0|8.0,10.0||
compute b=||160.0,60.0,80.0||
*lqprog(ge=3,c=||40.0,200.0||,a=a,b=b) x
```

The output is

In LPROG, minimum = 1000.00000000
Solution $x =$
10.000000000 3.000000000

Lagrange Multipliers
2.500000000 10.000000000 0.000000000
Chapter 12: Special Models

Quadratic Programming

LQPROG solves quadratic programming problems of the form:

\[
(7) \quad \text{minimize} \quad \frac{1}{2} x'Qx + c'x
\]

subject to the constraints described earlier. As noted there, you can have components of \( x \) which are allowed to be negative, but you must arrange the input matrices so that the ones subject to the non-negativity constraint come first. The Hessian matrix of the objective function, \( Q \), is an \( nvar \times nvar \) symmetric matrix. The objective function is input to LQPROG using the options Q=Q matrix and C=C vector.

For quadratic problems, LQPROG uses the active set method with the conjugate gradient method (see Luenberger (1989)). An initial feasible solution is obtained by employing linear programming with artificial variables.

Note that you might have some work to do in transforming a problem to the form (7). For instance, suppose that your objective function is

\[ (8) \quad x_1^2 + x_2^2 + x_3^2 + x_1 x_3 - 10x_1 - 4x_3 \]

To convert a function like this which is written out as a second order polynomial in \( x_1, x_2, \ldots, x_n \), call the second order terms \( S_{ij}x_i x_j \). Then the elements for \( Q \) are:

\[ (9) \quad Q(i,j) = \begin{cases} 2S_{ij} & \text{for } i = j \quad \text{(diagonal elements)} \\ S_{ij} & \text{for } i \neq j \quad \text{(off-diagonal elements)} \end{cases} \]

The diagonal elements of \( Q \) are twice the quadratic coefficients on the “squared” terms, so for instance \( Q(1,1) \) should be twice the coefficient on \( x_1^2 \). The off-diagonal elements of \( Q \) are set equal to coefficients on the corresponding quadratic term: \( Q(1,2) \) would be the coefficient on \( x_1 x_2 \). For (8), the matrices in the objective function are:

\[ (10) \quad Q = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}, c' = [-10, 0, -4] \]
Example

The following estimates a regression subject to non-negativity constraints and an adding up constraint. In matrix form, the sum of squared residuals can be written

\[(y - X\beta)'(y - X\beta) = \beta'X'X\beta - 2y'X\beta + y'y\]  

The last term doesn’t involve \(\beta\) and so can be ignored in finding the optimal coefficients. If we multiply this by 1/2 we can read off \(Q = X'X\) and \(c = -X'y\). For illustration, we will add to this the constraint that \(1 \cdot \beta \leq 1\). \(X'X\) and \(X'y\) can be obtained as submatrices of the cross-moment matrix of the regressors and \(y\).

```plaintext
compute [symmetric] q = %xsubmat(%cmom,1,13,1,13)
compute [vector] c = -1.0*%xsubmat(%cmom,14,14,1,13)
compute a=%fill(1,13,1.0)
```

Example 12.9 Portfolio Optimization

LQPROG is the ideal tool for solving portfolio optimization problems. If you need to solve these subject to a restriction against taking short positions, there is no simple alternative. Even if short positions are allowed (when the portfolio problem can be solved directly by a sequence of matrix calculations), you’ll probably find that it’s simpler to use LQPROG with the option NNEG=0 than to work through a string of matrix operations.

This is example file PORTFOLIO.PRG.

```plaintext
open data berndt.xls
calendar(m) 1978
data(format=xls,org=columns) 1978:1 1987:12 mobil ibm weyer citcrp
market rkfree

Compute the covariance matrix of returns into OMEGA. Save the means as MU.

vcv(center,matrix=omega)
* mobil ibm weyer citcrp
compute n=%nvar
compute mu=%meanv
compute [rect] ones=%fill(1,n,1.0)

Compute the minimum variance portfolio, and its expected return. All the analysis is constraining portfolios to non-negative weights. If you want to allow short positions, add the NNEG=0 option to LQPROG.

lqprog(q=omega,a=ones,b=1.0,equalities=1) x
compute minr=%dot(x,mu)

Choose as upper limit the highest expected return.

compute maxr=%maxvalue(mu)
```
Create a grid series for expected values running from MINR to MAXR.

set erets 1 101 = minr+(maxr-minr)*.01*(t-1)
set srets 1 101 = 0.0

Find the minimum variance portfolio with the expected return ERETS(T). Note—the problem being solved is to min .5 x'Qx subject to the constraints. To get the variance (rather than .5 variance), we need to multiply the optimized value by 2.

do t=1,101
   lpqprog(noprint,q=omega,a=ones~~tr(mu),b=1.0~erets(t),equal=2)
   compute srets(t)=sqrt(2.0*%funcval)
end do t

scatter(style=lines,$
   header="Efficient Frontier w/o Risk-Free Asset","$
   vlabel="Expected Return","$
   hlabel="Sigma")
# srets erets

With riskfree assets, there’s no longer a constraint that the weights on the portfolio sum to one. Instead, the expected return constraint is altered to

$$\omega'(\mu - 1 \rho_i) = E - R_i$$

stats(noprint) rkfree
compute rf=%mean
set sretrf 1 101 = 0.0
compute arf=tr(mu)-rf
do t=1,101
   lpqprog(noprint,q=omega,a=arf,b=erets(t)-rf,equalities=1)
   compute sretrf(t)=sqrt(2.0*%funcval)
end do t
*
scatter(style=lines,$
   header="Efficient Frontier with Risk-Free Asset","$
   vlabel="Expected Return","$
   hlabel="Sigma")
# sretrf erets
# srets erets
# sretrf erets
12.6 Neural Networks

Artificial neural network models provide a powerful alternative to standard regression techniques for producing time-series and cross-sectional models. Neural networks are particularly useful for handling complex, non-linear univariate and multivariate relationships that would be difficult to fit using other techniques.

Despite the fact that neural networks are simply a class of flexible non-linear functional forms, there has arisen a whole different set of terminology to describe their structure and the fitting process. While we present the standard neural network terms, we try also to provide a translation into a standard regression framework.

Overview of Neural Networks

A neural network consists of an input layer containing one or more input “nodes” (in effect, the explanatory variables in a regression model), and an output layer of one or more output nodes (analogous to the dependent variable(s)). They normally also include a hidden layer, which lies between the input and output layers and consists of one or more hidden nodes.

Each input node is connected to each hidden node, and, in turn, each hidden node is connected to each output node. The model may also include direct connections between each input node and each output node. These can be used in addition to, or instead of, the hidden layer connections.

The diagram below represents a simple neural network with two input nodes ($I_1$ and $I_2$), one hidden node ($H_1$), and one output node ($O_1$). The solid lines represent the connections between the input nodes and the hidden node, and between the hidden node and the output node. The dashed lines represent direct connections between the input nodes and the output node.

The hidden nodes and output nodes each have sets of weighting values associated with them. These serve the same purpose as the coefficients of a regression model. Given a set of input values, the weights determine the output(s) of the network.

In particular, each hidden node has a bias weight (intercept, in effect), plus separate input weights for each input node attached to it. In the example above, $H_1$ would
have three weights: a bias weight (which we’ll call $h_{10}$), the input weight associated with input $I_1$ ($h_{11}$), and the input weight for $I_2$ ($h_{12}$). Given the input values $I_1$ and $I_2$, the value of this hidden node would given by:

$$u_1 = h_{10} + I_1 h_{11} + I_2 h_{12}$$

The actual output of the hidden node is determined by applying a “squashing” function to the value $u$, which scales the output to range between zero and one.

Output nodes have a similar set of weights. For the example above, we would have $o_{10}$ (the bias weight on output node 1), $o_{11}$ (the weight of output node 1 for hidden node 1), $d_{11}$ (weight of output node 1 on the connection with input node 1), and $d_{12}$ (weight of output node 1 on the connection with input node 2). If we use a logistic squashing function for the hidden node, the output of this model would be:

$$s_1 = \frac{1}{(1 + e^{-u_1})}$$

$$O_1 = o_{01} + o_{11}s_1 + d_{11}I_1 + d_{12}I_2$$

Fitting a neural network model involves “training” the model by supplying sets of known input and output values, and allowing the neural network algorithm to adjust the hidden node and output node weights until the output produced by the network matches the actual output in the training sample to the desired degree of accuracy.

Once trained, the model can then be used to generate new output data (fitted values or forecasts) from other sets of inputs. Assuming the fit is good, and that the relationships represented by the sample input and output data generalize to other samples, the model can produce good predictions.

However, care must be taken to avoid “overfitting” the model. This occurs when a network is trained to such close tolerances that it ends up modeling the “noise” in the training sample, not just the “signal.” Such a network will generally do a poor job of predicting appropriate output values for other sets of inputs. See below for details.

The **NNLEARN** and **NNTEST** instructions

The **NNLEARN** instruction is used to define a new neural network model, and to do additional training on an existing model. The **NNTEST** instruction is used to generate output (that is, fitted values) from a set of inputs, using a network model estimated using **NNLEARN**. For simple models, you may only need to use **NNLEARN** once to fit the network. For more complex models, you will often use **NNLEARN** and **NNTEST** in an iterative process, first training the model using **NNLEARN**, using **NNTEST** to generate output for comparison with training or validation samples, doing additional training as needed with **NNLEARN**, and so on.

**SERIES** variables are used for both the inputs and outputs on **NNLEARN** and **NNTEST**. This puts all of RATS’ data handling and transformation capabilities at your disposal.
in preparing data for neural network modeling, and for handling output. For example, you can: use the standard sample-range capabilities to train and test models using different samples; use standard lag or lead notation on the inputs for constructing time-series models; and graph fitted values against actual output.

**The Backpropagation Algorithm**

**NNLEARN** uses backpropagation techniques with an adaptive learning rate algorithm to train the model to a user-specified level of convergence. Translation: this is similar to steepest descent except that the derivatives for each weight are adjusted separately based upon the history of recent iterations. If you have more than one hidden node, the basic model isn’t identified in the standard statistical sense. Starting from randomized initial conditions, backpropagation does a lot of searching back and forth to find features for each of the nodes to pick up. This takes a lot of calculation, but allows the model to fit quite complex surfaces.

Because the entire process of fitting a neural network is so different from standard hill-climbing procedures, the options for controlling the process and determining convergence are different as well. For instance, there is an **ITERATIONS** option, but the required limit on this in most applications is quite large. The 100 or so that are usually the most required for a hill-climbing procedure will be far too small. **NNLEARN** does less calculation per “iteration” but also accomplishes less in terms of improving the fit (and may very well accomplish nothing). You might need tens or even hundreds of thousands of iterations to properly train a complex network.

**Using NNLEARN**

To create a neural net model, run the **NNLEARN** instruction with a set of input and output data, using the **SAVE** option to save the generated weights in a **memory vector**.

To do additional training on the same model, just do another **NNLEARN** instruction using the same memory vector on the **SAVE** option. You can either train on the same data set (perhaps setting a tighter convergence criterion or simply allowing more iterations if the desired criterion hasn’t been met), or you can train the model using other training samples (different entry ranges of the same input and output series, or entirely different input/output series). **NNLEARN** will use the existing values stored in the memory vector as starting values for the estimation. The new weights computed by **NNLEARN** will then be saved back into the memory vector, replacing the previous values.

You may want to copy the contents of the memory vector to another array after each training step (with an instruction of the form **COMPUTE array = memory vector**). This allows you to go back to an earlier model if you find that subsequent training has resulted in overfitting. It also makes it easier to compare the outputs generated by the model at various stages of training.

You can also save memory vectors to disk for use in a later session. Use **OPEN COPY** and **WRITE(UNIT=COPY)** to write the vector to disk, and **DECLARE VECTOR, OPEN DATA** and **READ(VARYING)** to read it back into memory.
Chapter 12: Special Models

Tips on Fitting Neural Network Models

As noted above, a potential pitfall of neural network models is that they are prone to “overfitting” the data. The goal in creating a neural network is to model accurately the underlying structural relationship between your input data and output data. If you use too many hidden nodes, or allow the model to train too long (if you use too tight a convergence criterion), your neural network may overfit the data, meaning that in addition to fitting the underlying signal, the network also models the noise in the training sample. Although an overfitted model will do an excellent job of modelling a particular training sample, it will often do a very poor job of modelling the general behavior of the process.

There are several ways to avoid overfitting. One is to make sure that you use only the minimum required number of hidden nodes. However, finding the optimal number of hidden nodes is often very difficult. If you end up using too few hidden nodes, the network will be unable to produce a good fit.

The other is to stop the training process before overfitting occurs. We would recommend that you use this method, because it is usually much easier to stop the training at the appropriate point than to try and determine the optimal number of hidden nodes. Also, this approach allows you to err on the side of having too many hidden nodes, rather than too few.

As suggested above, this is an iterative process, with the following basic steps:

1) Create and train the model using \texttt{NNLEARN}, with a fairly loose convergence criterion, and/or a relatively low limit on the number of iterations. Be sure to use the \texttt{SAVE} option to save the memory vector.

2) Use a \texttt{COMPUTE} instruction to copy the current values of the memory vector to another array. Use a different name each time you do this step. If you find that you have overtrained the model, just go back to the memory vector saved previously and either use it as is, or do additional training (but to a looser criterion than the one that produced the overfitted model), or using a different set of training data.

3) Generate fitted values from the model using \texttt{NNTEST} with the estimated memory vector. Use graphs or \texttt{PRINT} instructions to compare the fitted and actual values. If you have a “validation” sample (a set of input and output values excluded from the training sample to be used to test the quality of the fit), use the validation sample inputs, and compare the resulting output to the validation sample outputs (by looking at the sum of squared errors, graph the validation and fitted values, etc.).

4) Repeat these steps until the desired fit is achieved. You can do additional training using the same set of sample inputs and outputs with a tighter convergence criterion (or higher iteration limit), or you can use additional training samples. Be sure to use the same memory vector on the \texttt{SAVE} option each time.
so the model starts up where it left off. However, by saving each set of weights into a different array in step (2), you’ll be able to go back to previous states if you find that the model eventually trains to the point of being overfit.

You are looking for the level of training that produces the minimum error when comparing network output to a separate validation sample. As the model fit improves with training, the errors should decrease steadily. At some point, however, the errors may begin to rise again, indicating that the model is probably starting to overfit the training sample (and thus providing a poorer fit for the validation sample).

We recommend that you always use the \texttt{TRACE} option to track the progress of the estimation. Some models will require tens of thousands or hundreds of thousands of epochs to produce a good fit. You may also find that \texttt{NNLEARN} sometimes reaches “plateaus” where the mean square error doesn’t change much for several thousand iterations, and then makes a significant improvement.

Also, be aware that the initial values used for the internal weights of a new network are determined randomly (if you aren’t using \texttt{SAVE} to supply an existing model and weights), so results will differ slightly from run to run. You can use the \texttt{SEED} instruction to seed the random number generator if you want the results of a program to be exactly reproducible.

Finally, you may occasionally run into situations where \texttt{NNLEARN} has trouble getting started, perhaps doing many thousands of epochs without making any significant progress. In such cases, you might want to interrupt and re-start the \texttt{NNLEARN} instruction in the hopes that a different set of initial values will perform better. However, these cases are fairly rare. In general, you simply need to be patient.

\textbf{Convergence}

In order to construct a useful neural network model, you will need to train the model sufficiently so that it models accurately the underlying behavior of the data, but not so tightly that it “overfits” the data used to train the model. That is, you want to model the “signal” present in your data, but not the noise.

RATS provides three options for controlling the convergence of the model. In most cases, you will use these options in an iterative process that involves invoking \texttt{NNLEARN} several times for the same model. The options available are:

\texttt{iters=iteration limit [no limit]}
\texttt{cvcrit=convergence criterion [.00001]}
\texttt{rsquared=minimum R-squared level}

The \texttt{CVCRIT} and \texttt{RSQUARED} options are mutually exclusive—they provide two ways of specifying the convergence criterion for the learning process. Both can produce equivalent fits, they simply offer different ways of thinking about the criterion. The default setting is \texttt{CVCRIT=.00001} (if you use both, RATS will take the \texttt{CVCRIT} setting).
If you use the `CVCRIT` option, RATS will train the model until the mean square error (the mean of the squared error between the output series and the current output values of the network) is less than the `CVCRIT` value.

If you use the `RSQUARED` option, RATS will train the model until the mean square error is less than \( (1 - R^2)\sigma^2 \), where \( R^2 \) is the value specified in the `RSQUARED` option, and \( \sigma^2 \) is the smallest of the output series variances.

The main disadvantage of `CVCRIT` is that it is dependent on the scale of the variables in the model. For example, suppose a `CVCRIT` of .0001 produces a good fit for a particular model. If you took the same model, but multiplied the output series by a factor of 10000, this `CVCRIT` setting would probably be much too tight.

The `RSQUARED` option is particularly handy for problems where you have a reasonable idea of what kind of “goodness of fit” you can expect from the model. Perhaps more importantly, the `RSQUARED` criteria is less dependent on the scale of the output because it is scaled by the output variance.

By default, `NNLEARN` will iterate until it satisfies the criteria set with `CVCRIT` or `RSQUARED`. You can use `ITERS` to place an upper limit on the number of iterations that `NNLEARN` will perform—it will stop iterating after the specified number of iterations, even if the criteria has not yet been met.

### Padding the Output Range

The individual inputs are all rescaled internally to a range of \([0, 1]\). In most cases, this will ensure that the coefficients are all within a few orders of magnitude of 1, which makes it easier to fit by backpropagation. Just as with any type of model that is fit to data, if you try to forecast values using explanatory variables that are far removed from those used in fitting (training) it, the results may be unreliable.

A more serious problem with neural nets, though, comes from the restriction on the range of the output variables. The rescaling of the inputs is done for convenience: since each has a freely estimated multiplicative coefficient whereever it appears, any scale choice can be “undone” by doing the opposite rescaling on the coefficients. However, the rescaling of the outputs is mandatory because of the use of the “squashing” function. If the output values are also rescaled to \([0, 1]\), then the network’s outputs will be constrained to the range from the training sample, since the squashing function can’t produce a value larger than 1. This will be a problem if you’re attempting to forecast a trending variable. To avoid this problem, you can use the option `PAD=fraction to pad`. This provides 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 smaller 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.
Examples

This example fits a neural network to the function:

\[ y_i = \sin(20x_i) \text{ where } x_i = 0.01, 0.02, ..., 1.00 \]

You can experiment with this example to see the effects of changing the number of hidden nodes and/or the convergence criterion. We’ve used the CVCRT option in this example—if you use RSQUARED, try a setting of about 0.8:

```
all 100
set input = t/100.
set sinewave = sin(input*20)
nnlearn(hidden=6,save=memvec,trace,cvcrit=.01,ymax=1.0,ymin=-1.0)
# input
# sinewave
nntest / memvec
# input
# output
graph(key=below) 2
# sinewave
# output
```

This is an example from Tsay (2005, pp. 180-181). It fits a model with two hidden nodes and direct connection from inputs to the returns on IBM stock, using three lags as inputs. The sample through 1997:12 is used for training, while the sample from 1998 on is forecast.

```
nnlearn(rsquared=.10, iters=100000, hidden=2, direct, save=nnmodel) * 1997:12
# ibmln{1 2 3}
# ibmln

nntest 1998:1 1999:12 nnmodel
# ibmln{1 2 3}
# nnfore
@uforeerrors ibmln nnfore
```
Example 12.10 Neural Network

This fits a neural net to a binary choice model. Just like a probit model (which is also estimated here), the neural net attempts to explain the $YESVM$ data given the characteristics of the individuals. Aside from a different functional form, it differs because the neural net uses the sum of squared errors rather than the likelihood as a criterion function. This is example NEURAL.PRG.

```
open data probit.dat
data(org=obs) 1 95 public1_2 public3_4 public5 private $
   years teacher loginc logproptax yesvm

   Linear probability model. Compute the “fitted” values. Because the LPM doesn’t constrain the fitted values to the [0,1] range, some of them may be (and are) outside that.
linreg yesvm
# constant public1_2 public3_4 public5 private years teacher $
   loginc logproptax
prj lpmfitted

   Probit model. Compute the fitted probabilities.
ddf(dist=probit) yesvm
# constant public1_2 public3_4 public5 private years teacher $
   loginc logproptax
prj(distr=probit,cdf=prfitted)

   Neural network. We use two hidden nodes. (One won’t be much different from the models above). Note that the CONSTANT isn’t included in the explanatory variables, since it’s automatically included.
nnlearn(hidden=2, iters=10000, save=nnmeth)
# public1_2 public3_4 public5 private years teacher $
   loginc logproptax
# yesvm

   Compute the forecast values from the network.
nntest / nnmeth
# public1_2 public3_4 public5 private years teacher $
   loginc logproptax
# testvm

   Compute the number of correct predictions for the various models
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>>lpmyes prfitted>.5>>prbyes
*
report(action=define,hlabels=||"Vote","Actual",$ 
   "Neural Net","LPM","Probit"||)
report(atcol=1) "No" nos nnnos lpmnos prbnos
report(atcol=1) "Yes" yes nnyes lpmyes prbyes
report(action=show)
```
Chapter 13
Simulations, Bootstrapping, and Monte Carlo Techniques

RATS can handle easily a wide variety of Monte Carlo experiments and simulations. With its many programming features plus a wide variety of probability functions, it has the needed flexibility, since each application of these techniques tends to have its own special characteristics.

The emphasis here is on demonstrating the tools available, along with several fully worked examples which can provide a guide for your own work.
Chapter 13: Simulations/Bootstrapping

13.1 Bookkeeping

Overview

RATS provides several instructions, including \texttt{SIMULATE}, \texttt{FORECAST}, and \texttt{BOOT}, and a number of built-in functions, such as \texttt{%RAN} and \texttt{%UNIFORM}, which make it possible to implement a wide variety of Monte Carlo experiments and simulations. We will discuss these special instructions and functions in detail later on. First, let’s look at the basic bookkeeping techniques that will be a part of almost any bootstrapping or simulation task.

Bookkeeping

Usually, it is a fairly simple task to set up for a single draw from your model. However, there is very little call for the results of a single draw—the typical goal is an estimate of the probability of an event, or a moment of a distribution. To compute these, you need to loop over the code performing the simulations and do some bookkeeping with the results. RATS provides the following tools to accomplish this:

1. \texttt{DO} for looping over a block of instructions. RATS provides several other looping functions (\texttt{DOFOR}, \texttt{LOOP}, \texttt{WHILE}, and \texttt{UNTIL}), but \texttt{DO} is the one most often used for simulations.
2. \texttt{COMPUTE} for updating counters and totals.
3. \texttt{EWISE} for general record keeping with matrices.
4. \texttt{SET} and \texttt{GSET} for updating counters and totals with series.

These examples use techniques described in the next section for generating the random draws.

Progress Indicators

Before you do 10,000 replications on a model, it’s a good idea to start with a more modest number and make sure that the program is doing what you want. Write your code using a variable like \texttt{NDRAWS} to represent the number of desired draws so you can change it easily. When you’re ready for a full run, keep in mind that these methods are called, with good reason, \textit{computationally intensive}. Even with a fast computer, it can sometimes take minutes, or even hours, to run enough replications to get results at the desired level of accuracy. (In general, accuracy improves with the square root of the number of draws.) We use the \texttt{INFOBOX} instruction in these situations to give an idea of how things are going.

\begin{verbatim}
infobox(action=define,progress,lower=1,upper=ndraws) "Progress"
do draws=1,ndraws
  .... instructions ...
  infobox(current=draws)
end do draws
infobox(action=remove)
\end{verbatim}
Computing a Probability

To compute a probability, you need to keep track of the number of times an event occurred, and divide by the number of draws that you made. You can use an INTEGER variable or entries of a series or array to hold this information. For instance, the following uses the SIMULATE instruction to generate simulations of a GDP model, and uses the simulated data to compute the probability distribution for the onset of the first recession (defined as two consecutive quarters of decline in GDP). The series HIST counts the number of times this is reached at a particular period.

```plaintext
smpl 2006:3 2008:4
set hist = 0.0
do draw=1,ndraws
    simulate(model=gdpmodel)
    do date=2006:3,2008:4
        if gdp(date)<gdp(date-1).and.gdp(date-1)<gdp(date-2) {
            compute hist(date)=hist(date)+1
            break
        }
    end do date
end do draws
set hist = hist/ndraws
```

Computing Moments

You compute these using the same formulas you would for any other random sample. To estimate the mean, sum the generated values and divide by the number of draws. To estimate a variance, add up the squares, divide by the number of draws and subtract the squared estimate of the mean. For example, the following computes the expected value of the maximum and minimum values achieved for a log Normal diffusion approximated by simulation over a grid. In this, MU is the expected return per year, SIGMA the volatility, PERIOD the fractional part of a year being analyzed and BASEPRICE the original price. GRID is the number of equally spaced time periods used in the simulation.

```plaintext
compute hdrift=(mu-.5*sigma**2)*(period/grid)
compute hsigma=sigma*sqrt(period/grid)
* 
set price 1 grid = 0.0
compute totalmax=0.0,totalmin=0.0
do draws=1,ndraws
    compute pcond=baseprice
    do i=1,grid
        compute pcond=price(i)=pcond*exp(hdrift+%ran(hsigma))
    end do i
    compute totalmax=totalmax+%maxvalue(price),$
    totalmin=totalmin+%minvalue(price)
end do draws
disp "Maximum" totalmax/ndraws "Minimum" totalmin/ndraws
```
Chapter 13: Simulations/Bootstrapping

Computing Fractiles

Fractiles (quantiles) are needed when you are computing critical values for a test. They are also useful when there is some concern that a distribution is asymmetric; if so, standard error bands computed using the mean and variance may give a misleading picture of the distribution. Unfortunately, there are no sufficient statistics for the fractiles of a distribution: you can’t summarize the data by keeping a running sum as was done in the two examples above. To compute fractiles, you have to save the entire set of draws and do the calculations later. Set up a vector, or possibly a matrix of vectors if you’re keeping track of more than one random variable, and dimension the vectors to the number of draws that you’re making. The instruction `STATISTICS` with the `FRACTILES` option computes a standard collection of fractiles for a data series. You can also use the `%FRACTILES(array, fractiles)` function, which takes as parameters an array of data and a vector of desired fractiles, and returns those fractile points in the data set. This example produces simulations of `TBILLS`. `%FRACTILES` is used to find the 25th and 75th percentiles.

```plaintext
dec vect sims(ndraws)
do draw=1,ndraws
   simulate(model=ratemodel)
   compute sims(draw)=tbills(2006:6)
end do draw
compute ratefract=%fractiles(sims,||.25,.75||)
disp "25-75%iles of T-Bills" "to" ratefract(2)
```

Computing a Density Function

Just as with fractiles, you need to save all the simulated values. Here, they must be saved in a data series, which you process using the instruction `DENSITY`. For instance, the following computes an estimated density function for the autoregressive coefficient when the data actually follows a random walk. The coefficient estimates go into the series `BETAS`. The density is estimated, then graphed.

```plaintext
set betas 1 ndraws = 0.0
do i=1,ndraws
   set(first=0.0) y = y{1}+%ran(1.0)
   linreg(noprint) y
      # y{1}
   compute betas(i)=%beta(1)
end do i
density betas / xb fb
scatter(style=lines)
   # xb fb
```

480  *RATS User’s Guide*
Computing a Covariance Matrix of Coefficients

A covariance matrix of coefficients is a special case of a moment estimator. There are, however, some special tools available in RATS to help you with this. The following set up can be used with no modifications other than replacing the code for generating draws.

\[
\begin{align*}
\text{compute } & \text{[vector] betamean} = \%\text{zeros(number of regressors,1)} \\
\text{compute } & \text{[vector] betacmom} = \%\text{zeros(number, number)} \\
\text{do } & i=1, \text{ndraws} \\
& \quad \text{... code to compute draw for coefficients } \%\text{BETA} \text{ ...} \\
& \quad \text{compute betamean} = \text{betamean} + \%\text{beta} \\
& \quad \text{compute betacmom} = \text{betacmom} + \%\text{outerxx(\%beta)} \\
\text{end do } & i \\
\text{compute betamean} = \text{betamean} / \text{ndraws} \\
\text{compute betacmom} = \text{betacmom} / \text{ndraws} - \%\text{outerxx(betamean)}
\end{align*}
\]

You can print the regression with the new coefficient vector and covariance matrix by using the CREATE, COEFFS and COVMAT options of LINREG. You must also use the option FORM=CHISQUARED, because BETACMOM is a direct estimate of the covariance matrix:

\[
\text{linreg(create, covmat=betacmom, coeffs=betamean, form=chisq)} \quad \ldots.
\]

Weighted Draws

These examples have assumed that all draws receive equal weight. There are techniques described later in the chapter which require placing unequal weights on the draws. The adjustment required for this is fairly simple when you're computing means and other moments: just sum the weight of a draw times the value obtained on that draw, and, at the end, divide by the sum of the weights.

It's not so simple with the fractiles and the density function. In addition to saving all the simulated values (do not multiply by the weights), you need to save the weights for each draw. On DENSITY, include the option WEIGHTS=series of weights. This will do the appropriate weighting of observations. To compute fractiles, use the \%WFRACTILES function instead of \%FRACTILES. This adds a third parameter which is the vector or series of weights. Note that the computation of fractiles for weighted draws can be somewhat time consuming. If you do many draws, it may even take longer than generating the draws in the first place.

\[
\begin{align*}
\text{density(weigths=weights, smpl=weights>0.00, bandwidth=.05) persist } & \$ 1 \text{ draws xx dx} \\
\text{scatter(style=line,}$ & \$ \\
& \quad \text{header="Density of Persistence Measure (alpha+beta)"}) \\
& \# xx dx
\end{align*}
\]
13.2 Bootstrapping and Simulation Tools

In addition to the standard bookkeeping routines, most simulation and bootstrapping programs will make use of one or more of the specialized instructions and functions described below. These are used to generate the random draws and simulations that will be used to compute the moment statistics, probabilities, etc.

SIMULATE, FORECAST and UFORECAST

The instruction SIMULATE forecasts an equation or system of equations, automatically drawing shocks from a Normal distribution. It can handle linear equations or non-linear formulas. The syntax for SIMULATE is very similar to FORECAST, except there are extra options inputting the covariance matrix for the Normal draws (CV=covariance matrix or FACTOR=factor of covariance matrix). You can do random Normal simulations for a single equation using UFORECAST with the SIMULATE option. The variance for the draws for UFORECAST is the one saved with the equation or regression being forecast.

You can also generate data using a FORECAST instruction, by including an INPUT, SHOCKS, PATHS, or MATRIX option. While SIMULATE can only draw from a single multivariate Normal distribution, FORECAST with the MATRIX or PATHS options will let you can feed any set of shocks into your forecasts.

Virtually every simulation or bootstrapping project requires a loop of some kind (usually a DO loop) and various bookkeeping instructions, as was described in the previous section. Most of the bookkeeping is handled using COMPUTE, SET, GSET and EWISE instructions. The following is a stylized example of the use of SIMULATE. This has a five equation model, with three structural equations (formulas X1EQ, X2EQ and X3EQ) and two identities (X4ID and X5ID).

```
vcc(matrix=v)
  # resids1 resids2 resids3
group(vcc=v) model5 x1eq>>fx1 x2eq>>fx2 x3eq>>fx3 $
  x4id>>fx4 x5id>>fx5
smpl 2007:1 2007:12
do draws=1,ndraws
  simulate(model=model5)
  ... bookkeeping ...
end do draws
```
Chapter 13: Simulations/Bootstrapping

Built-In Functions for Generating Random Draws

RATS provides the following built-in functions for taking draws from various univariate real distributions:

- `%ran(x)` Returns a draw from a Normal(0, x^2)
- `%uniform(x1,x2)` Returns a draw from a Uniform(x1,x2)
- `%rangamma(r)` Random gammas with shape parameter r
- `%ranchisqr(df)` Random chi-squared with df degrees of freedom
- `%ranbeta(a,b)` Random beta with shape parameters a and b.
- `%rant(df)` Random Student-t with df degrees of freedom.

You can use SET to fill elements of a series using random draws. For example:

```
set u = %ran(sqrt(%seesq))
```

creates U as a series of random Normals with variance %SEESQ.

You can use COMPUTE for scalars and arrays. For arrays, you generally need to declare and dimension the array before using COMPUTE. However, note that there is a special function `%RANMAT` for the most common operation of this type: it both dimensions a matrix and fills it with standard Normals.

```
compute e = %ran(1.0)
declare symmetric s(4,4)
compute s = %rangamma(mu)
compute r = %ranmat(5,5)
```

E will be a single N(0,1) draw, S will be a 4x4 SYMMETRIC filled with random gammas, and R will be a 5x5 RECTANGULAR filled with N(0,1) draws.

Inverse Method

If a random variable has an invertible cumulative distribution function, you can draw variates easily by just applying the inverse to a uniform (0,1) draw. For instance, the logistic has distribution function

\[ \frac{1}{1 + e^{-x}} \]

If y is a U(0,1) draw, then solve \( y = \frac{1}{1 + e^{-x}} \) to get \( x = \log\left(\frac{y}{1-y}\right) \). Since the distribution is symmetric, you can also get logistic draws with the slightly simpler \( x = \log\left(\frac{1}{y} - 1\right) \).

The following code generates a vector of 100 draws from a standard logistic:

```
dec vect draws(100)
ewise draws(i)=log(1/%uniform(0,1)-1)
```

Do not do \( \log(\%uniform(0,1)/(1-%uniform(0,1))\). Each use of \%uniform will give a different value, when we want the same one in both places. The original inverse formula (with the y’s occurring in two places) can be done either in two steps:
ewise draws(i)=%uniform(0,1)
ewise draws(i)=log(draws(i)/(1-draws(i)))

or in one by using a comma to separate the two parts of the calculation:

ewise draws(i)= y=%uniform(0,1),log(y/(1-y))

Another example of the use of the inverse method is the Cauchy, which can be generated using

ewise draws(i) = tan(%pi/2.0*%uniform(-1.0,1.0))

Direct Method

Many random variables can be obtained by some function of Normal, uniform, chi-squared, gamma and beta random variables. For instance, another way to get draws from a Cauchy is to take the ratio of two standard Normals:

ewise draws(i)=%ran(1.0)/%ran(1.0)

The inverse method is slightly more efficient in this case, although most efficient is to use the fact that a Cauchy is a $t$ with 1 degree of freedom and use %rant(1).

Multivariate Distributions

To draw a multivariate Normal with mean vector $X$ and covariance matrix $\Sigma$, it is necessary first to get a factor of the $\Sigma$ matrix: a matrix $F$ such that $FF^\prime = \Sigma$. Any such factor will do. The simplest one to obtain with RATS is the Choleski factor computed using the RATS function $\%DECOMP$. The (mean zero) multivariate Normal can then be drawn using the function $\%RANMVNORMAL(F)$.

The following draws a single vector ($XDRAW$) from this distribution. If, as is typical, the actual draws would be inside a loop, the first two lines can be outside it:

dec rect f
compute f = %decomp(sigma)
compute xdraw = x+%ranmvnormal(f)

Note that there are some specialized functions for drawing multivariate Normals whose means and covariance matrices are derived from standard calculations. The most important of these is $\%RANMVPOST(H1,B1,H2,B2)$, which takes a pair of mean vectors ($B$) and precision matrices ($H$) and draws from the multivariate Normal posterior derived by combining them.

To get a draw from a multivariate Student-$t$, you need to draw a full vector using $\%RANT$, then premultiply it by a factor of the scale matrix. If $N$ is the size of the target, then
dec vect u(n)
compute f=%decomp(sigma)
compute u=%ran(df)
compute tdraw=x+f*u

Note that $SIGMA$ is not the covariance matrix of the multivariate $t$-distribution; it’s the covariance matrix of the underlying Normal. The variance of the distribution will be $SIGMA^*(DF/(DF-2))$.

A Wishart distribution is a multivariate generalization of a gamma and is used in modelling a covariance matrix. It specifies a density function for symmetric $N \times N$ matrices. The $%RANWISHART(N,x)$ function in RATS takes two parameters: the first is the dimension of the output matrix ($N$) and the second is the degrees of freedom ($x$). It produces an $N \times N$ matrix whose expected value is $x$ times an $N \times N$ identity matrix. More commonly, however, the “scale matrix” isn’t the identity, but a covariance matrix estimated from the sample. As is the case with the multivariate Normal, you start with a factor of this scale matrix. The two functions available for more general Wishart draws are $%RANWISHARTF(F,x)$ and $%RANWISHARTI(F,x)$. The first draws a Wishart, the second an inverse Wishart. The latter is more commonly used, as it arises in standard Bayesian methods applied to multivariate regressions. The kernel of the density function from which $%RANWISHARTI$ draws $\Sigma$ is

$$
|\Sigma|^{-x-k-1/2} \exp\left(-\frac{1}{2} tr\left(FF'\Sigma^{-1}\right)\right)
$$

where $k$ is the dimension of $\Sigma$. For instance, if the posterior for $\Sigma$ is

$$
|\Sigma|^{-(T-p-k-1)/2} \exp\left(-\frac{1}{2} tr\left((T\hat{\Sigma})\Sigma^{-1}\right)\right), \text{ by inspection, } x = T - p, \text{ and } FF' = (T\hat{\Sigma})^{-1}.
$$

If $T$ is the number of observations from the most recent regression, and $\hat{\Sigma}$ is $%SIGMA$, the following will draw a matrix $W$ from the posterior:

```r
compute f = %decomp(inv(%nobs*%sigma))
compute w = %ranwisharti(f,%nobs-p)
```

The function $%RANSPHERE(N)$ draws a random “direction” in $N$ space. It draws uniformly from the unit sphere in $N$ dimensions, that is, from $\left\{ x \in \mathbb{R}^N : \|x\| = 1 \right\}$.
Example 13.1 Monte Carlo Test Simulation

This example (MONTEARCH.PRG) does 500 repetitions of a test for ARCH (Engle (1982)). The model simulated is

\[ y_t = \beta_0 + \beta_1 x_t + u_t \]
\[ u_t \sim N\left[0, \sigma^2 \left(1 + \alpha u_{t-1}^2\right)\right] \]

Because the (conditional) variance of \( u \) is not constant, the draws for the model must be generated using the non-linear systems solution procedure. \textsc{simulate} only draws from a fixed distribution, so we have to add a third equation to the model which computes a standard Normal variate \( V \). \textsc{frmls} for \( u \) and \( y \) are then “identities.”

You might think that you could simplify this three-equation setup by using a \%\text{ran} function directly in the \texttt{udef frml}. This, however, will draw a new random number at every pass through the solver, so the model will never converge. \textsc{simulate} draws only a single set of values and solves the model given those.

This program generates the process over the period 2 through 150, starting with \( U(1) = 0 \). Since this starts “cold” from an arbitrary value, we use only the last 100 observations to compute the test statistics.

The test-statistics (\( TR^2 \)) are stored in the series \texttt{teststat}. At the end, a \texttt{statistics} instruction computes its fractiles.

```plaintext
compute ndraws=10000,useobs=100,endobs=50+useobs
all endobs
set x 1 endobs = t
*
compute sigma=1.0,alpha  =.20
frml(variance=sigma**2) vdef v = 0.0
frml(identity) udef u = v*sqrt(1+alpha*u{1}**2)
frml(identity) ydef y = 2.0 + 3.0*x + u
group archmod vdef udef ydef>>y
*
set u 1 1 = 0.0
set teststat 1 ndraws = 0.0
do draws=1,ndraws
    simulate(model=archmod) 2 endobs-1 2
    linreg(noprint) y 51 endobs resids
    # constant x
    set usquared 51 endobs = resids**2
    linreg(noprint) usquared 52 endobs
    # constant usquared{1}
    compute teststat(draws)=%trsquared
end do draws
stats(fractiles,nomoments) teststat
```
Chapter 13: Simulations/Bootstrapping

13.3 Rejection Method

Sometimes, distributions can’t be derived as simple functions of the elementary distributions such as normal, gamma and beta. For instance, posterior densities can be the product of two distributions which can’t easily be combined. The Rejection method is a tool which can often be used in univariate cases to generate draws. The Rejection (or Acceptance–Rejection) method is used within RATS to generate draws from the normal and gamma distributions.

We’ll look first at a trivial example. Suppose that you need to take draws from a $N(\mu, \sigma^2)$ truncated to the interval $[a,b]$. An obvious way to do this is to draw $N(\mu, \sigma^2)$ deviates, then reject any that fall outside $[a,b]$. The accepted draws would have the desired distribution, but this process will be inefficient if the probability of the Normal draw falling in $[a,b]$ is fairly low.

An alternative is to draw a random number $x$ from $U(a,b)$. Also draw a random number $z$ from $U(0,1)$. Compare $z$ with the ratio between $f_N(x|\mu,\sigma^2)$ and the maximum that $f_N(x|\mu,\sigma^2)$ achieves on $[a,b]$. If $z$ is less, accept $x$; if not, reject it. This trims away the uniform draws to match the shape of the Normal. This will be more efficient if the density function is fairly constant on $[a,b]$, so that the comparison ratio is close to one and almost all draws are accepted.

A stylized procedure for doing the rejection method is

```
loop
    compute x=draw from the proposal density
    compute a=acceptance function
    if %uniform(0.0,1.0)<a
        break
end loop
```

The value of $x$ when the loop breaks is the draw.

If you know that you have set this up correctly, and you know that the probability of accepting a draw is high, then you can adopt this code as is. However, this will loop until a draw is accepted, and, if you make a bad choice for the proposal density, you might end up with an acceptance function which produces values very close to zero, so this could loop effectively forever. A “failsafed” alternative is

```
do trys=1,maximum_tries
    compute x=draw from the proposal density
    compute a=acceptance function
    if %uniform(0.0,1.0)<a
        break
end do trys
```

This will quit the loop if it fails to accept a draw in MAXIMUM_TRIES. How that limit should be set will depend upon the situation. If you set it to 100 and find that you’re routinely hitting the limit, there is probably a better way to generate the draws.
In order to apply the rejection method to draws from the density \( f \), you need to be able to write

(1) \( f(x) \approx \alpha(x)g(x) \)

where the proposal density \( g(x) \) is one from which we can conveniently make draws and

(2) \( 0 \leq \alpha(x) \leq 1 \)

In our two examples for drawing from the truncated Normal:

(3a) \( g = N(\mu, \sigma^2), \alpha = I_{[a,b]} \)

(3b) \( g = U(a,b), \alpha = f_N(x | \mu, \sigma^2)/\max_{[a,b]} f_N(\bullet | \mu, \sigma^2) \)

Any density for which \( f(x)/g(x) \) is bounded will (theoretically) work as a proposal. As a general rule, you need to choose a distribution which has tails as thick (or thicker) than the target density. For instance, if you use a Cauchy proposal, you will get a small percentage of very extreme draws. However, if \( f \) is a thin-tailed distribution, \( f/g \) will be very small out there, and the extreme draws will be rejected. In other words, the rejection procedure thins out the Cauchy tails by rejecting most tail draws. If, however, \( f/g \) is quite large in the tails, the only way we can “thicken” the tails is by accepting the tail draws and rejecting most of the draws near the mode.

Note that \( \alpha(x) \) doesn’t need to hit the maximum value of 1 in order to be used. So long as \( f(x)/g(x) \) is bounded, we could always choose

(4) \( \alpha(x) = (f(x)/g(x))/\max\{f(\bullet)/g(\bullet)\} \)

(where the max is taken over the support of \( f \)) and that will give the most efficient rejection system for that particular choice of \( g \). However, the rejection method is often applied in situations where the distribution changes slightly from one draw to the next. It’s quite possible that you would spend more time computing the maximum in order to achieve “efficient” draws than it would take to use a simpler but less efficient choice of \( \alpha(x) \). For instance, if we look at (3b) above, we could use

(5) \( g = U(a,b), \alpha = \exp(-(x-\mu)^2/2\sigma^2) \)

If \( \mu \in [a,b] \), this will be exactly the same as (3b). If it isn’t, then this will reject more draws than (3b), since \( \alpha(x) \) is strictly less than one for all the x’s that will be generated by the proposal density. Whether the extra work of finding the normalizing constant in (4) is worth it will depend upon the situation. Here, the added cost isn’t high because the maximum will be at one of \( \{\mu,a,b\} \). If, however, \( f \) and \( g \) are non-trivial density functions, the maximum might very well only be found by some
iterative root finding technique like Newton’s method. Since finding the normalizing constant needs to be done just once per set of draws, the extra work will generally pay off when you need many draws from the same distribution.

The following general strategy works in quite a few situations where a density is formed by multiplying a Normal by a non-Normal continuous density. The log density for a $N(\mu, \sigma^2)$ can be written

\[
\log p(x) = -\frac{a}{2}x^2 + bx + c, \quad \text{where} \quad \sigma^2 = 1/a, \mu = b/a
\]

If the non-Normal density can be approximated with an expansion around $x_0$ as

\[
\log q(x) = -\frac{d}{2}(x - x_0)^2 + e(x - x_0) + f = -\frac{d}{2}x^2 + (e + dx_0)x + f^*, \quad \text{then}
\]

\[
\log p(x)q(x) = \log p(x) + \log q(x) = -\frac{1}{2}(a + d)x^2 + (b + e + dx_0)x + (f^* + c)
\]

which (by inspection) is

\[
N\left(\frac{b + e + dx_0}{a + d}, \frac{1}{a + d}\right)
\]

In the Bayesian analysis of the stochastic volatility model (see Jacquier, Polson and Rossi (1994)), it’s necessary to get draws from the following density (kernel) on the log variance $z$ ($y$ is observed data):

\[
\exp(-z/2)\exp\left(-.5y^2\exp(-z)\right)\times\exp\left(-\left(z - \mu\right)^2/2\sigma^2\right)
\]

The final factor is just a Normal, but the first two make up an inverse gamma on $\exp(z)$. The combined density can’t be simplified into a more basic density. The authors of the paper proposed a Markov Chain Monte Carlo procedure (see Section 13.8) to draw from this. Tsay (2005, section 12.7) approximates the density with a grid (Section 13.9). However, a simpler rejection method can be used instead, as described in Kim, Shephard and Chib (1998). If we write

\[
\log q(x) = -\frac{1}{2}z - \frac{y^2}{2}\exp(-z)
\]

this is a globally concave function. As such, it lies below its tangent line at any expansion point, so

\[
\log q(z) \leq \log q^*(z) \equiv -\frac{1}{2}z_0 - \frac{y^2}{2}\exp(-z_0) + \left(-\frac{1}{2} + \frac{y^2}{2}\exp(-z_0)\right)(z - z_0), \quad \text{and}
\]

\[
p(x)q(x) = p(x)q^*(x)\left(q(x)/q^*(x)\right)
\]
Chapter 13: Simulations/Bootstrapping

Thus, we can use \( p(x)q^*(x) \) as the proposal density with acceptance function 
\( q(x)/q^*(x) \). In (6) and (7), \( p \) and \( q^* \) have

\[
(14) \quad a = \frac{1}{\sigma^2}, b = \frac{\mu}{\sigma^2}, d = 0, e = \left( -\frac{1}{2} + \frac{y^2}{2} \exp(-z_0) \right), \text{ so the proposal density is}
\]

\[
(15) \quad N\left( \mu - \frac{\sigma^2}{2} + \frac{\sigma^2 y^2}{2} \exp(-z_0), \sigma^2 \right)
\]

This will work regardless of the choice of \( z_0 \), but might be quite inefficient if the expansion point is chosen poorly. Kim, Shephard and Chib suggest using \( z_0 = \mu \). If the shape of the combined density were dominated by the Normal factor (as it will often be if it were a Normal likelihood combined with a non-Normal prior), that would probably work quite well. However, that’s not necessarily the case here. Both densities are functions of the data and an outlier value for \( y \) can make the mean of the Normal factor an inefficient expansion point. (We’ve found that 1000 tries might not be enough for some data points). With a bit of extra calculation, it’s possible to refine the expansion by moving closer to the mode of the final density. This is done by taking a few Newton-Raphson steps, using a quadratic expansion, before using the linear expansion when actually drawing random numbers. (You can’t use the quadratic expansion for the rejection method because the approximation won’t necessarily lie above the true density).

```plaintext
compute zsqterms=1.0/sigmasq
compute zterms =mu/sigmasq
compute z0=mu
do iters=1,10
    compute zsqadjust=y**2*exp(-z0)*.5
    compute zadjust =-.50+.50*y**2*exp(-z0)*(1+z0)
    compute zvar =1.0/(zsqterms+zsqadjust)
    compute zmean = (zterms+zadjust)*zvar
    if abs(z0-zmean)/sqrt(zvar)<.001
        break
    compute z0=zmean
end do iters

compute zsqadjust=0.0
compute zadjust =-.50+.50*y**2*exp(-z0)
compute zvar =1.0/(zsqterms+zsqadjust)
compute zmean = (zterms+zadjust)*zvar
do trys=1,1000
    compute zdraw=zmean+%ran(sqrt(zvar))
    compute a=exp(-.50*y**2*$
        (exp(-zdraw)-exp(-z0)*(1+z0)+zdraw*exp(-z0)))
    if %uniform(0.0,1.0)<a
        break
end do trys
```
13.4 Standard Posteriors

**Standard Normal Linear Model - Coefficients**

The natural prior for the coefficients in the model

\[ y = X\beta + u, \ u \mid X \sim N(0, h^{-1}I) \]

takes the form

\[ \beta \sim N(\hat{\beta}, \tilde{\mathbf{H}}) \]

For many reasons, these priors are usually stated in terms of their precision \((H)\) rather than the variance. \(H\) is the generalized inverse of \(H\), which allows the prior to be uninformative (have “infinite” variance) in some dimensions or directions. The data evidence from (1) is summarized as

\[ \beta \sim N(\hat{\beta}, h^{-1}(XX)^{-1}) \]

The posterior from combining (2) and (3) is

\[ \beta \sim N\left(\frac{\hat{\beta}}{h}, X'X \right) \]

RATS offers two convenience functions for generating draws from the distribution in (4): %RANMVPOST and %RANMVPOSTCMOM. %RANMVPOST \((H1, B1, H2, B2)\) takes the precision matrices \((H1\) and \(H2)\) and their corresponding means \((B1\) and \(B2)\) and generates a draw. %RANMVPOSTCMOM \((CMOM, h, H2, B2)\) assumes that \(CMOM\) is the cross moment matrix of \(X, y\) (\(X\) variables first), \(h\) is the \(h\) (reciprocal of residual variance) from (1) and \(H2\) and \(B2\) are the precision and mean from (2).

**Standard Normal Linear Model - Variance**

The natural prior for the reciprocal of the variance of \(u\) is

\[ \nu s^2 h \sim \chi^2_v \]

(Note: \(s^2\) is a scale parameter for the prior). The data evidence from (1) for \(h\) is summarized as

\[ \left( y - X\beta \right)' \left( y - X\beta \right) h \sim \chi^2_T \]

(5) and (6) combine to produce the posterior

\[ \left( y - X\beta \right)' \left( y - X\beta \right) + \nu s^2 \sim \chi^2_{T+v} \]
If we call the term in parentheses RSSPLUS, we can get a draw for $h$ with

\[
\text{compute } hu = \frac{\%ranchisqr(nu+%nobs)}{rssplus}
\]

Note that this draws the reciprocal of the variance. That’s what $\%\text{RANMVPOSTCMOM}$ wants as its second parameter.

The sum of squared residuals $(y - X\beta)'(y - X\beta)$ can be computed using $\text{SSSTATS}$ if you have the residuals already computed (into, say, the series $u$):

\[
s\text{stats} / u**2 >> rssbeta
\]

In most cases, however, it will be easier to use the convenience function $\%\text{RSSCMOM(CMOM,Beta)}$. Like $\%\text{RANMVPOSTCMOM}$, $\%\text{RSSCMOM}$ works with the cross product matrix of $X,y$ ($X$ variables first). A complete sequence of drawing $h$ given $\beta$ and drawing $\beta$ given $h$ is:

\[
\begin{align*}
\text{compute } & \quad \text{rssplus} = nu * s2 + \%\text{rsscmom(cmom,beta)} \\
\text{compute } & \quad hu = \frac{\%ranchisqr(nu+%nobs)}{rssplus} \\
\text{compute } & \quad beta = \%\text{ranmvpostcmom(cmom,hu,priorh,priorb)}
\end{align*}
\]

**Multivariate Normal Regression (VAR)-Jeffreys Prior**

Write a vector autoregression (or any multivariate regression with identical explanatory variables) as

(8) \[ y_t = X\beta + u_t, \quad t = 1, \ldots, T \]

$X_t$ is a $p$-vector of explanatory variables, $y_t,u_t$ are $n$-vectors and $\beta$ is the $p \times n$ matrix of coefficients. (We use a matrix for $\beta$ in this discussion as much as possible since that’s the arrangement that is used by the functions $\%\text{MODELGETCOEFFS}$ and $\%\text{MODELSETCOEFFS}$). The $u_t$ are assumed i.i.d. over time with distribution $N(0,\Sigma)$.

Let the OLS estimates for $\beta$ and $\Sigma$ be $B$ and $S$. With the standard diffuse (Jeffreys) prior of

(9) \[ f(\beta,\Sigma) \propto |\Sigma|^{(n+1)/2} \]

the posterior is

(10) \[ \Sigma \sim IW[(TS)^{-1},T-p] \]

and, given $\Sigma$,

(11) \[ vec(\beta) \sim N[vec(B),\Sigma \otimes (X'X)^{-1}] \]

$\Sigma$ has an inverse Wishart distribution, which can be drawn using the function $\%\text{RANWISHARTI}$. This takes as its arguments a factor of $(TS)^{-1}$ and the degrees of freedom. If the VAR is estimated using $\text{ESTIMATE}$, the following will generate a draw for $\Sigma$. 

492  *RATS User’s Guide*
compute fwish =%decomp(inv(%nobs*%sigma))
compute wishdof=%nobs-%nreg
compute sigmad =%ranwisharti(fwish,wishdof)

To draw from (11), note that

\[(12) \quad \left(F_x \otimes F_{xx}\right)\left(F_x \otimes F_{xx}\right)^\prime = \Sigma \otimes XX \quad where \quad F_x F_x^\prime = \Sigma \quad and \quad F_{xx} F_{xx}^\prime = \left(X'X\right)^{-1}\]

so a factor of the (possibly) very large covariance matrix can be obtained from factors of the component matrices. And $F_{xx}$ is a function just of the observable data, and so can be computed just once. Given the component factors, the function %RANMVKRON(FSIGMA,FXX) will draw a $p \times n$ matrix which is the “unvec”ed draw from (11) with a mean of zero. Adding the $B$ matrix will give a draw for $\beta$. The following sequence draws $\Sigma$ (as SIGMAD) and $\beta$ (as BETADRAW). The first five lines depend only upon the data and can (and should) be done outside the simulation loop. Only the last three need to be done within it.

\[
\begin{align*}
\text{estimate} \\
\text{compute fwish } &= \text{%decomp(inv(%nobs*%sigma))} \\
\text{compute wishdof } &= \text{%nobs-%nreg} \\
\text{compute fxx } &= \text{%decomp(%xx)} \\
\text{compute betaols } &= \text{%modelgetcoeffs(varmodel)} \\
\text{compute sigmad } &= \text{%ranwisharti(fwish,wishdof)} \\
\text{compute fsigma } &= \text{%decomp(sigmad)} \\
\text{compute betadraw } &= \text{betaols} + \text{%ranmvkron(fsigma,fxx)}
\end{align*}
\]

### Multivariate Normal Regression (VAR)-General Priors

The most time-consuming part of drawing a multivariate Normal vector of high dimension is factoring the covariance matrix. If we have a six variable VAR with 20 coefficients per equation, factoring the $120 \times 120$ covariance matrix and generating a draw based upon that takes roughly 30 times as long as using %RANMVKRON with the $20 \times 20$ component already factored. And this gap gets larger as the size of the covariance matrix increases. The Kroneker product form saves time both by avoiding factoring a full-sized matrix, and also by taking advantage of the structure of the matrix in multiplying out to get the final draw.

However, unless the prior on the coefficients takes a very specific form, the posterior covariance matrix will not have a convenient structure (see Kadiyala and Karlsson (1997)). And $\Sigma$ will not have an unconditional distribution as it does in (10).

If we keep the standard diffuse prior for $\Sigma$, but now have an informative multivariate Normal prior for $\beta$:

\[(13) \quad f(\beta, \Sigma) \propto |\Sigma|^{(n+1)/2} \exp\left(-\frac{1}{2} vec(\beta - \overline{B})' \overline{H} vec(\beta - \overline{B})\right)\]
then, the posterior for $\Sigma$ is given by

$$
\Sigma | \beta \sim IW\left((Y - X\beta)'(Y - X\beta), T\right)
$$

and the posterior for $\text{vec}(\beta) | \Sigma$ is a multivariate Normal with precision

$$
\Sigma^{-1} \otimes XX + \bar{H} \quad \text{and mean}
$$

$$
\left(\Sigma^{-1} \otimes XX + \bar{H}\right)^{-1}\left((\Sigma^{-1} \otimes XX)\text{vec}(B) + \bar{H}\text{vec}(\bar{B})\right)
$$

While it’s possible to apply %RANMVPOST to this, there is an even greater advantage in using a pair of cross-moment based functions to avoid unnecessary calculations. These are %SIGMACMOM(CMOM, B), which computes $(Y - X\beta)'(Y - X\beta)$ from (14) and %RANMVKRONCMOM(CMOM, SINV, H, B), which, given $\Sigma^{-1}$, generates a draw from the posterior given by (15) and (16). Note that if the B argument (the prior mean) in %RANMVKRONCMOM is $p \times n$, so will be the draw; while if B is stacked in vector form, the draw will be as well. The cross product matrix in this case should have the X variables first, then all the Y’s. This is the order that CMOM(MODEL=varmodel) will create. The following will create BDRAW given SIGMAD and SIGMAD given BDRAW. The CMOM will be outside the loop, the others inside of it. Note that this is an example of Gibbs sampling (Section 13.7).

```r

load('cmom.r')
cmom(model=varmodel)

compute bdraw = %ranmvkroncmom(cmom, inv(sigmad), hprior, bprior)
compute rssmat = %sigmacmom(cmom, bdraw)
compute sigmad = %ranwisharti(%decomp(inv(rssmat)), %nobs)
```

### Probabilities

The natural prior for a binomial random variable with unknown probability $p$ is the beta. If you have data with $n$ observations and $k$ successes, and a beta prior with parameters $a, b$, the posterior is beta with parameters $a + k, b + n - k$. In most cases, the easiest way to get the value of $k$ is with the instruction SSTATS applied to a logical or relational expression. For instance, the following counts (and puts into the variable COUNT) the number of times that $S_{i-1}$ was 1 and $S_i$ was 2. The number of observations in the sample, that is, the number of times $S_{i-1} = 1$ will be put in %NOBS. The COMPUTE then makes a draw from the posterior combining this with a beta prior with parameters $\gamma_1, \gamma_2$.

```r

sstats(smpl=(s{1}=1)) 2 * s==2>>count
compute e1 = %ranbeta(gamma11+count, gamma12+%nobs-count)
```
13.5 Monte Carlo Integration

Monte Carlo integration uses randomization techniques to compute approximations to the value of integrals for which analytical methods can’t be applied and standard numerical methods, such as quadrature, are slow and difficult. Suppose the integral to be computed can be written

\[ I = \int h(x) f(x) \, dx \]

where \( f(x) \) is a density function for a (convenient) probability distribution. Then \( I \) is the expected value of \( h \) over that distribution. If \( h \) is well-behaved, then the law of large numbers will apply to an independent random sample from the density \( f \), and \( I \) can be approximated by

\[ \hat{I} = \frac{1}{n} \sum h(x_i) \]

This approximation improves with the size of the random sample \( n \). The accuracy of the Monte Carlo estimates is governed by the Central Limit Theorem: if we use

\[ \frac{1}{n} \sum h^*(x_i) \]

to approximate \( E h(x) \)

the variance of the numerical error is approximately

\[ \frac{1}{n} \text{var } h^* \]

While this process can be applied outside of statistical work (need to integrate a function \( g \) – find a clever \( hf \) combination), it is mainly used in Bayesian statistics where a complex posterior distribution can be “mapped out” by reporting the expected values of interesting functions. The practical examples in the earlier parts of this chapter are actually all fairly trivial instances of Monte Carlo integration. For instance, a probability is just the expected value of an indicator variable on a particular event: \( h \) is one or zero depending upon whether the draw for \( x \) satisfies a condition.

The following prices a vanilla European call using Monte Carlo, estimates the standard deviation of the error in this calculation, and compares it with the Black-Scholes value. Because only the value at expiration matters, this can be simulated without generating a full path for the security price. This is on the example file MCPriceEurope.rat.

```rat
compute expire=5.0/12.0, strike=52.0, price=50.0, rate=.1, sigma=.40
compute hdrift = (rate-.5*sigma**2)*expire
compute hsigma = sigma*sqrt(expire)
compute ndraws = 10000
compute total=0.0, total2=0.0
```
Chapter 13: Simulations/Bootstrapping

```
do draws=1,ndraws
    compute payoff = exp(-rate*expire)*$
    %max(0,price*exp(hdrift+%ran(hsigma))-strike)
    compute total  = total+payoff,total2=total2+payoff**2
end do draws
compute mcvalue = total/ndraws
compute mcvar   = (total2/ndraws-mcvalue**2)/ndraws
disp "Value by MC" total/ndraws "Std Dev" sqrt(mcvar)
*
compute d1 = (log(price/strike)+expire*(rate+.5*sigma**2))/$
    (sigma*sqrt(expire))
compute d2 = d1-sigma*sqrt(expire)
*
compute value = price*%cdf(d1)-strike*exp(-rate*expire)*%cdf(d2)
disp "Value by Black-Scholes" value
```

### Antithetic Acceleration

In antithetic acceleration, we use a clever choice of $h^*$ to get a reduction in the variance. Suppose that the density $f$ is symmetric around $x_0$. Because of the symmetry, the function

$$h^*(x) = (1/2)(h(x) + h(2x_0 - x))$$

has the same expected value as $h$. And its variance will be lower as long as there is a correlation between $h(x)$ and $h(2x_0 - x)$. In particular, if $h$ is linear in $x$, var$h^*$ is zero: every draw for $x$ will produce $h^*(x) = h(x_0)$, which is, of course, the correct expected value. If $h$ is close to being linear over the region where $f$ is high, we would see a very substantial reduction in variance for the antithetical method versus independent draws.

The option pricing example above seems a good candidate for antithetic acceleration, as flipping the sign of the draw will change a high security price to a low one and vice versa. Since you’re doing two function evaluations per draw, antithetic acceleration shows signs of working if the variance is smaller than 1/2 the standard MC.

```
compute total=0.0,total2=0.0
do draws=1,ndraws
    compute u = %ran(hsigma)
    compute payoff1 = exp(-rate*expire)*$
        %max(0,price*exp(hdrift+u)-strike)
    compute payoff2 = exp(-rate*expire)*$
        %max(0,price*exp(hdrift-u)-strike)
    compute total  = total +.5*(payoff1+payoff2)
    compute total2 = total2+.25*(payoff1+payoff2)**2
end do draws
compute mcvalue = total/ndraws
compute mcvar   = (total2/ndraws-mcvalue**2)/ndraws
disp "Value by MC-Antithetic" total/ndraws "Std Dev" sqrt(mcvar)
```
Example 13.2 Monte Carlo Integration

The impulse responses from a VAR are highly non-linear functions of the coefficients. Monte Carlo integration is one way to examine the distribution of these, so we can properly access the statistical significance of the point values that are generated by IMPULSE.

This applies the techniques of Section 13.4 for the VAR with the diffuse prior, and uses the basic Choleski factorization to identify the shocks. This uses antithetic acceleration by drawing a new value of $\Sigma$ and $\beta$ on the odd draws, then flipping $\beta$ around the OLS estimates (the mean of the posterior) on the even draws.

It can be adapted to any just–identified structural VAR by replacing the decomposition of the draw for the $\Sigma$ matrix by the calculation of the factorization. It can’t be used for an overidentified structural VAR; that requires the use of importance sampling (Section 13.6). For some just-identified factorizations which can’t be done with simple matrix operations (if you need, for instance, CVMODEL to calculate the parameters), you may actually be better off also using importance sampling.

This can also be adapted to be used with other priors (at a substantially higher computational cost). As is the case with many applications of simulation techniques, there is actually more programming involved in extracting the information from the simulations than in doing the simulations themselves. At the end of the program, we do a rather fancy set of graphs to display the output. This will generate an $n \times n$ “matrix” of graphs, with the responses of a variable in a row, the response to a variable in a column. Following the recommendation of Sims and Zha (1999), this computes fractiles, not standard errors. The 16% and 84% fractiles correspond to one standard deviation if we were doing symmetrical error bands based upon estimates of the variance.

This is MONTEVAR.PRG. Note that most simple applications of this can be done using the procedure MONTEVAR.SRC. This program should be used as a guide if you need to adapt its techniques.

```
compute lags=4 ;*Number of lags
compute nstep=16 ;*Number of response steps
compute nkeep=2500 ;*Number of keeper draws
*
open data haversample.rat
cal(q) 1959
data(format=rats) 1959:1 2006:4 ftb3 gdph ih cbhm
*
set loggdp = log(gdph)
set loginv = log(ih)
set logc = log(cbhm)
*
system(model=varmodel)
variables loggdp loginv logc ftb3
```
lags 1 to lags
det constant
dend(system)
******************************************************************
estimate
compute nvar =%nvar
compute fxx =%decomp(%xx)
compute fwish =%decomp(inv(%nobs*%sigma))
compute wishdof=%nobs-%nreg
compute betaols=%modelgetcoeffs(varmodel)
*
declare vect[rect] responses(nkeep)
declare rect[series] impulses(nvar,nvar)

infobox(action=define,progress,lower=1,upper=nkeep) "$Monte Carlo Integration"
do draws = 1,nkeep
  if %clock(draws,2)==1 {
    compute sigmad =%ranwisharti(fwish,wishdof)
compute fsigma =%decomp(sigmad)
compute betau =%ranmvkron(fsigma,fxx)
compute betadraw=betaols+betau
  }
else
  compute betadraw=betaols-betau
compute %modelsetcoeffs(varmodel,betadraw)
impulse(noprint,model=varmodel,factor=fsigma,$
  results=impulses,steps=nstep)

Store the impulse responses

dim responses(draws)(nvar*nvar,nstep)
ewise responses(draws)(i,j)=$
  impulses((i-1)/nvar+1,%clock(i,nvar))(j)
infobox(current=draws)
end do draws

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
compute ylabel(i)=ll
end do i
infobox(action=remove)

grparm(bold) hlabel 18 matrixlabels 14
grparm axislabel 24
Because we want a common scale for all responses of a single variable, we need to do all the calculations for a full row of graphs first. Note, by the way, that this graph transposes rows and columns from the arrangement used in the original MONTEVAR.

```
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)
```
13.6 Importance Sampling

Monte Carlo integration is fairly straightforward if the density \( f \) is an easy one from which to generate draws. However, many models do not produce the convenient Normal-gamma or Normal-Wishart posteriors that we saw in Section 13.4. One method which can be used for more difficult densities is known as importance sampling. For technical details beyond those provided here, see Geweke (1989).

Importance sampling attacks an expectation over an unwieldy density function using

\[
E_f(h(x)) = \int h(x)f(x)dx = \int h(x)(f(x)/g(x))g(x)dx = E_g(hf/g)
\]

where \( f \) and \( g \) are density functions, with \( g \) having convenient Monte Carlo properties.

The main problem is to choose a proper \( g \). In most applications, it’s crucial to avoid choosing a \( g \) with tails which are too thin relative to \( f \). If \( f \) and \( g \) are true density functions (that is, they integrate to 1), we can use

\[
\hat{h} = (1/n)\sum h(x_i) f(x_i)/g(x_i)
\]

to estimate the desired expectation, where the \( x_i \) are drawn independently from \( g \).

If \( \int |h(x)|f(x)dx < \infty \), then \( \int |h(x)(f(x)/g(x))|g(x)dx < \infty \), so the strong law of large numbers applies to (2), and \( \hat{h} \) will converge a.s. to \( E_f(h(x)) \).

However, while the sample means may converge, if the variance of \( hf/g \) doesn’t exist, the convergence may be extremely slow. If, on the other hand, that variance does exist, the Central Limit Theorem will apply, and we can expect convergence at the rate \( n^{-1/2} \).

To take a trivial example, where the properties can be determined analytically, suppose \( h(x) = x \), and \( f \) is a Normal with mean zero and variance 4. Suppose we choose as \( g \) a standard Normal. Then

\[
f/g = \frac{1}{2} \exp \left( \frac{3}{8} x^2 \right) \quad \text{and} \quad \int \left( h(x)f(x)/g(x) \right)^2 g(x)dx = \int \frac{x^2}{4\sqrt{2\pi}} \exp \left( -\frac{1}{4} x^2 \right) dx = \infty
\]

Convergence of \( \hat{h} \) with this choice of \( g \) will be painfully slow.

Suppose now that \( f \) is a Normal with mean zero and variance 1/4 and again we choose as \( g \) a standard Normal. Now

\[
\int \left( h(x)f(x)/g(x) \right)^2 g(x)dx = \int \frac{4x^2}{\sqrt{2\pi}} \exp \left( -\frac{7}{2} x^2 \right) dx = \frac{4}{7\sqrt{7}} = .216
\]
For this particular $h$, not only does the importance sampling work, but, in fact, it works better than independent draws from the true $f$ density. The standard error of the importance sampling estimate is $\sqrt{\frac{216}{n}}$, while that for draws from $f$ would be $\sqrt{\frac{25}{n}}$. This result depends the shape of the $h$ function—in this case, the importance function gives a lower variance by oversampling the values where $h$ is larger. If $h$ goes to zero in the tails, sampling from $g$ will still work, but won’t do better than draws from $f$. (A thin-tailed $g$ works respectably only for such an $h$.)

The lesson to be learned from this is that, in practice, it’s probably a good idea to be conservative in the choice of $g$. When in doubt, scale variances up or switch to fatter-tailed distributions or both. For instance, the most typical choice for an importance function is the asymptotic distribution from maximum likelihood estimates. You’re likely to get better results if you use a fatter-tailed $t$ rather than the Normal.

All of the above was based upon $f$ and $g$ being true density functions. In reality, the integrating constants are either unknown or very complicated. If we don’t know $f/g$, just $w = f^* / g^* = cf / g$, where $c$ is an unknown constant, then

$$\frac{1}{n} \sum h(x_i)w(x_i) = \frac{1}{n} \sum h(x_i)cf(x_i)/g(x_i) \to cE_f (h(x_i))$$

Using the second of these to estimate $c$ gives the key results

$$\hat{h} = \sum h(x_i)w(x_i)/\sum w(x_i)$$

$$s_h^2 = \left[ \sum (h(x_i)w(x_i))^2 / (\sum w(x_i))^2 \right] - \hat{h}^2 / n$$

There’s one additional numerical problem that needs to be avoided in implementing this. Particularly for large parameter spaces, the omitted integrating constants in the density functions can be huge. As a result, a direct calculation of $w$ can produce machine overflows or underflows. (The typical computer can handle real numbers up to around $10^{500}$.) To avoid this, we would advise computing $w$ by

$$\exp(\log f^*(x) - \log f_{\text{max}}^* - \log g^*(x) + \log g_{\text{max}}^*)$$

where $f_{\text{max}}^*$ and $g_{\text{max}}^*$ are the maximum values taken by the two kernel functions.

Now all of the above shows how to compute the expectation of a measurable function of the random variable $x$. If you want to compute fractiles, you need to use the function %WFRACTILES. Fractiles are estimated by sorting the generated values and locating the smallest value for which the cumulated normalized weights exceeds the requested fractile. The proof of this is in the Geweke article. If you want to estimate a density function, add the WEIGHT option to your DENSITY instruction.
Example 13.3 Importance Sampling

This applies importance sampling to a GARCH model. The log likelihood for a GARCH model has a non-standard form, so it isn’t possible to draw directly from the posterior distribution, even with “flat” priors. Importance sampling is one way to conduct a Bayesian analysis of such a model; we’ll show another in Example 13.5.

The importance function used here is a multivariate Student-$t$, with the mean being the maximum likelihood estimate and the covariance matrix being the estimated covariance matrix from GARCH. This is fattened up by using 5 degrees of freedom.

The true density function is computed using GARCH with an input set of coefficients and METHOD=EVAL which does a single function evaluation at the initial guess values. Most instructions which might be used in this way (BOXJENK, CVMODEL, FIND, MAXIMIZE, NLLS) have METHOD=EVAL options.

This is GARCHIMPORT.PRG.

```
open data haversample.rat
calendar(m) 1957
data(format=rats) 1957:1 2006:12 ftbs3
graph(header="US 3-Month Treasury Bill Rate")
# ftbs3

Estimate a linear regression to get its residual variance for use as the pre-sample value.

linreg ftbs3
# constant ftbs3{1}
compute h0=%sigmasq

Estimate a GARCH with Normally distributed errors

garch(p=1,q=1,reg,presample=h0,distrib=normal) / ftbs3
# constant ftbs3{1}

Set up for importance sampling.
FXX is the factor of the covariance matrix of coefficients
XBASE is the estimated coefficient vector
FBASE is the final log likelihood

compute fxx  =%decomp(%xx)
compute xbase=%beta
compute fbase=%logl

This is the number of draws and the degrees of freedom of the multivariate t distribution from which we’re drawing.

compute ndraws=10000
compute drawdf=5.0

Initialize vectors for the first and second moments of the coefficients
```
compute sumwt=0.0,sumwt2=0.0
calculate [vect] b=%zeros(%nreg,1)
calculate [symm] bxx=%zeros(%nreg,%nreg)
declare vect u(%nreg) betau(%nreg)

This is used for an estimated density of the persistence measure (ALPHA+BETA). We need to keep the values of the draws and the observation weights in order to use the DENSITY instruction.

set persist 1 ndraws = 0.0
set weights 1 ndraws = 0.0

* 
do draws=1,ndraws

Draw a %NREG vector from a t with DRAWDF degrees of freedom, mean 0 and identity scale matrix.

calculate u=%rant(drawdf)

Premultiply by FXX to correct the scale matrix and add the mean to get the draw for the coefficients

calculate betau=xbase+fxx*u

Compute the density of the draw relative to the density at the mode. The log of this is returned by the value of %RANLOGKERNEL() from the %RANT function.

calculate gx=%ranlogkernel()

Compute the log likelihood at the drawn value of the coefficients

garch(p=1,q=1,reg,presamp=h0,method=eval,init=betau) / ftbs3
# constant ftbs3{1}

Compute the difference between the log likelihood at the draw and the log likelihood at the mode.

calculate fx=%logl-fbase

It’s quite possible for %LOGL to be NA in the GARCH model (if the GARCH parameters go sufficiently explosive). If it is, make the weight zero. Otherwise exp(...) the difference in the log densities.

calculate weight=%if(%valid(%logl),exp(fx-gx),0.0)

Update the accumulators. These are weighted sums.

calculate sumwt=sumwt+weight,sumwt2=sumwt2+weight**2
calculate b=b+weight*betau,bxx=bxx+weight*%outerxx(betau)

Save the drawn value for persist and weight.

calculate persist(draws)=betau(4)+betau(5)
calculate weights(draws)=weight
end do draws
Chapter 13: Simulations/Bootstrapping

The efficacy of importance sampling depends upon function being estimated, but the following is a simple estimate of the number of effective draws.

\[
\text{disp "Effective Sample Size" sumwt**2/sumwt2}
\]

Transform the accumulated first and second moments into mean and variance

\[
\text{compute b=b/sumwt,bxx=bxx/sumwt-%outerxx(b)}
\]

Create a table with the Monte Carlo means and standard deviations

\[
\text{report(action=define)}
\]
\[
\text{report(atrow=1,atcol=1,fillby=cols) "a" "b" "gamma" "alpha" "beta"}
\]
\[
\text{report(atrow=1,atcol=2,fillby=cols) b}
\]
\[
\text{report(atrow=1,atcol=3,fillby=cols) %sqrt(%xdiag(bxx))}
\]
\[
\text{report(action=show)}
\]

Estimate the density of the persistence measure and graph it

\[
\text{density(weights=weights,smpl=weights>0.00,bandwidth=.05) }
\]
\[
\text{persist 1 draws xx dx}
\]
\[
\text{scatter(style=line,}$
\]
\[
\text{header="Density of Persistence Measure (alpha+beta)"})
\]
\[
\# xx dx
\]
13.7 Gibbs Sampler

The Gibbs sampler (Gelfand and Smith (1990)) is one of several techniques developed recently to deal with posterior distributions which not long ago were considered to be intractable. Monte Carlo integration for impulse responses (Example 13.2) is able to work well despite the large number of parameters in the underlying vector autoregression because, with the help of a convenient choice of prior, it is fairly easy to make draws from the posterior distribution. Unfortunately, there are very few multivariate distributions for which this is true. Even in a simple linear regression model, all it takes is a slight deviation from “convenience” in the prior to produce a tangled mess in the posterior distribution, making direct draws with the basic toolkit of random Normals and gammas impossible.

The Gibbs sampler can be brought into play when the parameters can be partitioned so that, although an unconditional draw can’t be obtained directly, each partition can be drawn conditional on the parameters outside its partition. The standard result is that if we draw sequentially from the conditional distributions, the resulting draws are, in the limit, from the unconditional distribution. Because this is a limit result, it is common for practitioners to ignore a certain number of early draws (called the “burn-in”) which might not be representative of the unconditional distribution.

If importance sampling (Section 13.6) is able to work successfully in a given situation, it usually should be chosen over Gibbs sampling, because the draws are independent rather than correlated. However, it often becomes quite hard to find a workable importance function as the dimension of the parameter space gets larger.

A General Framework

The following is a general set-up that we’ve found useful for Gibbs sampling.

```plaintext
compute nburn=# of burn-in draws         Note: as written this does nburn+1
compute ndraws=# of accepted draws

Set initial values for parameters. Do any calculations which don’t depend upon
the draws. Initialize bookkeeping information

infobox(action=define,progress,lower=-nburn,upper=ndraws) "$Gibbs Sampler"
do draw=-nburn,ndraws
infobox(current=draw)

    Do the next draw

    if draw<=0
        next

    Update bookkeeping information

end do draw
infobox(action=remove)
```
Chapter 13: Simulations/Bootstrapping

Mean-Variance Blocking

The most common blocking for the Gibbs sampler is between the regression parameters and the variance or precision. In Section 13.4 for the standard Normal linear model, we can draw $\beta$ conditional on $h$ and $h$ conditional on $\beta$. For the VAR with the general prior, we can draw $\beta$ conditional on $\Sigma$ and $\Sigma$ conditional on $\beta$. The following is the most efficient way to do Gibbs draws from a linear regression. Outside the loop, do

\begin{verbatim}
cmom
# x variables y
linreg(cmom) y
# x variables
compute beta=%beta
to calculate the cross moment matrix and get an initial value for the coefficients.
Also, set up the prior mean (PRIORB) and precision (PRIORH) and the degrees of
freedom (NU) and scale factor (S2) for the residual precision. Inside the loop, do

compute rssplus=nu*s2+%rsscmom(cmom,beta)
compute hu     =%ranchisqr(nu+%nobs)/rssplus
compute beta   =%ranmvpostcmom(cmom,hu,priorh,priorb)
\end{verbatim}

Parameter Space Augmentation

Gibbs sampling can be also be used in situations where there are unobservable variables at each time period. For instance, the Markov switching models (Section 12.3) have the unobservable $S_t$ in additional to the model’s other parameters. These are added to the parameter set. Getting draws for regression parameters given the states is quite simple if the model is otherwise linear. The tricky part is getting the draws for the latent variables. This is typically done one at a time; that is, draws are done sequentially (as $t$ goes from 1 to $T$) for

\begin{equation}
S_t | S_1, \ldots, S_{t-1}, S_{t+1}, \ldots, S_T, \theta
\end{equation}

where $\theta$ are the other parameters. Note that this can be very time-consuming. In addition to the cost of doing (at least) $T$ function evaluations per Gibbs sweep, because of the high correlation between the $S$ values (in most cases), the chain also has a high degree of correlation from one step to the next, so it requires a long burn-in time and many draws. (As a general rule, Gibbs sampling works best when the blocks of parameters are close to being independent of each other, and worst when correlated parameters are drawn separately).
**Example 13.4 Gibbs Sampling**

This is based upon the Shiller Smoothness prior (Example 5.5). That example uses the simpler process of “mixed estimation,” which adjusts the prior on the lag coefficients based upon the residual variance. If the prior on the coefficients is kept separate, it is no longer as easy to get the posterior distribution. The program is GIBBS.PRG. Note that most of the program is devoted to saving and processing statistics from the simulation.

```plaintext
open data haversample.rat
calendar(m) 1947
data(format=rats) 1947:1 2007:4 fltg ftb3
set shortrate = ftb3
set longrate = fltg

Set the number of draws.

Set the prior degrees of freedom (NU) and mean (S2) for the precision (reciprocal of the variance) for the regression residuals.

Set the prior precision for the second difference in the coefficients.

compute nburn=100
compute nkeep=5000
compute s2=.50**2.0
compute nu=4.0
compute hb=1.0/(.03**2)

Generate the precision matrix for the prior. For this example, the prior is flat on the constant and zero lag coefficient, and the second differences on the remainder of the lag polynomial are given independent Normal priors with common precision. In general, PRIORH will be the (generalized) inverse of your prior variance.

declare rect dummy(22,26)
declare symm priorh(26,26)
fmatrix(diff=2) dummy 1 3
compute priorh=hb*(tr(dummy)*dummy)
compute priorb=%zeros(26,1)

Compute the cross-moment matrix

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

<table>
<thead>
<tr>
<th>Example 13.4 Gibbs Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is based upon the Shiller Smoothness prior (Example 5.5). That example uses the simpler process of “mixed estimation,” which adjusts the prior on the lag coefficients based upon the residual variance. If the prior on the coefficients is kept separate, it is no longer as easy to get the posterior distribution. The program is GIBBS.PRG. Note that most of the program is devoted to saving and processing statistics from the simulation.</td>
</tr>
<tr>
<td>open data haversample.rat</td>
</tr>
<tr>
<td>calendar(m) 1947</td>
</tr>
<tr>
<td>data(format=rats) 1947:1 2007:4 fltg ftb3</td>
</tr>
<tr>
<td>set shortrate = ftb3</td>
</tr>
<tr>
<td>set longrate = fltg</td>
</tr>
<tr>
<td>Set the number of draws.</td>
</tr>
<tr>
<td>Set the prior degrees of freedom (NU) and mean (S2) for the precision (reciprocal of the variance) for the regression residuals.</td>
</tr>
<tr>
<td>Set the prior precision for the second difference in the coefficients.</td>
</tr>
<tr>
<td>compute nburn=100</td>
</tr>
<tr>
<td>compute nkeep=5000</td>
</tr>
<tr>
<td>compute s2=.50**2.0</td>
</tr>
<tr>
<td>compute nu=4.0</td>
</tr>
<tr>
<td>compute hb=1.0/(.03**2)</td>
</tr>
<tr>
<td>Generate the precision matrix for the prior. For this example, the prior is flat on the constant and zero lag coefficient, and the second differences on the remainder of the lag polynomial are given independent Normal priors with common precision. In general, PRIORH will be the (generalized) inverse of your prior variance.</td>
</tr>
<tr>
<td>declare rect dummy(22,26)</td>
</tr>
<tr>
<td>declare symm priorh(26,26)</td>
</tr>
<tr>
<td>fmatrix(diff=2) dummy 1 3</td>
</tr>
<tr>
<td>compute priorh=hb*(tr(dummy)*dummy)</td>
</tr>
<tr>
<td>compute priorb=%zeros(26,1)</td>
</tr>
<tr>
<td>Compute the cross-moment matrix</td>
</tr>
<tr>
<td>cmom</td>
</tr>
<tr>
<td># constant shortrate{0 to 24} longrate</td>
</tr>
<tr>
<td>Compute the OLS regression to provide an initial value for the beta vector</td>
</tr>
<tr>
<td>linreg(cmom) longrate</td>
</tr>
<tr>
<td># constant shortrate{0 to 24}</td>
</tr>
<tr>
<td>compute beta=betaols=%beta</td>
</tr>
</tbody>
</table>
Specialized code for the bookkeeping is needed in this example. We are keeping track of the following:

1. Full distribution on the intercept
2. Full distribution on the zero lag coefficient
3. Full distribution of the sum of the lag coefficients
4. The first and second moments for all coefficients

1, 2, and 3 each require a series with length equal to the number of draws. To allow us to compute quickly the sum of the lag coefficients, we generate a vector (called SUMMER) which, when dotted with a draw for the coefficients, will give the sum of the lag coefficients.

4 requires simply a pair of series with length equal to the number of coefficients, all elements initialized to zero.

def vect summer(%nreg)
  ewise summer(i)=%if(i>=2,1.0,0.0)
  *
  set inter 1 nkeep = 0.0
  set coeff1 1 nkeep = 0.0
  set sums 1 nkeep = 0.0
  *
  set comoment1 1 %nreg = 0.0
  set comoment2 1 %nreg = 0.0
  *
  infobox(action=define,progress,lower=-nburn,upper=nkeep) $
  "Gibbs Sampler"
  do draw=-nburn,nkeep
    infobox(current=draw)
    Draw residual precision conditional on previous beta
    compute center=nu*s2+%rsscmom(%cmom,beta)
    compute hu =%ranchisqr(nu+%nobs)/center
    Draw betas given hu
    compute beta=%ranmvpostcmom(%cmom,hu,priorh,priorb)
    if draw<=0
      next
    Do the bookkeeping here. For 1, 2, 3, stuff the function of the current draw into the "DRAW" element of the result series.
    compute inter(draw)=beta(1)
    compute coeff1(draw)=beta(2)
    compute sums(draw)=%dot(beta,summer)
Accumulate the sum and sum of squares of the coefficients

\[
\begin{align*}
\text{set comoment1 } & 1 \ %nreg = \text{comoment1}+\beta(t) \\
\text{set comoment2 } & 1 \ %nreg = \text{comoment2}+\beta(t)^2
\end{align*}
\]
end do draw
infobox(action=remove)

\[
\begin{align*}
density \text{ inter } 1 \ \text{nkeep ginter finter} \\
density \text{ coeff1 } 1 \ \text{nkeep gcoeff1 fcoeff1} \\
density \text{ sums } 1 \ \text{nkeep gsums fsums}
\end{align*}
\]

\[
\begin{align*}
s\text{catter(style=lines,window="Posterior for Intercept")} \\
\# \ \text{ginter finter} \\
s\text{catter(style=lines,window="Posterior for Lag 0")} \\
\# \ \text{gcoeff1 fcoeff1} \\
s\text{catter(style=lines,window="Posterior for Sum")} \\
\# \ \text{gsums fsums}
\end{align*}
\]

\[
\begin{align*}
\text{set comoment1 } & 1 \ %nreg = \text{comoment1}/\text{nkeep} \\
\text{set comoment2 } & 1 \ %nreg = \sqrt{\text{comoment2}/\text{nkeep}-\text{comoment1}^2} \\
\text{set upper } & 1 \ %nreg = \text{comoment1}+2.0*\text{comoment2} \\
\text{set lower } & 1 \ %nreg = \text{comoment1}-2.0*\text{comoment2} \\
\text{set ols } & 1 \ %nreg = \beta_{\text{ols}}(t)
\end{align*}
\]

\[
\begin{align*}
\text{graph(number=0,window="Graph of Lag Distribution") 4} \\
\# \ \text{comoment1 2 %nreg} \\
\# \ \text{upper 2 %nreg 2} \\
\# \ \text{lower 2 %nreg 2} \\
\# \ \text{ols 2 %nreg}
\end{align*}
\]
13.8 Metropolis-Hastings

The Gibbs sampler (Section 13.7) requires an ability to generate draws from the conditional distributions. And again, there are many cases where the conditional distributions don’t have convenient Monte Carlo properties. It’s still possible to generate a Markov chain which will converge to the correct distribution, by using techniques which are similar to importance sampling (Section 13.6). The main difference between these is that importance sampling generates independent draws and weights them to achieve the correct distribution. Because the Gibbs sampler is only approximating the distribution at the end of a chain, a weighting system won’t work. Instead, the “weighting” is done by keeping high density draws for multiple sweeps.

While there are quite a few variations of this procedure, there are two principal methods of generating a test draw: you can either draw from a fixed distribution (similar to what is done with importance sampling), or you can draw from a distribution centered around the last draw. The distribution from which you make the draw is known as the proposal distribution or jumping distribution. Let $x$ be the previous draw and $y$ be the test draw. You compute a jumping probability $\alpha$. The “pseudo-code” then would be:

```plaintext
if %uniform(0.0,1.0)<alpha {
    accept y
} else {
    keep x
}
```

It’s possible for $ALPHA$ to be bigger than 1. If that’s the case, then the draw will always be accepted.

**Fixed proposal distribution**

Let $f$ be the target density and $g$ be the proposal density. (These only need to be known up to a constant of integration). Then

$$
(1) \quad \alpha = \frac{f(y)g(x)}{f(x)g(y)}
$$

As in importance sampling, you need to watch for overflows in the calculation. To safeguard against this, compute this (if possible) as

$$
(2) \quad \alpha = \exp\left(\log f(y) - \log f(x) + \log g(x) - \log g(y)\right)
$$
Random Walk Metropolis

If \( y \) is drawn from a density which is conditional on \( x \), let \( g(x, y) \) denote the density for the transition from \( x \) to \( y \). The general formula is

\[
(3) \quad \alpha = \frac{f(y)g(x, y)}{f(x)g(y, x)}
\]

However, if the transition density has \( y \mid x \sim N(x, \Sigma) \) where \( \Sigma \) doesn’t depend upon \( x \), then \( g(y, x) = g(x, y) \), so (3) simplifies to

\[
(4) \quad \alpha = \frac{f(y)}{f(x)}
\]

This is known as Random Walk Metropolis. This simplification will happen whenever \( g(y, x) = g(x, y) \), which generally would mean a Normal or \( t \) centered at the previous draw. Note that if you do any linearization to get the proposal density (as in Section 13.3), that will make the density dependent upon \( x \) for more than the mean. If you do that, you have to do the analogous linearization around \( y \) to be able to compute \( g(y, x) \).

Choosing a Proposal Density

This can be quite difficult, and may require quite a bit of experimentation, particularly if you’re applying this to a multi-dimensioned \( x \). With either type of proposal, if the distribution is too broad, you don’t move very often because most of the draws hit low densities. If it’s too narrow, you might move only very slowly once you’ve landed in a high density area. It’s a good idea to keep track of the percentage of times you jump. If this is too small, it may indicate that you need to make some type of adjustment. But taking many jumps doesn’t necessarily indicate a successful set-up. You can move around quite a bit in one area, but fail to move beyond it.
Example 13.5 Metropolis-Hastings

This uses Metropolis-Hastings for estimating the same GARCH model used in Example 13.3. This blocks the parameters into the mean AR(1) parameters and the GARCH parameters. To illustrate both methods, the AR(1) parameters are handled with a fixed proposal and the GARCH are done by Random Walk Metropolis. This is GARCHGIBBS.PRG.

open data haversample.rat
calendar(m) 1957
data(format=rats) 1957:1 2006:12 ftbs3

Estimate a linear regression to get its residual variance for use as the pre-sample value.

linreg ftbs3
# constant ftbs3{1}
compute h0=%sigmasq

Estimate a GARCH

garch(p=1,q=1,reg,presample=h0,hseries=h) / ftbs3
# constant ftbs3{1}

Pull out the regression coefficients (1 and 2). This forms block one of the Metropolis/Gibbs. Get a decomp of its submatrix of the covariance matrix for use in the proposal density. Do a draw from this to initialize the Gibbs sampler and save the log kernel of the density at the draw.

compute xbeta0=%xsubvec(%beta,1,2)
compute fbeta =%decomp(%xsubmat(%xx,1,2,1,2))
compute xbeta =xbeta0+%ranmvnormal(fbeta)
compute gx =%ranlogkernel()

Pull out the GARCH coefficients (3,4 and 5). This forms block two. Again, get the decomp of its covariance matrix. Because these are being done by Random Walk Metropolis, we don’t need the proposal densities.

compute fgarch=%decomp(%xsubmat(%xx,3,5,3,5))
compute xgarch=%xsubvec(%beta,3,5)
*
compute nburn=1000,nkeep=10000

Initialize vectors for the first and second moments of the coefficients

compute sumwt=0.0,sumwt2=0.0
compute [vect] b=%zeros(%nreg,1)
compute [symm] bxx=%zeros(%nreg,%nreg)
dec vect betau(%nreg) betadraw ygarch
Counters for the number of jumps

compute bjumps=0,gjumps=0
do draws=-nburn,nkeep

Drawing regression parameters. Evaluate the log likelihood (into FX) at the values for XBETA and XGARCH.

garch(p=1,q=1,reg,method=eval,presample=h0,$
   initial=xbeta--xgarch) / ftbs3
# constant ftbs3{1}
set u = %resids
compute fx=%logl

Draw a test vector from the (fixed) proposal distribution. Recalculate the log likelihood (into FY) and fetch the log density at the draw (into GY).

compute ybeta=xbeta0+%ranmvnormal(fbeta)
compute gy=%ranlogkernel()
garch(p=1,q=1,reg,method=eval,presample=h0,$
   initial=ybeta--xgarch) / ftbs3
# constant ftbs3{1}
compute fy=%logl

Compute the jump alpha. If we need to move, reset XBETA and GX, and copy the residuals from Y into U.

compute alpha=exp(fy-fx+gy-gx)
if %uniform(0.0,1.0)<alpha {
   compute bjumps=bjumps+1
   compute xbeta=ybeta,gx=gy
   set u = %resids
}

Evaluate the log likelihood of a GARCH model (into FX) on the residuals at the current settings for XGARCH.

garch(p=1,q=1,nomean,method=eval,presample=h0,$
   initial=xgarch) / u
compute fx=%logl

Draw from the proposal distribution centered around XGARCH and evaluate the log likelihood into FY.

compute ygarch=xgarch+%ranmvnormal(fgarch)
garch(p=1,q=1,nomean,method=eval,presample=h0,$
   initial=ygarch) / u
compute fy=%logl
Chapter 13: Simulations/Bootstrapping

Compute the jump alpha. (Because draws for the GARCH parameters can turn the model explosive, check for an invalid FY).

\[
\text{compute } \alpha = \%\text{if}(\%\text{valid}(fy), \exp(fy-fx), 0.0) \\
\text{if } \%\text{uniform}(0.0, 1.0) < \alpha \{ \\
\quad \text{compute } gjumps = gjumps + 1 \\
\quad \text{compute } xgarch = ygarch \\
\} \\
\text{if } \text{draws} <= 0 \\
\quad \text{next}
\]

Update the accumulators if we're past the burn-in

\[
\text{compute } \beta_{\text{end}} = xbeta - xgarch \\
\text{compute } b = b + \beta_{\text{end}}, bxx = bxx + \%\text{outerxx}(\beta_{\text{end}}) \\
\text{end do draws} \\
\]

\* disp "Percentage of jumps for mean parms" $ \\
\quad 100.0 * \text{float}(bjumps)/(\text{nburn}+\text{nkeep}+1) \\
\quad \text{disp } "Percentage of jumps for GARCH parms" $ \\
\quad 100.0 * \text{float}(gjumps)/(\text{nburn}+\text{nkeep}+1)
\]

Transform the accumulated first and second moments into mean and variance

\[
\text{compute } b = b/\text{nkeep}, bxx = bxx/\text{nkeep} - \%\text{outerxx}(b) \\
\text{report(action=define)} \\
\text{report(atrow=1,atcol=1,fillby=cols) } \"a\" \ "b\" \ \"gamma\" \ \"alpha\" \ \"beta\" \\
\text{report(atrow=1,atcol=2,fillby=cols) } b \\
\text{report(atrow=1,atcol=3,fillby=cols) } \%\sqrt{\%\text{diag}(bxx)} \\
\text{report(action=show)}
\]
13.9 Griddy Gibbs

Approximating a density on a grid will generally be the last resort if you can’t seem to make the rejection method (13.3) and Metropolis-Hastings (13.8) work respectably. The idea behind this is, if you can compute \( f(x_i) \) (the desired univariate density to a constant multiple) on an ordered grid of points \( x_1, \ldots, x_G \), then, using the trapezoidal rule gives us

\[
\int f(x) \, dx \approx I = \sum_{i=1}^{G-1} f(x_i) + f(x_{i+1}) \frac{(x_{i+1} - x_i)}{2}
\]

The summands in this are all non-negative, so they partition the overall sum. Draw a uniform random number from \((0, I)\) and walk through the partial sums until you hit the interval \([x_i, x_{i+1})\) which puts the sum over the value of the random number. The approximate draw is within that interval and is found by solving a quadratic equation. (The approximating density is linear within that interval, so its integral is quadratic.)

The RATS function \%RANGRID(xgrid, fgrid) can do this type of draw given the grid (XGRID vector) and the corresponding densities (FGRID vector). The following is an alternative method for dealing with the stochastic volatility model discussed in Section 13.3. (This is a different parameterization). A draw for \( h \) is needed from

\[
h^{3/2} \exp \left( -\frac{y^2}{2h} \right) \exp \left( -\frac{(\log h - \mu)^2}{2\sigma^2} \right)
\]

The following will do a (quite good) approximation to this:

```
compute ngrid=500
dec vect hgrid(ngrid) fgrid(ngrid)
ewise hgrid(i)=i*(20.0*%variance)/ngrid
ewise fgrid(i)=hgrid(i)**(-1.5)*exp(-y**2/(2*hgrid(i)))*$
    exp(-(log(hgrid(i))-mu)**2/(2*sigma))
compute h=%rangrid(hgrid,fgrid)
```

It does so, however, at quite a high cost of computing time for a grid this fine. And because \( (2) \) is defined on \((0, \infty)\), you have to be sure that the grid goes well out into the tail. Note that \%RANGRID doesn’t require that the grid be equally spaced. In a situation like this, you would be better off with a grid that’s tightly spaced where the density is high (if you know where that is) and a looser one in the tail.

The advantage of using the grid is that it doesn’t require as much ingenuity as the rejection method or Metropolis-Hastings, which are the two main alternatives. You’ll have to figure out what tradeoff you’re willing to accept for human time vs computer time.
13.10 Bootstrapping and Resampling Methods

All of the methods in the previous sections draw data from a continuous distribution. Bootstrapping and other resampling techniques instead reshuffle an existing set of numbers. The key instruction is `BOOT`, which draws a random set of integers. By default, `BOOT` draws with replacement, but with the option `NOREPLACE` will do the draws without replacement.

A draw of random integers will give you the entry numbers you need to sample from a data set. You then use `SET` instructions to generate the resampled data. For instance, the following draws entry numbers between 1950:1 and 2006:12 (the estimation range) and puts them into 2007:1 to 2007:12 of the `SERIES[ INTEGER]` named `ENTRIES`. The series `PATH1`, `PATH2` and `PATH3` will then be random samples from `RESIDS1`, `RESIDS2` and `RESIDS3` over 1950-2006. Note that the three residual series are sampled together. The resampling thus retains any contemporaneous relationship that is present in the data set.

```plaintext
group  model5  x1eq>>fx1  x2eq>>fx2  x3eq>>fx3  x4id>>fx4  x5id>>fx5
smpl  2007:1  2007:12
do draws=1,ndraws
   * Choose random entries between 1950:1 and 2006:12
   boot  entries / 1950:1 2006:12
   set path1 = resids1(entries)
   set path2 = resids2(entries)
   set path3 = resids3(entries)
   forecast(model=model5,paths)
   # path1 path2 path3
   ... bookkeeping ...
end do draws
```

If you have a single equation, and just want a simple bootstrap based upon the residuals, you can use `UFORECAST` with the `BOOTSTRAP` option. For instance, in the example above, if we just wanted to bootstrap `X1EQ`, we could just do

```plaintext
do draws=1,ndraws
   uforecast(boot,equation=x1eq)  fx1  2007:1  2007:12
   ... bookkeeping ...
end do draws
```

Block Bootstrapping

In its simplest use, `BOOT` samples uniformly from the indicated range. This is appropriate when the observations are independent, or when you are resampling some component of the data (typically residuals), that have been transformed to be (approximately) independent. However, in some applications with correlated data, you may not have available a model which will transform the data to independent components. In order to maintain the observed form of dependence, you need to resample the data in blocks. This is done using `BOOT` with the `BLOCK=block size` option. There are three methods for doing block bootstrapping; you choose this using
the **METHOD** option. The default is **METHOD=OVERLAP**. This will allow any block of block size entries from the resampling zone to be chosen. **METHOD=NOOVERLAP** partitions the resampling zone into disjoint blocks and only selects from those. **METHOD=STATIONARY** doesn’t sample by blocks. Instead, when sampling for an entry, it either takes the next item in the resampling zone, or (with probability) \(1/\text{block size}\), starts a new block. With this, the data are resampled with blocks with an expected size of \(\text{block size}\), while in the other two methods, they are exactly \(\text{block size}\).

There are also some applications where it might be desired to oversample more recent values. You can do this with **BOOT** with the option **GEOMETRIC=decay rate**. **decay rate** is the rate at which the probability of being chosen declines, with the last entry in the resampling zone having the highest probability. The closer **decay rate** is to 1, the closer this comes to uniform sampling.

In addition to the **BOOT** instruction, there are several functions which choose a random integer or set of integers.

- **%raninteger(l,u)** draws an integer uniformly from \(\{l, l+1, \ldots, u-1, u\}\)
- **%ranpermute(n)** returns a \text{VECTOR}[\text{INTEGER}] with a random permutation (ordering) of the numbers \(\{1, \ldots, n\}\)
- **%rancombo(n,k)** returns a \text{VECTOR}[\text{INTEGER}] with a random combination of \(k\) values from \(\{1, \ldots, n\}\) drawn without replacement.
- **%ranbranch(p)** returns a randomly selected “branch” from \(\{1, \ldots, \text{dim}(p)\}\) where \(p\) is a \text{VECTOR} giving the (relative) probabilities of the different branches.

For instance, the following draws a “poker hand”, a combination of 5 numbers from 1 to 52:

\[
\text{compute cards}=%\text{rancombo}(52,5)
\]

If \(\text{PPOST}\) is an \(n\)-\text{VECTOR} with relative probabilities of a “break” at a given location in the interval \(\text{lower}\) and \(\text{lower}+n-1\), then

\[
\text{compute break}=%\text{ranbranch}(\text{ppost})+\text{lower}-1
\]

will choose a random break point in that range with probabilities weighted according to the values in \(\text{PPOST}\).
Example 13.6 Bootstrapping a GARCH Model

This does a Value at Risk (VaR) calculation for the dollar/yen exchange rate using a bootstrapped GARCH(1,1) model to generate the simulated returns. (See, for instance, Tsay (2005) for more on calculation of VaR). You can’t simply take bootstrap draws of the residuals from a GARCH process because of the serial dependence of the variance process. Instead, the estimated residuals are standardized by dividing by the square root of their estimated variance, and the GARCH process is simulated out of sample with the bootstrapped standardized residuals scaled up by the simulated variance.

This program is included with RATS on the file GARCHBOOT.PRG.

```
all 6237
open data g10xrate.xls
data(format=xls,org=columns) / usxjpn

   Convert to percent daily returns
set x = 100.0*log(usxjpn/usxjpn{1})

   Estimate the GARCH(1,1) model.
garch(p=1,q=1,resids=u,hseries=h) / x

   Generate a forecasting formula from the results of the GARCH estimation. GSTART and GEND are the regression range, which we need for drawing standardized residuals.
compute gstart=%regstart(),gend=%regend()
compute b0=%beta(1),chat=%beta(2),ahat=%beta(3),bhat=%beta(4)
frml hf = chat+bhat*h{1}+ahat*u{1}**2

   Standardize the historical residuals
set ustandard gstart gend = u/sqrt(h)

   SPAN is the number of periods over which returns are to be computed. NDRAWS is the number of bootstrapping draws
compute span=10
compute ndraws=10000

   Extend out the H series (values aren’t important—this is just to get the extra space).
set h gend+1 gend+span = h(gend)
*  
dec vect returns(ndraws)
```
do draws=1,ndraws

   This draws standardized u’s from the USTANDARD series

   boot entries gend+1 gend+span gstart gend

   Simulate the GARCH model out of sample, scaling up the standardized residuals
   by the square root of the current H.

   set udraw gend+1 gend+span = ustandard(entries)
   set u  gend+1 gend+span = (h(t)=hf(t)), udraw(t)*sqrt(h(t))

   Figure out the cumulative return over the span. As written, this allows for the
   continuation of the sample mean return. If you want to look at zero mean returns,
   take the B0 out.

   sstats gend+1 gend+span b0+u>>returns(draws)
end do draws

   Compute desired fractiles of the returns

   compute [vect] pvals=||.01,.05,.10||
   compute [vect] VaR=%fractiles(returns,pvals)
   report(action=define,hlabels=||"P","VaR/$100"||)
   do i=1,3
      report(atrow=i,atcol=1) pvals(i) VaR(i)
   end do i
   report(action=show)
Example 13.7 Approximate Randomization

Approximate randomization is a technique for testing “unrelatedness” in a fairly general way. It can be applied in place of analysis of variance tests and the like. The null hypothesis to be tested is that some variable $X$ is unrelated to another variable $Y$. The method of attack is to take random permutations of the sample $X$’s. If $X$ is, indeed, unrelated to $Y$, then the actual sample should be fairly typical of the population of permutations. Choose an appropriate test statistic and count the number of times the permuted samples produce a more extreme statistic than the actual sample. (In “exact” randomization, all permutations are examined. That is clearly only possible for very small sample sizes).

As an example, let’s take another look at the heteroscedasticity tests done in Example 6.1. The null hypothesis is that the variance is unrelated to population size. We’ll look at two different choices for the test statistic: the first is the ratio of variances between the two subsamples (which would be testing against the alternative that the variance for small states is greater than those for large ones) and the second is the rank correlation between population and the squared residuals (which tests more generally that the variance decreases with population size).

This is example RANDOMIZE.PRG.

open data states.wks
data(org=obs,format=wks) 1 50 expend pcaid pop pcinc
order(all) pop
set pcexp = expend/pop
linreg pcexp
# constant pcaid pcinc

The first test statistic is the ratio between the sum of the residuals squared over the first 22 observations (small states) to that over the last 22 (large states), skipping the middle six. The second is the correlation between the ranks of the population (which is just the entry number, since the data set is sorted by population) and the squared residual.

set ressqr = %resids**2
sstats / ressqr*(t<=22)>>sum1 ressqr*(t>=29)>>sum2
compute refer1=sum1/sum2
set prank = t
order(rank=vrank) ressqr
compute refer2=%corr(prank,vrank)
COUNT1 and COUNT2 are the number of times we get a more extreme value after reshuffling. We use 999 shuffles.

```
compute count1=count2=0.0
compute ns=999
do draws=1,ns

Use BOOT with NOREPLACE to come up with a new permutation of the RESSQR variable. Recompute the test statistic for these.

boot(noreplace) entries 1 50
set shuffle = ressqr(entries(t))
sstats / shuffle*(t<=22)>>sum1 shuffle*(t>=29)>>sum2
compute teststat=sum1/sum2
compute count1=count1+(teststat>refer1)
order(rank=vrank) shuffle
compute teststat=%corr(prank,vrank)
compute count2=count2+(teststat<refer2)
end do draws

The p-values for the tests are computed by taking \((\text{COUNT}+1)/\text{(DRAWS}+1)\). The +1’s are needed because we are, in effect, adding the actual sample in with the draws and seeing where it ends up.

```

disp "Test 1, p-value" (count1+1)/(ns+1)
disp "Test 2, p-value" (count2+1)/(ns+1)
Chapter 14
Cross Section and Panel Data

While the primary focus of RATS is on time series data and time series techniques, it also offers strong support for analysis of cross section and panel data. The emphasis of this chapter is on specialized techniques which are applied mainly (or solely) to these types of data sets.

Probit and Logit Models
Censored and Truncated Samples
Hazard Models
Panel Data Sets
Fixed and Random Effects
14.1 Overview

This chapter describes special techniques used in the analysis of cross-section and panel data sets. The emphasis is on techniques which are used almost exclusively for such data sets, such as probit models. Cross-section data sets are also more likely than time series data sets to require use of weighted least squares (Section 5.4), tests for heteroscedasticity (Section 6.4), and subsample selections (Section 5.2 and the instruction SMPL).

Below is a brief description of cross-sectional and panel data sets. Topics relating to cross-sectional analysis begin on the next page. For information on working with panel data, see Sections 14.5 and 14.6.

Cross-Sectional Data

Cross-sectional data sets consist of a single observation for a number of individuals (or firms or countries, etc.). Do not use a CALENDAR instruction for cross-sectional data. Set the number of observations to process either with ALLOCATE or on the DATA instruction. For instance, suppose you have data on the number of males and number of females in fifty states. You could either do something like:

allocate 50
open data statepop.dat
data(format=free,org=obs) / males females

or

open data statepop.dat
data(format=free,org=obs) 1 50 males females

By omitting the CALENDAR instruction, you are telling RATS that your data set has no time series properties. RATS will label entries with the entry number alone. RATS will still calculate a Durbin–Watson statistic. This is likely to have little value, but if your data set is ordered on a particular variable, the Durbin–Watson may be able to detect a specification error related to that variable.

Panel Data

Panel data sets have several observations, collected over time, for a number of individuals. They share properties with both time series data sets and cross-sectional data sets. RATS has a special form of the CALENDAR instruction to describe a panel data set. See Section 14.5 for more information.
14.2 Probit and Logit Models

Background

You can use the instruction **DDV** (discrete dependent variables) to estimate probit and logit models. (The instructions **PRB** and **LGT** from earlier versions of RATS are still available). If there are only two choices, the models take the form:

\[
P(Y_i = 1|X_i) = F(X_i\beta)
\]

where the dependent variable \(Y\) takes either the value 0 or 1, \(X\) is the set of explanatory variables and

- \(F(z) = \Phi(z)\) (the standard normal cdf) for the probit
- \(F(z) = \exp(z)/(1+\exp(z))\) for the logit.

See, for example, Wooldridge (2002, section 15.3). Please note that RATS requires a numeric coding for \(Y\). RATS treats any non-zero value as equivalent to “1”.

It’s very useful to look at a framework from which logit and probit models can be derived. Suppose that there is an unobservable variable \(y_i^*\) which measures the “utility” of choosing 1 for an individual. The individual is assumed to choose 1 if this function is high enough. Thus,

\[
y_i^* = X_i\beta + u_i
\]

\[
Y_i = \begin{cases} 
1 & \text{if } y_i^* \geq 0 \\
0 & \text{if } y_i^* < 0 
\end{cases}
\]

If the \(u_i\) are assumed to be independent standard normal, the result is a probit model; if they’re independent logistic, we get a logit. (RATS also allows use of the asymmetric extreme value distribution). This type of underlying model is used to derive many of the extensions of the simple binary choice probit and logit.

Estimation

Probit and logit models require fitting by non-linear maximum likelihood methods, as discussed in Chapter 7. However, their likelihood functions are usually so well-behaved that you can simply let the program take off from the standard initial estimates (all zeros). You may, in bigger models, need to boost the \(\text{ITERS}\) option. But usually all you need is

**ddv(dist=probit) depvar** (use **DIST=LOGIT** for logit)

\# list of regressors

See the instruction **DDV** in the *Reference Manual* for technical details. **DDV** includes the standard options for computing robust covariance matrices, but keep in mind that the estimates themselves may not be consistent if the assumption about the distribution isn’t correct.
Using PRJ

The instruction **PRJ** has many options designed to help with diagnostics and predictions for logit and probit models. In its most basic use, after estimating a model by **DDV**, you can compute the series of the “index” $X_i\beta$ by just

```
prj index
```

With the option **CDF**, you can generate the series of predicted probabilities of the “1” choice. Use the option **DIST=LOGIT** if you want these to be calculated for the logit or **DIST=EXTREME** for the extreme value; the default is to compute these for the normal, regardless of your choice on the **DDV**. Some additional statistics can be obtained using the options **DENSITY**, **MILLS** and **DMILLS**.

**PRJ** will also let you compute the index, density and predicted probability for a single input set of $X$’s. The values are returned as the variables %PRJFIT, %PRJDENSITY and %PRJCDF. The two options which allow this are **XVECTOR** and **ATMEAN**. The **ATMEAN** option requests that the calculation be done at the mean of the regressors over the estimation range, while with **XVECTOR** you provide a vector at which you want the values calculated. This example computes the “slope coefficients” for a probit, giving the derivatives of the probability with respect to the explanatory variables evaluated at the mean.

```
ddv(dist=probit) grade
#  constant gpa tuce psi
prj(atmean,dist=probit)
disp "Slope Coefficients for probit"
disp %prjdensity*%beta
```

Generalized Residuals

With **DDV**, you can compute the *generalized residuals* by including the **GRESIDS** option. If the log likelihood element for an observation can be written $g(X_i\beta)$, the generalized residuals are the series of derivatives $g'(X_i\beta)$. This has the property that

\[
\partial/\partial \beta \sum g(X_i\beta) = \sum g'(X_i\beta)X_i = 0
\]

that is, the generalized residuals are orthogonal to the explanatory variables, the way the regular residuals are for a least squares regression. They crop up in many diagnostic tests on logit and probit models. For instance, the following (from Greene (2003, example 21.4)) computes an LM test for heteroscedasticity related to the series **KIDS** and **FAMINC**. The test statistic determines whether two constructed variables are also orthogonal to the generalized residuals.

```
ddv(gresids=gr) lfp
#  constant wa agesq faminc we kids
prj fit
set z1fit = -fit*kids
set z2fit = -fit*faminc
```
Chapter 14: Cross Section/Panel Data

mcov(opgstat=lm) / gr
# %reglist() z1fit z2fit
cdf(title="LM Test of Heteroscedasticity") chisqr lm 2

Ordered Probit and Logit

In an ordered probit model, there are three or more choices, with the possible choices having a natural ordering. A single index function combined with a partitioning of the real line is used to model the choice process. If we have $m$ choices, let $a_1, ..., a_{m-1}$ be the upper bounds on the intervals (the top choice is unbounded above). If an individual’s index is less than $a_1$, she chooses 1; if between $a_1$ and $a_2$, she chooses 2, etc. An ordered probit occurs if the index function $I$ takes the form

$$I_i = X_i' \beta + u_i$$

where $u_i$ is a standard normal, and an ordered logit if $u$ is a logistic. The probability that an individual chooses $j$ is

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

To estimate an ordered probit using DDV, add the option TYPE=ORDERED. Do not include CONSTANT in the regressors: a non-zero intercept would simply shift the cutpoints, leaving an unidentified model.

This is a part of an example from Wooldridge (2002, 15-5):

```
ddv(type=ordered,dist=probit,cuts=cuts) pctstck
# choice age educ female black married $
  finc25 finc35 finc50 finc75 finc100 finc101 wealth89 prftshr
```

The CUTS option returns the vector of cut points. Only the standard regression coefficients are included in the %BETA vector. Note that RATS does not require that the dependent variable have a specific set of values for an ordered probit; they just have to have a numeric coding in which each choice has a unique value, and these increase in the natural order of the choices. For instance, you could use 1,2,...,$m$, or 0,1,...,$m-1$. However, the labeling on the output for the cutpoints will be CUT (1), CUT (2),..., regardless of the coding that you use.
Chapter 14: Cross Section/Panel Data

Multinomial Logit

There are two main formulations of the logit model for multiple (unordered) choices: the multinomial logit and the conditional logit, though the dividing line between the two is sometimes a bit unclear. The common structure of these is that for each individual there are (linear) functions \( f_1, \ldots, f_m \) of the explanatory variables and it’s assumed that

\[
P(\text{choose } j) = \frac{\exp(f_j)}{\sum_{i=1}^m \exp(f_i)}
\]

In the classic multinomial logit, the \( f \) functions for individual \( i \) take the form

\[
f_j = X_i \beta_j
\]

that is, each choice uses the same explanatory variables, but with a different set of coefficients. With this structure, the coefficients on one of the choices can be normalized to zero, as subtracting (say) \( X_i \beta_1 \) from all the \( f \) values will leave the probabilities unchanged. Thus, the model needs to estimate \( m-1 \) sets of coefficients. To estimate a multinomial logit model, add the option \texttt{TYPE=MULTINOMIAL}. (RATS doesn’t do multinomial probit, so you don’t need the \texttt{DIST} option for logit). For instance, the following is a part of an example from Wooldridge (2002, example 15.4).

\begin{verbatim}
ddv(type=multinomial, smpl=smpl) status
# educ exper expersq black constant
\end{verbatim}

The choices are (numerically) coded into the dependent variable. You can choose whatever scheme you want for this, as long as each choice uses a single number distinct from all other choices. For instance, you can use 0, 1, 2.... or 1, 2, 3,... However you code it, the coefficients are normalized to be zero for the lowest numbered choice.

The alternative to the multinomial logit is the conditional logit model of McFadden (1974). Here the \( f \)’s take the form

\[
f_j = X_j \beta
\]

where \( X_j \) is the set of attributes of choice \( j \) to individual \( i \). Note that the coefficient vector is the same. The idea behind the model is that there is a common “utility function” over the attributes which is used to generate the probabilities. Theoretically, this model could be used to predict the probability of an alternative which hasn’t previously been available (a proposed transit system, for instance), as the probabilities depend only upon the attributes of the choice. If you want to do this, however, you have to be careful about the choice of variables: any choice-specific dummies would render the calculations involving new choices meaningless.
The “X” variables for the conditional logit take quite a different form from those for the multinomial logit and the other models considered earlier. To keep the overall form of the instruction similar to the rest of DDV, the data are input differently: there is one observation in the data set for each combination of an individual and choice. If, for instance, there are four possible choices and 500 individuals, the data set should have 2000 observations. Internally, DDV will compute likelihoods on an individual by individual basis, and will report the number of individuals as the number of usable observations. The dependent variable will be a zero-nonzero coded variable which is non-zero if the individual made the choice indicated by that entry. In order for DDV to calculate the likelihood, you need a separate series which distinguishes the data for the individuals. Use the INDIVS option to show what series that is. The MODES option isn’t required to estimate the model, but allows additional diagnostics; it provides the name of the series which shows which choice an entry represents.

The following is a part of an example from Greene (2003, section 21.7.8). The data set is 210 individuals, 4 choices (air, train, bus, car) for each. The data set itself doesn’t include the individual and mode identifiers, so they are constructed as a function of the entry number. This uses choice-specific dummies, which are also generated. Note that you don’t include CONSTANT, as it would wash out of the probabilities. This also estimates a separate model for the individuals who didn’t choose “air” as part of a Hausman test. In order to do this, you just need to knock out all the “air” entries using a SMPL option. If DDV finds an individual who chose none of the options in the data set being used in the estimation, that observation will be dropped from the calculations, which is what we want.

```
set indiv  = (t+3)/4
set which  = %clock(t,4)
set chosen = mode
*
set dair   = which==1
set dtrain = which==2
set dbus   = which==3
set airhinc = dair*hinc
*
* Conditional logit model
*
ddv(type=conditional,indiv=indiv,modes=which) chosen
# dair dtrain dbus gc ttme airhinc
ddv(type=conditional,indiv=indiv,modes=which,smpl=which<>1) chosen
# dair dtrain dbus gc ttme airhinc
```
Using MAXIMIZE

You can also use the *MAXIMIZE* instruction to estimate probit and logit models that don’t fit into one of the forms that can be handled by *DDV*. The key for doing this is the function *%IF*:

\[
%IF(x, y, z) = \begin{cases} 
y & \text{if } x \neq 0 \\
z & \text{if } x = 0 
\end{cases}
\]

If you create a *FRML* called *ZFRML* which computes the equivalent of \( X_i \beta \), then the log likelihood *FRMLS* for probit and logit are

\[
\begin{align*}
\text{frml probit} &= (z=zfrml(t))\ , \ %if(y,\log(%cdf(z)),\log(%cdf(-z))) \\
\text{frml logit} &= (z=zfrml(t))\ , \ %if(y,z,0.0) - \log(1+\exp(z))
\end{align*}
\]

The easiest way to create *ZFRML* is by doing a *FRML* instruction with either the LASTREG option (after doing a regression) or with the REGRESSORS option.

Samples with Repetitions

In econometric work, we occasionally have a data set where there are repetitions on each set of values for the explanatory variables. Ideally, you will have one series with the number of repetitions on the \( X \) values, and another series indicating either the total number or the fraction of “successes.” You can estimate a probit or logit for such a data set by using *MAXIMIZE*. If series *REPS* and *SUCCESS* are the number of repetitions and the number of successes, the log likelihood for probit is

\[
\text{frml probit} = (z=zfrml(t))\ , \ success*\log(%cdf(z))+(reps-success)*\log(1-%cdf(z))
\]

If the data are limited to the fraction of successes *without* total counts, use

\[
\text{frml probit} = (z=zfrml(t))\ , \ fract\_success*\log(%cdf(z))+(1-fract\_success)*\log(1-%cdf(z))
\]

Heteroscedastic Probit

This estimates a probit model allowing for possible heteroscedasticity in the residual of the “index” process. The scedastic model takes the form

\[
\exp(b_0 \times \text{KIDS} + b_1 \times \text{FAMINC})
\]

\[
\text{frml(regress, vector=bp) zfrml} \\
\# constant wa agesq faminc we kids \\
\text{frml(regress, vector=bh) hfrml} \\
\# kids faminc \\
\text{frml hprobit} = (z=zfrml/sqrt(exp(hfrml))),$ \\
\%if(lfp,\log(%cdf(z)),\log(%cdf(-z))) \\
\text{nonlin bp bh} \\
\text{maximize(pmethod=simplex, piter=10, method=bhhh) hprobit}
\]
Example 14.1 Logit and Probit Models

This computes several logit and probit models for exercise 11.7 in Pindyck and Rubinfeld (1998). It tests the significance of the “Children in School” dummies using both a Wald test (by **EXCLUDE**) and likelihood ratio. This is example **PROBIT.PRG**.

```
open data probit.dat
data(org=obs) 1 95 public1_2 public3_4 public5 private $
  years teacher loginc logproptax yesvm

  Linear probability model

linreg yesvm
  # constant public1_2 public3_4 public5 private $
    years teacher loginc logproptax
  *
  ddev(dist=logit) yesvm
  # constant public1_2 public3_4 public5 private $
    years teacher loginc logproptax
  ddev(dist=probit) yesvm
  # constant public1_2 public3_4 public5 private $
    years teacher loginc logproptax

  Test whether “Children in School” dummies are significant. Use “Wald” test first.

exclude(title="Wald Test of Children in School Dummies")
  # public1_2 public3_4 public5 private

  Likelihood ratio test. We already have the unrestricted model.

compute logunres = %logl
  *
  ddev(dist=probit) yesvm
  # constant years teacher loginc logproptax
compute logres=%logl
  *
compute lratio=2*(logunres-logres)
cdf(title="LR Test for Children in School Dummies") chisqr $
  lratio 4
```
Example 14.2 Probit and Linear Probability Models

This is taken from Johnston and DiNardo (1997), pp. 415-425. It analyzes the probability of union membership using a sample of 1000 individuals with probit, logit and linear probability models. This is example UNION.PRG.

```
open data cps88.asc
data(format=prn,org=columns) 1 1000 age exp2 grade ind1 married $
   lnwage occ1 partt potexp union weight high

Fit a linear probability model. Do a histogram of the predicted “probabilities”

linreg union
   # potexp exp2 grade married high constant
prj fitlp
density(type=histogram) fitlp / gx fx
scatter(style=bargraph,$
   header="Histogram of Predicted Values from a LPM")
   # gx fx

Probit model. Show a scatter graph of the predicted probabilities of the probit vs the LPM. Include the 45 degree line on the graph.

ddv(dist=probit) union
   # potexp exp2 grade married high constant
prj(dist=probit,cdf=fitprb)
scatter(style=dots,lines=||0.0,1.0||,$
   header="Probit vs Linear Probability Model","$
   hlabel="Predicted Probability(Probit)",$
   vlabel="Predicted Probability(LP)")
   # fitprb fitlp

Evaluate the predicted effect of switching industries on individuals currently in low union industries. This is done by evaluating the predicted probabilities with the value of “high” turned on for all the individuals, then knocking out of the sample those who were already in a high union industry. The predicted probabilities are obtained by “dotting” the coefficients with the required set of variables, then evaluating the normal CDF (%CDF) at those values.

set z = %dot(%beta,||potexp,exp2,grade,married,1,1||)
set highprb = %if(high,%na,%cdf(z))
scatter(style=dots,vmin=0.0,lines=||0.0,1.0||,header=$
   "Effect of Affiliation on Workers in Low-Union Industries")
   # fitprb highprb
```
14.3 Censored and Truncated Samples

Background

Consider the basic regression model:

\[ y_i = X_i \beta + u_i \]

Suppose you wish to analyze this model using a cross-sectional sample. If the sample is chosen randomly from all available individuals, you can just use standard regression techniques. Similarly, there are no special problems if you select your sample based upon values of the exogenous variables.

In this section, we examine situations in which the sample is limited in some manner based upon the values of the dependent variable \( y \). The two simplest types of samples with limited dependent variables are truncated and censored samples.

In a truncated sample, an observation is left out of the observable sample if the value of \( y \) does not meet some criterion. For example, suppose you want to use payroll data to study the number of hours worked. You will have a truncated sample because your study will exclude people who work zero hours and are thus not on a payroll.

A censored sample (“tobit” model) has some observations for which we do not observe a true value of the dependent variable. For instance, in a study of unemployment duration, we will not see the true duration for an individual still unemployed at the end of the survey period.

More generally, if the value of a variable which is determined simultaneously with the dependent variable influences whether an observation is in the sample, then the sample suffers from selectivity bias.

We can see the statistical problem produced by such samples in the following example, where the sample is truncated at zero:

\[ y_i = X_i \beta + u_i \; ; \; \text{observe } i \text{ only if } y_i > 0 \]

If \( X_i \beta \) is small (less than 0), observation \( i \) can only be in the sample if \( u \) is large enough that \( y \) is positive. If you estimate by least squares, \( u \) and \( X_i \beta \) will be negatively correlated and \( \hat{\beta} \) will be biased towards zero. The mere fact that an observation is in the sample gives us at least some information about its residual.

Estimation

For the simplest forms of these, you can use the instruction \textbf{LDV} (limited dependent variables). Some other models can be done using \textbf{MAXIMIZE}, while others have a likelihood too complex for attack with maximum likelihood but can be estimated consistently by two-step methods.
Maximum Likelihood

Start with the normal linear model:

\[ y_i = X_i \beta + u_i, \quad u_i \sim N(0, \sigma^2) \]  

i.i.d.

If \( y_i \) is truncated below at \( TR \) (that is, only observations for which \( y_i \geq TR \) are in the sample), the (log) likelihood element for entry \( i \) is

\[ K - \frac{1}{2} \log \sigma^2 - \frac{1}{2} \sigma^2 (y_i - X_i \beta)^2 - \log \Phi \left( \frac{X_i \beta - TR}{\sigma} \right) \]

It turns out that a slight change to the parameterization, due to Olsen (1978), makes this a better behaved function: instead of \( \{\beta, \sigma\} \), change to \( \{\gamma, h\} \equiv \{\beta/ \sigma, 1/ \sigma\} \). The log likelihood is then

\[ K + \log h - \frac{1}{2} (h y_i - X_i \gamma)^2 - \log \Phi (X_i \gamma - h TR) \]

This is what \texttt{LDV} does internally, so if you do a \texttt{TRACE} option, it will show coefficient estimates for that parameter set. The model is switched back to the standard parameterization (with recomputed covariance matrix) once estimation is finished.

If \( y \) is censored below at \( TR \) (we don’t observe a true value of \( y \) if \( y < TR \)), then the log likelihood elements are (again with Olsen’s parameterization)

\[ \begin{cases} K + \log h - \frac{1}{2} (h y_i - X_i \gamma)^2 & \text{for observations which are not censored} \\ \log \Phi (h TR - X_i \gamma) & \text{for those that are} \end{cases} \]

This is often called the Tobit model. To use this with \texttt{LDV}, the censored values of \( y \) should be equal to their limit.

Using LDV

\texttt{LDV} allows you to estimate a model with either censoring or truncation, and limited either above, below or on both ends. You select which form of limit the dependent variable has with the options \texttt{TRUNCATE} or \texttt{CENSOR}. The choices for each are \texttt{LOWER}, \texttt{UPPER} or \texttt{BOTH}. (The default is \texttt{NEITHER}).

To set the truncation limit, use the option \texttt{UPPER} for the upper limit (if it exists) or \texttt{LOWER} for the lower limit. The limit can be a single value for all entries, or can vary across individuals. It does, however, have to be computable prior to estimation, that is, you can’t have the limit depending upon estimated parameters. For instance, the following estimates a model with censoring at both ends with a lower limit of 0 and upper of 4. (In the application from which this is taken (Greene (2003, p. 774)), the dependent variable is forcibly being censored at 4 because the values above 4 were categories, rather than observable values.)
set ytop = %min(y,4)
ldv(censor=both,lower=0.0,upper=4.0) ytop
# constant z2 z3 z5 z7 z8

and, from the same application, this estimates the model truncated at zero, restricting it to the sample with non-zero values of the series AFFAIR.

ldv(truncate=lower,lower=0.0,smpl=affair) y
# constant z2 z3 z5 z7 z8

If some entries are censored, while others aren’t, you can show this by providing a series for UPPER or LOWER with entries in that set to %NA (missing value) for those that aren’t limited. Suppose, for instance, that you have a series LIMIT in the data set which is the upper limit for an entry if it’s non-zero, and shows no limit if it’s zero. You could estimate this with a sequence like:

set upper = %if(limit==0,%na,limit)
ldv(censor=upper,upper=upper) ...

Because the log likelihood is well-behaved (when reparameterized as shown), and the second derivatives are rather simple to compute, LDV always estimates by Newton-Raphson (Section 7.2). You can use the standard set of options for computing robust covariance matrices, but keep in mind that the estimates themselves are unlikely to be consistent if the assumption of normality is wrong.

Like DDV, you can compute the “generalized residuals” with LDV by using the GRESIDS option. Note, however, that these are computed using the reparameterized model. The only difference between the generalized residuals based upon the two parameterizations is in a scale factor, which is usually of no consequence.

LDV also can do interval estimation, where the “dependent variable” really consists of a pair of values which bracket an otherwise unobservable amount. Use the option INTERVAL, and provide the bracketing values with the UPPER and LOWER options. Use an %NA for either one to show that it is unlimited; for instance, if an entry in the UPPER series is %NA, it means that the value was unbounded above. Note that you still need a dependent variable series, though its only real purpose is to show which entries should be used in estimation: any entry with a missing value in the dependent variable is skipped. The following, for instance, is from Verbeek (2004, example 7.2.3). The data were generated by starting with an initial bid (BID1). If an individual indicated an unwillingness to pay that much, a lower number was offered which also could be accepted or rejected. If the initial bid was accepted, a higher value was tried. The series UPPER and LOWER are created to show the upper and lower bounds created by this sequence.

set upper = %if(nn,bidl,%if(ny,bid1,%if(yn,bidh,%na)))
set lower = %if(nn,%na,%if(ny,bidl,%if(yn,bid1,bidh)))
ldv(upper=upper,lower=lower,interval,gresid=gr) bid1
# constant
Chapter 14: Cross Section/Panel Data

Using MAXIMIZE

It's useful to see how to estimate a limited dependent variables model using MAXIMIZE, in case you get an application which doesn't fit the form of LDV. As was the case with the logit and probit models, this is simplified by creating a FRML which computes $X\beta$.

To estimate a regression truncated below, do something like the following (where the series LOWER is set equal to the truncation values):

```
nonlin(parmset=sparms) sigmasq
linreg y
# constant x1 x2
frml(lastreg,vector=b,parmset=bparms) zfrml
compute sigmasq=%seesq
frml truncate = (z=zfrml(t)) ,$
   %logdensity(sigmasq,y-z)-%logcdf(sigmasq,z-lower)
maximize(method=bfgs,parmset=bparms+sparms) truncate
```

This estimates the model using the standard parameterization, starting from the least squares estimates.

The FRML for the log likelihood function for censored observations requires use of %IF to select the appropriate branch.

```
frml tobit = (z=zfrml(t)) , %if(y==lower,$
   %logcdf(sigmasq,z-lower),$
   %logdensity(sigmasq,y-z))
```

For example, this would estimate the example from Greene using MAXIMIZE censoring below (only) at 0:

```
linreg y
# constant z2 z3 z5 z7 z8
nonlin(parmset=sparms) sigmasq
frml(lastreg,vector=b,parmset=bparms) zfrml
compute sigmasq=%seesq
frml tobit = (z=zfrml(t)) , %if(y==0,$
   %logcdf(sigmasq,-z)),$
   %logdensity(sigmasq,y-z))
maximize(method=bfgs,parmset=bparms+sparms) tobit
```
Sample Selectivity and Heckit Estimators

The model with censoring below at 0 is often called the *Tobit* model. In the original context, the dependent variable was expenditure on a car. If the individual didn’t buy a car, this showed zero. One potential problem with the tobit model is that it combines two decisions (buy a car or not, and, if so, spend how much) into a single regression equation. An alternative way to model this is to have a separate model (a probit, presumably) for the first decision, and given a “yes” answer on the first decision, a regression to explain the amount spent.

The first step probit is straightforward. The second step, however, is likely subject to selectivity bias. If the underlying model for the probit is to choose to purchase if and only if

\[ X_i \gamma + v_i > 0 \]

and the model for the expenditure is

\[ y_i = X_i \beta + u_i \]

(the use of the same explanatory variables is just for convenience), then, if there is a correlation between \( u \) and \( v \), the estimates of \( \beta \) in (8) will be biased. Heckman’s idea (1976) is to compute the bias in \( u \):

\[ E(u_i \mid X_i, X_i \gamma + v_i > 0) \]

and adjust the second regression to take that into account. If \( u \) and \( v \) are assumed to be joint normal and i.i.d. across individuals, then \( u \) can be written:

\[ u_i = \lambda v_i + \xi_i \]

where \( v_i \) and \( \xi_i \) are independent normals. So

\[ E(u_i \mid X_i, X_i \gamma + v_i > 0) = E(\lambda v_i + \xi_i \mid X_i, X_i \gamma + v_i > 0) = E(\lambda v_i \mid X_i, X_i \gamma + v_i > 0) \]

With \( v \) as a standard normal (the usual assumption for a probit), then it can be shown that

\[ E(v \mid v > -z) = \frac{\phi(z)}{\Phi(z)} \]

where \( \phi \) is the density of the standard normal and \( \Phi \) is its cumulative density. \( \phi/\Phi \) is the reciprocal of Mills’ ratio, which you can obtain using the MILLS option of PRJ after estimating the first step probit model. Since \( \lambda \) is unknown, adding the inverse Mills’ ratio to the regressor set and allowing its coefficient to be estimated will give (under suitable conditions) a consistent estimator of \( \beta \). This estimator is sometimes known as Tobit II or Heckit.
The options on \texttt{PRJ} which are useful for doing these types of two-step estimators are \texttt{MILLS} (inverse Mills' ratio) and \texttt{DMILLS} (derivative of the inverse Mills' ratio). The above was all based upon a first stage probit, which conveniently is based upon a standard normal with bottom truncation at zero. The \texttt{MILLS} option, however, can do a broader range of calculations. In particular, it can handle a non-unit variance, and truncation either at the top or bottom. In general, the expected value for a truncated $N(0,\sigma^2)$ is given by

\begin{equation}
E(v|v > -z) = \frac{\sigma \phi(z/\sigma)}{\Phi(z/\sigma)}
\end{equation}

When we're looking to compute the expected value of a residual from the projection in a first stage estimator whose distribution is truncated at the value $T_i$, the normalized $z_i$ takes the following values:

\begin{equation}
z_i = \begin{cases} 
\text{Bottom truncation} & \frac{(X_i\hat{\gamma} - T_i)}{\sigma} \\
\text{Top truncation} & \frac{(T_i - X_i\hat{\gamma})}{\sigma} 
\end{cases}
\end{equation}

The \texttt{MILLS} option computes the value of $\phi/\Phi$ for these values. \texttt{PRJ} has three options for describing the truncation and normalization procedure.

\texttt{scale} = \text{the value of $\sigma$ [1]}
\texttt{upper} = \text{SERIES or FRML of upper truncation points [unused]}
\texttt{lower} = \text{SERIES or FRML of lower truncation points [series of zeros]}

The truncation points can differ among observations. For instance, cutoffs may depend upon some demographic characteristics. However, truncation must either be top truncation for all observations or bottom truncation for all. Note that, by default, the calculation is precisely the one needed for the Tobit II estimator with its first stage probit.

\textit{As first step in “heckit” procedure, estimate a probit for the “INLF” variable, using the explanatory variables from the hours equation}

\begin{verbatim}
  ddv(dist=probit) inlf
  # constant nwifeinc educ exper expersq age kidslt6 kidsge6

  Compute the inverse Mills' ratios from this

  prj(dist=probit,mills=lambda2)

  Run OLS again, including LAMBDA2 as an explanatory variable

  linreg(smpl=inlf) lwage
  # educ exper expersq constant lambda2
\end{verbatim}
Example 14.3 Tobit Models

This is an example from Verbeek (2004, example 7.4.3). It estimates the shares of alcohol ($SHARE1$) and tobacco ($SHARE2$) using both standard Tobit (censored at zero) and a two-step estimator with a separate probit estimators. It’s example `TOBIT.PRG`.

```
open data tobacco.asc
data(format=free,org=columns) 1 2724 bluecol whitecol flanders $
    walloon nkids nkids2 nadults lnx share2 $
    share1 nadlnx agelnx age d1 d2 w1 w2 lnx2 age2

Tobit I models

ldv(lower=0.0,censor=lower) share1
# constant age nadults nkids nkids2 lnx agelnx nadlnx
ldv(lower=0.0,censor=lower) share2
# constant age nadults nkids nkids2 lnx agelnx nadlnx

OLS regressions on positive observations only

linreg(smpl=share1>0) share1
# constant age nadults nkids nkids2 lnx agelnx nadlnx
linreg(smpl=share2>0) share2
# constant age nadults nkids nkids2 lnx agelnx nadlnx

Tobit II models, which require first step probits for non-zero consumption. While not strictly necessary, we remap the share values into 0–1 dummies. (DDV would work fine with just the zero-non-zero coding).

set choice1 = share1>0
set choice2 = share2>0

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 *

ddv(noprint) choice2
# constant age nadults nkids nkids2 lnx $
    agelnx nadlnx bluecol whitecol
prj(mills=lambda)
linreg(smpl=share2>0,title="Tobit II") share2
# constant age nadults nkids nkids2 lnx agelnx nadlnx lambda
```
Chapter 14: Cross Section/Panel Data

14.4 Hazard Models

Background

Hazard models are a generalization of simple models of “time to failure.” They are used extensively in biometrics to analyze survival times. In econometrics, they are used in modelling transitions from one state to another. Examples are unemployed to employed, working to retired, unmarried to married. A good reference on the subject is Lancaster (1990).

The basic assumption is that the probability of leaving the initial state to the final state over a very short period of time can be modeled as

\[
P(\text{exiting in } [t, t + dt] \text{ given no exit before } t) = \theta(t, X_i(t)) dt
\]

where \( X_i(t) \) are the characteristics of individual \( i \) at time \( t \). In the simplest case, \( \theta \) is a constant, and the exit times follow an exponential distribution with mean \( 1/\theta \). Obviously, the more interesting cases depend upon the individual or length of time. \( \theta \) is known as the hazard function, perhaps not the best choice of terms for economics examples, but in biometrics these models are most often used to analyze deaths and other less than pleasant occurrences.

Estimation

In full generality, it is almost impossible to estimate the parameters governing the hazard function. If the hazard function varies over time, either because it directly depends upon time, or (even more difficult) the characteristics of individuals change over time, the above would require integrating \( \theta \) over \( t \) to get the probability of exiting at \( T \) or earlier.

A common simplification is to assume that the dependence upon \( t \) comes only through a function which is independent of the individual. This is called a proportional hazard model. For estimation purposes, it is usually assumed that \( \theta \) is

\[
\theta_t(t) \exp(X_i \beta)
\]

If the baseline hazard function \( \theta_0(t) \) is simple enough, it’s possible to estimate the parameters directly using MAXIMIZE. The density function for the exit time is

\[
\theta_0(t) \exp(X_i \beta) \exp\left(-\exp(X_i \beta) \int_0^t \theta_0(s) ds\right)
\]

If, for instance, \( \theta_0 \) is a constant, the FRML to compute the log of this would be

\[
\text{frml logl = z=zfrml,$$
\log(\text{theta})+z\exp(z)\ast\text{theta}\ast\text{exittime}
\]
In many cases, however, it is easier to explain how exit times might vary across individuals than to explain the actual lengths of time to exit. Without a precise form for the baseline hazard, full maximum likelihood can’t be used. However, there is an alternative method of attack called partial likelihood. The idea here is that, if we look at the time an individual exits, we have a valid model of the relative probabilities of exit for all individuals who have still not exited. For each possible setting of \( \beta \), we can form, for each individual, the probability that she is the one who exits at her exit time rather than someone else. It turns out this has all the desirable properties of a likelihood.

The one tricky part about the partial likelihood is that the probability calculation for an individual requires a sum across a subsample, a subsample which changes with the individual. In the example, we handle this by defining for each individual a “Risk Set” vector which has 1’s only for those data points where the exit time is at least as large as it is for the individual in question. We also need to include an initialization FRML on the MAXIMIZE which computes the hazard functions for all the data points. Throughout this, NOBS is the number of observations. It’s a good idea to use a variable for this and not code the number in directly. This is from Greene (2003, example 22.10). The first does maximum likelihood assuming a two-parameter Weibull distribution for \( \theta(\tau) \):

```plaintext
nonlin b0 b1 p
frml zfrml = b0+b1*prod
frml weibduration = (z=zfrml),(w=-exp(z)*dur**p),$
  z+w+log(p)+(p-1)*log(dur)
compute p=1.0
maximize(method=bfgs) weibduration
```

*RISKSET* is a vector of vectors. Each individual has a vector and each vector has a slot for each individual. For individual \( i \), slot \( j \) will be 1 if \( j \)’s exit time is greater than or equal to \( i \)’s.

```plaintext
compute nobs=62
dec vect[vect] riskset(nobs)
dec vect hazards(nobs)
do i=1,nobs
  dim riskset(i)(nobs)
  wise riskset(i)(j)=dur(j)>dur(i)
end do i
```

The formula INIT gets computed at the start of each function evaluation to recalculate the relative hazard rates. These are put into the vector HAZARDS. The probability that an individual is the one to exit at her exit time is the ratio of her relative hazard to the sum of those hazards across the risk set for her exit time.

```plaintext
nonlin b
frml hazard = exp(b*prod)
frml init = %do(i,1,nobs,hazards(i)=hazard(i)),0.0
frml logl = log(hazards(t))-log(%dot(riskset(t),hazards))
compute b=0.0
maximize(trace,method=bfgs,start=init) logl
```
Chapter 14: Cross Section/Panel Data

14.5 Panel and Related Data Sets

Overview
The following pages provide the basic information you need to use panel data sets in RATS. Topics covered include data set organization, reading data into RATS, and statistical techniques.

Organization of Panel Data Sets
Panel data refers to data sets which include data for several individuals (or firms, countries, etc.) over some number of time periods. The data set itself will usually have one of the following structures:

A. Each variable has a single long block of values which contains data for all individuals. This is typical for survey data. The data can be grouped by individual or by time. For example, the data in the column on the left are grouped by individual (all time periods for the first individual, followed by all time periods for the second individual, etc.), while the column on the right is grouped by time (time period 1 for all individuals, time period 2 for all individuals, etc.):

<table>
<thead>
<tr>
<th>Grouped by Individual</th>
<th>Grouped by Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual 1, 1997:1</td>
<td>Individual 1, 1997:1</td>
</tr>
<tr>
<td>Individual 1, 1998:1</td>
<td>Individual 2, 1997:1</td>
</tr>
<tr>
<td>Individual 1, 1999:1</td>
<td>Individual 1, 1998:1</td>
</tr>
<tr>
<td>Individual 2, 1997:1</td>
<td>Individual 2, 1998:1</td>
</tr>
<tr>
<td>Individual 2, 1998:1</td>
<td>Individual 1, 1999:1</td>
</tr>
<tr>
<td>Individual 2, 1999:1</td>
<td>Individual 2, 1999:1</td>
</tr>
</tbody>
</table>

We are considering here only situations where there are a fixed number of time periods per individual.

B. Each individual has a separate time series for each variable. This is most common when data are assembled from different sources.

C. Each variable has a single long block of values, with a separate series identifying which individual is represented by an observation.

RATS and Panel Data
Of these three, the special panel data features of RATS apply only to type “A” data grouped by individual. RATS stores a panel data series as an \( N \times T \) stream of data, where \( N \) is the number of individuals in the series, and \( T \) is the number of time periods. The first \( T \) entries of the series represent all the time periods for the first individual, the next \( T \) entries represent all the entries for individual two, and so on. Panel data series always have the same number of entries per individual. Missing values are used to pad ranges if the data aren’t aligned naturally.

If your data are organized differently, you will need to convert it to this form in order to use the data in RATS as panel series. The instruction **PFORM** can be used for most such conversions.
Chapter 14: Cross Section/Panel Data

Note that you only need to set up your data set as a panel if you are planning to use a statistical technique (such as a fixed effects estimator) which actually utilizes the panel structure.

**CALENDAR and ALLOCATE with Panel Data**

To work with panel data sets, you need to use the appropriate `CALENDAR` and `ALLOCATE` instructions. The syntax of `CALENDAR` for panel data is

```
calendar (panelobs=no. of periods, other options) parameter (optional)
```

The `PANELOBS` option gives the number of time periods per individual. The rest of the `CALENDAR` instruction is the same as for time series data: it describes the time series structure of the data within each cross-section unit. However, you don’t **need** to set this other information to use the panel data features of RATS.

The syntax for an `ALLOCATE` for panel data is

```
allocate individuals//lastperiod
```

where `individuals` is the total number of individuals in the panel set. Technically, `lastperiod` is the number of time periods for the last individual, though it invariably is just the “number of periods” from `CALENDAR`.

**Examples**

```
calendar (panelobs=20)
allocate 50//20
```

This is an undated panel data set, with 50 individuals, and 20 time periods (observations) per individual.

```
calendar (panelobs=48,q) 1988:1
allocate 20//1999:12
```

This is a panel data set, with 20 individuals and 48 time periods per individual. Each cross-sectional unit has quarterly data, starting with January, 1988.

**Referencing Series Entries**

With panel series, you reference time period `n` of individual `m` as `m//n`. For example:

```
compute starting2 = income(2//1)
```

sets `STARTING2` to entry 1 for individual 2 of `INCOME`, and

```
declare vector firstper(20)
ewise firstper(i)=panelseries(i//1960:1)
```

creates `FIRSTPER` as a `VECTOR` of the 1960:1 entries for the first 20 individuals. Note that `m` and `n` can be any integer value or an integer-valued expression. As shown above, `n` can also be a date if you specified a date scheme on your `CALENDAR`. 
Chapter 14: Cross Section/Panel Data

Reading Panel Data From a File

You bring panel data into RATS using the same procedure as time series data.

1. Use \texttt{CALENDAR(PANEL=\ldots)} and \texttt{ALLOCATE} instructions.
2. Open a data file with \texttt{OPEN DATA}.
3. Read in the data with a \texttt{DATA} instruction.

RATS format is the only one which really “understands” panel data. With your data in RATS format, you can select a reduced set of entries from within each cross-section. For instance, if you have data from 1955, but only want to work with the data from 1970, you can set a \texttt{CALENDAR} appropriately:

\begin{verbatim}
calendar(panel=84,q) 1970:1
allocate 13//1990:4
\end{verbatim}

You can also “pad” out the cross-sectional units. If you have data only for 8 observations apiece, but use \texttt{PANEL=10} on \texttt{CALENDAR, DATA} will insert missing values for entries 9 and 10 within each unit.

However, if you are reading the data from any other format, the organization of the data on the file must match the \texttt{CALENDAR} \textit{exactly}. Each individual must have exactly the number of periods specified by the \texttt{PANELOBS} option. See “Forming a Panel Data Set” on the page 546 for instructions on getting your data into this format.

Special Functions

RATS has two functions which can be very useful when working with panel data.

\begin{itemize}
\item \texttt{%PERIOD(t)} returns the time period corresponding to entry \texttt{t}. This can be very useful for setting time period dummies (see below).
\item \texttt{%INDIV(t)} returns the number of the individuals corresponding to entry \texttt{t}.
\end{itemize}

For example, given a panel set with 2 individuals, and five periods per individual, entry 6 is the first period for the second individual, so \texttt{%PERIOD(6)} is 1 and \texttt{%INDIV(6)} is 2.

\begin{verbatim}
set ltrend = %period(t)
\end{verbatim}

creates \texttt{LTREND} as a trend which repeats within each cross-sectional unit. If you are using a “dated” \texttt{CALENDAR}, you can use the date functions (\texttt{%YEAR}, \texttt{%MONTH}, \texttt{%DAY}, \texttt{%WEEKDAY}) to determine the year, month, day, or day of the week of a given entry.

The function \texttt{%SERIESENTRY(series,entry)} also has value in certain circumstances. In \texttt{SET} and \texttt{FRML}, a reference to an entry in a panel data series is treated as missing if it’s in the zone for a different individual than the one for the current entry \texttt{T}. This is usually what you want, since it prevents lags of a series from crossing the boundary into another individual’s data range. This becomes a problem, however, when you \textit{want} the transformation to cross boundaries. Perhaps you’re doing boot-
strapping, and the population from which you’re drawing is the full set of observations. Or perhaps you’re trying to extract data from across individuals into a new series. In either case, instead of making the standard series reference, use the \%SERIESENTRY function, with the series and the entry being referenced as separate parameters. For instance, the following does a random draw from all 200 observations of the series RESIDS:

```
boot select 1/1 10/20
set udraw = %seriesentry(resids,select(t))
```

Creating Dummies

If you need separate dummies for each individual, a quick way to generate them is as follows:

```
dec vector[series] dummies(number of individuals)
do i=1,number of individuals
 set dummies(i) = %indiv(t)==i
end do i
```

This creates the VECTOR of SERIES called DUMMIES as your dummy variables. You can use DUMMIES by itself in a regressor list to get the whole set, or refer to individual series as DUMMIES(1), DUMMIES(2), etc.

If you need a set of dummies multiplied by a regressor, just do the same thing, but multiply by the regressor in the SET instruction:

```
set dummies(i) = (%indiv(t)==i)*xreg
```

Handling Lags

If you run a regression involving lags or leads, any observation which requires a lagged or led value not available within a cross-section is dropped from the regression. Similarly, if you use a lag or lead in a SET instruction, you will get a value of NA if the lag or lead goes outside the cross-section. For instance, in

```
set dx = x-x{1}
```

DX will have a missing value in the first entry in each cross-sectional unit.

If you need a lag series to have a value of zero when it goes out of the individual’s range, you can do that with something like this:

```
dec vect[series] flags(n-2)
do i=n-1,2,-1
 set flags(i-1) = %if(\%period(t)<=i,0.0,lcrmrte\{i\})
end do i
```

This generates a set of lag series for lags 2 to n–1, each of which zeros out when the time period is less than or equal to the lag being created.
Forming a Panel Data Set: The Instruction PFORM

If your data are not already in the correct form for a RATS panel data set, you may be able to use the instruction PFORM to rearrange it. Our recommendation is that you run a program to transform the data and write it to a RATS format file. Use the RATS format file when you are actually analyzing the data.

PFORM can take several different forms depending upon the current organization of your dataset. Note that you use PFORM to create one series at a time. If you have a nicely blocked dataset, but the grouping is by time, not individual, use

\[
\text{pform} (\text{transpose}, \text{blocks}=\text{number of time periods per individual}) \ \text{newseries} \\
\text{# oldseries}
\]

If you need to concatenate separate series to create a panel set, use

\[
\text{pform} \ \text{newseries} \\
\text{# list of individual series}
\]

and if you have a dataset with a tag series for the individuals, use

\[
\text{pform}(\text{indiv}=\text{series of individual tags}) \ \text{newseries} \\
\text{# oldseries}
\]

Panel Data Transformations: the Instruction PANEL

In addition to individual and time dummies, it is often necessary to transform data by subtracting off individual or time period means. This is done with the instruction PANEL. For the input series, PANEL creates a linear combination of the current entry (ENTRY weight), the mean of the series for an individual (INDIV weight), the mean across individuals for a given time period (TIME weight), the sums in each direction (ISUM and TSUM) and observation counts (ICOUNT and TCOUNT). For instance,

\[
\text{panel(entry}=1.0,\text{indiv}=-1.0) \ \text{series} / \ \text{dseries} \\
\text{panel(time}=1.0) \ \text{series} / \ \text{timemeans}
\]

The first of these creates DSERIES as the deviations from individual means of SERIES. The second creates TIMEMEANS as a series of means for the different time periods. This is copied across the individuals, so that, for instance, the first time period in each individual’s block will be equal to the mean of the first time period across all individuals.

PANEL also has an option to compute a separate sample variance for the data for each individual or for each time period. That series can be used as a SPREAD option in a LINREG. It should only be used if you have enough data per individual to give a reasonably sharp estimate of the variance.

\[
\text{linreg logc} \\
\text{# constant logpf 1f logq f2 f3 f4 f5 f6} \\
\text{panel( spreads=firmvar) %resids} \\
\text{linreg(spread=firmvar) logc} \\
\text{# constant logpf 1f logq f2 f3 f4 f5 f6}
\]
Selecting Subsamples

If you want to limit an estimation to a consecutive range of entries, you can simply specify the endpoints using the `start` and `end` parameters as you would with non-panel data. For example, to limit a regression to a range of observations for individual two, you would do something like this:

```
linreg y 2//1995:1 2//2000:12
# constant x1 x2
```

using panel-format date references.

However, this approach will *not* work for selecting a subset of time periods from *each* individual. For example, if you wanted to include observations from all individuals, you would *not* want to do:

```
linreg y 1995:1 2000:12
# constant x1 x2
```

RATS would just interpret these dates as referring to individual one, and would exclude all remaining individuals from the estimation.

Instead, you need to create a dummy variable with non-zero values in the entries you want to include, and then use this series with the `SMPL` option. The `%PERIOD` and `%INDIV` functions, and the various logical operators, are very handy for this. For example, to include data from 1995:1 through 2000:12 for all individuals:

```
set panelsmpl = %period(t)>=1995:1.and.%period(t)<=2000:12
linreg(smpl=panelsmpl) y
# constant x1 x2
```
14.6 Statistical Methods for Panel Data

Analysis of Variance: the Instruction PSTATS

Does a single statistical model seem to unite the individuals (or time periods) in the data set, or not? The instruction PSTATS can be used to help answer that question. It performs an analysis of variance test for common means, across individuals, across time, or both. This can be applied to data series themselves, but is more typically applied to the residuals from a simple regression across the entire data set.

To do this, use PSTATS with the option TEST. The EFFECTS option allows you to choose a test for INDIVIDUAL effects, TIME effects, or BOTH. The following output, for instance, gives a marginal significance level on the test for individual effects of .3597. We would conclude that there really isn’t compelling evidence that the individual means differ across individuals.

```
Analysis of Variance for Series RESIDS

Source       Sum of Squares  Degrees   Mean Square   F-Statistic  Signif Level
INDIV        1.5912296947623      39  .0408007614042        1.080 0.35971320
ERROR        6.0418415088589     160  .0377615094304
TOTAL        7.6330712036212     199

PSTATS will also test a series for equality of variances. To do this, use the SPREAD option with the appropriate EFFECTS option. (You can’t choose BOTH for this).
```

Linear Regressions

When you have a linear regression model, there are a number of techniques which can exploit the structure of a panel data set. See Hsiao (1986) for details. Because a panel data set has two or more observations per individual in the sample, it offers possibilities to reduce errors which would occur if only a simple cross section were available. However, it is necessary to make some assumptions which tie the model together across individuals. Among the possibilities:

1. All coefficients are assumed to be constant across individuals (and time).
2. Some coefficients are assumed to be constant, while others differ.
3. The coefficients are assumed to differ, but are “similar,” so that large differences are implausible.

In addition, there can be assumptions about a link among the error terms:

1. The error terms can be assumed to be independent across individuals (or across time), but correlated within an individual’s record.
2. The error terms can be assumed to be correlated at a given point in time across individuals, but independent across time.

Even if an assumption is plausible for your data set, you need the proper amount of data to employ the technique. For instance, if you have only a few observations per individual, it will be difficult, if not impossible, to allow the coefficients to vary across
individuals, since you don’t have enough data for any individual to tack down the coefficient estimates. Similarly, if you have many individuals, you may not be able to estimate freely a covariance matrix of error terms across individuals.

The most commonly employed techniques are the Fixed and Random effects estimators. Fixed effects allows intercepts to vary while keeping other coefficients fixed. Random effects is a related technique which works through assumptions on the error term. The next section is devoted to them. The Random Coefficients model (Swamy’s method) is shown in Example 14.5. We’ll describe the others more briefly here.

Regressions with Dummy Variables

If you want to allow coefficients other than the intercepts to vary (freely) across individuals, you can create individual dummies times regressors for each varying regressor. (If everything is fixed except the intercepts, you can use a fixed effects estimator.) If you have a very large number of individuals, this can become impractical, as the regression can be too big to be handled.

Part of Example 14.4 shows how to do a regression with dummy variables.

Heterogeneous Regressions

The instruction `SWEEP` with the option `GROUP=%INDIV(t)` can handle a variety of complicated operations based upon linear regressions in which the coefficients are allowed to vary (freely) across individuals. Among these are the calculations of means of the coefficient vectors (mean group estimators) and the extraction of residuals from “nuisance” variables.

```
sweep(group=%indiv(t),var=hetero)
# invest
# constant firmvalue cstock
```

does a “mean group” regression of `INVEST` on `CONSTANT`, `FIRMVALUE` and `CSTOCK`, producing in `%BETA` the average of the coefficients across individuals and in `%XX` the covariance matrix of the estimates allowing the variances to differ among individuals.

```
sweep(group=%indiv(t),series=tvar)
# dc lpc{1} lndi dp
# dy ddp constant
```

regresses each of the series `DC`, `LPC{1}`, `LNDI` and `DP` on `DY`, `DDP` and `CONSTANT`, with a separate regression for each individual, producing the `VECT[SERIES] TVAR` with the residuals from those regressions.
Chapter 14: Cross Section/Panel Data

Autoregressive Errors: Using AR1

There are some specialized techniques which combine autoregressive errors with some other panel data methods. These aren’t available in RATS. If you run an AR1 instruction on a panel data set, you have two ways of handling the estimation of ρ:

- You can assume that it is fixed across all cross-sections (default treatment).
- You can assume that it is different in each (use option DIFFERING).

We don’t recommend the use of DIFFERING unless you have many time periods per cross-section. The error introduced into the GLS procedure by using a large number of rather poorly estimated serial correlation coefficients can be very large.

If you use DIFFERING, AR1 does not iterate to convergence and leaves the ρ’s out of the covariance matrix of coefficients.

Robust Errors

With time series data, you can’t, in general, handle arbitrary and unknown patterns of serial correlation; the requirement for consistency is that the window width has to go to infinity slower than the number of data points, and to guarantee a positive definite covariance matrix, you need to use a lag window type (such as Newey-West) which further cuts the contribution of the longer lags. With panel data, however, you can rely on the “N” dimension to provide consistency. Assuming that “T” is relatively small, you can correct for arbitrary correlation patterns by using ROBUSTERRORS with the option LWINDOW=PANEL. This computes the following as the center term in the “sandwich” estimator, the positive definite:

\[(1) \sum_i \left( \sum X_{it} u_{it} \right) \left( \sum X_{it} u_{it} \right)\]

Seemingly Unrelated Regressions

Seemingly unrelated regression techniques (Section 5.11) are, in general, applied to panel data sets, though usually to “small N, large T” data sets. As implemented in RATS using the instruction SUR, these require a well-formed panel of type “B.” You can, however, estimate using a type “A” panel set, using the instruction PREGRESS (Panel Regress) with the option METHOD=SUR. This estimates the model

\[(2) y_{it} = X_{it} \beta + u_{it}\]

\[(3) E u_{it} u_{js} = \begin{cases} \sigma_{ij} & \text{if } t = s \\ 0 & \text{otherwise} \end{cases}\]

Note that the number of free parameters in the covariance matrix is \(N(N+1)/2\). If you have relatively short time series, you may quickly exhaust your ability to estimate an unconstrained covariance matrix if you include many individuals.
First Difference Regressions

If the model is (2) with the assumption that

\[ u_{it} = \varepsilon_i + \eta_{it} \]

one approach for dealing with the individual effects (\(\varepsilon\)) is to first difference the data within each individual, since

\[ u_{i,t} - u_{i,t-1} = \varepsilon_i + \eta_{i,t} - \varepsilon_i - \eta_{i,t-1} = \eta_{i,t} - \eta_{i,t-1} \]

While it’s possible to do this manually (with \texttt{SET} or \texttt{DIFFERENCE}), you can also run this using \texttt{PREGRESS} with the option \texttt{METHOD=FD}.

Instrumental Variables

Instrumental variables estimators are important in panel data work because, in many cases, correcting for individual effects will create correlation problems between the transformed errors and the transformed regressors. For instance, if the explanatory variables include lagged dependent variables, neither fixed effects nor first differencing will provide consistent estimates for “large \(N\), small \(T\)” (For first difference, the bias doesn’t go away even with large \(T\)).

To do instrumental variables with panel data, you use the same instructions as you would for time series data: \texttt{INSTRUMENTS} to set the instrument set and \texttt{LINREG, AR1} or another instruction with the \texttt{INST} option. The main difference isn’t with the instructions used, it’s the process required to create the instrument set: in panel data sets, the instruments are often only weakly correlated with the regressors, so often large sets of them are used. See, for instance, Example 14.6.
14.7 Fixed and Random Effects Estimators

Background

The basic regression model for a (balanced) panel data set is

\[ y_{it} = X_{it} \beta + u_{it}, \quad i = 1, \ldots, N; \quad t = 1, \ldots, T \]

These two types of estimators are designed to handle the systematic tendency of \( u_{it} \) to be higher for some individuals than for others (individual effects) and possibly higher for some time periods than for others (time effects).

The fixed effects estimator does this by (in effect) using a separate intercept for each individual or time period. Since \( N \) is usually large, we actually implement it either by subtracting out individual and/or time means using the instruction \texttt{PANEL} and then doing \texttt{LINREG}, or by using the \texttt{PREGRESS} instruction.

The random effects estimator is based upon the following decomposition of \( u_{it} \):

\[ u_{it} = \varepsilon_i + \lambda_t + \eta_{it} \]

where \( \varepsilon \) is the individual effect, \( \lambda \) the time effect, and \( \eta \) the purely random effect. \( \beta \) is estimated by GLS using the structure imposed upon \( u_{it} \) by this assumption.

A Comparison

There are advantages and disadvantages to each treatment of the individual effects. A fixed effects model cannot estimate a coefficient on any time-invariant regressor, such as sex, schooling, etc., since the individual intercepts are free to take any value. By contrast, the individual effect in a random effects model is part of the error term, so it must be uncorrelated with the regressors. This means, for example, that a systematic tendency to see higher values for those with higher levels of schooling will be reflected in the coefficient on schooling.

On the flip side, because the random effects model treats the individual effect as part of the error term, it suffers from the possibility of bias due to a correlation between it and regressors (such as between unobservable talent and observable schooling).

See the discussion of these points in Hausman and Taylor (1981).

Implementation

The instruction \texttt{PREGRESS} (Panel Regress) estimates both of these. This is similar in syntax to \texttt{LINREG}, but you choose \texttt{METHOD=FIXED} or \texttt{METHOD=RANDOM} and which type of effects to allow: \texttt{EFFECTS=INDIV}, \texttt{TIME} or \texttt{BOTH}. For random effects, \texttt{PREGRESS} first runs a fixed effects regression and estimates the variances of the components from the fixed effects residuals. You can, however, override this and provide your own variance estimates using the \texttt{VINDIV}, \texttt{VTIME} and \texttt{VRANDOM} options.
Example 14.4 Fixed and Random Effects Estimators

This estimates a model by fixed and random effects using \texttt{PREGRESS}. It also demonstrates how the equivalent regressions can be done using least squares with dummy variables and with panel data transformations. This is the example file \texttt{PANEL.PRG}.

\begin{verbatim}
cal(panelobs=20) 1935 all 10//1954:1 open data grunfeld.dat data(format=prn,org=cols)

Do fixed and random effects. The intercept is dropped from the fixed effects regression since it will be wiped out as a time-invariant variable

\begin{verbatim}
preg(method=fixed) invest # firmvalue cstock
preg(method=random) invest # constant firmvalue cstock
\end{verbatim}

Do a first difference regression. Again, the intercept is dropped.

\begin{verbatim}
preg(method=fd) invest # firmvalue cstock
\end{verbatim}

Do the SUR (cross sectional correlation) estimator

\begin{verbatim}
preg(method=sur) invest # constant firmvalue cstock
\end{verbatim}

Do fixed effects as least squares with dummy variables

\begin{verbatim}
dec vect[series] idummies(10) do i=1,10 set idummies(i) = %indiv(t)==i end do i linreg invest # firmvalue cstock idummies
\end{verbatim}

Do random effects by using panel data transformations. \texttt{%VRANDOM} and \texttt{%VINDIV} are the estimates of the component variances from the \texttt{PREGRESS} instructions. The coefficient estimates are identical to those from \texttt{PREGRESS}, but the standard errors are slightly different because the \texttt{LINREG} on the transformed variables estimates a new \texttt{SIGMA**2}, while \texttt{PREGRESS} uses the \texttt{%VRANDOM} value.

\begin{verbatim}
compute theta=1-sqrt(%vrandom/(%vrandom+20*%vindiv))
panel(entry=1.0,indiv=-theta) invest / ifix
panel(entry=1.0,indiv=-theta) firmvalue / ffix
panel(entry=1.0,indiv=-theta) cstock / cfix
set constfix = 1-theta
linreg ifix # constfix ffix cfix
\end{verbatim}
\end{verbatim}
Example 14.5 Random Coefficients Model

The basic idea is that the cross-section coefficient vectors are “drawn” from a distribution with a common mean. This is sometimes known as Swamy’s (1970) model. An almost identical analysis from a purely Bayesian standpoint is given in Leamer (1978), pp 272-5.

The assumptions underlying the model are

1. $\beta_i = \beta + u_i$, with
2. $\text{var}(u_i) = \Delta$

If $\Delta$ is small, the coefficients will be almost identical. If it were extremely large, the individual coefficient vectors would be, in effect, unconstrained. With a moderate value, the estimator suggests that they have similar, but not identical, values.

This requires three passes. The first collects information on the individual regressions. The second determines the precision of the overall estimate, and the final one takes a matrix weighted average of the individual estimates to get the grand estimate. The example file is \texttt{SWAMY.PRG}. This (and the related mean groups estimator) are also provided by the procedures \texttt{SWAMY.SRC} and \texttt{MEANGROUP.SRC}.

```
cal(panelobs=20) 1935
all 10//1954:1
open data grunfeld.dat
data(format=prn,org=cols)
compute nindiv=10

dec vect[symm] xxmats(nindiv)
dec vect[vect] betas(nindiv)
dec vect sigmas(nindiv)
dec symm delta(3,3)
dec symm xxgls(3,3)
dec vect betagls(3)
dec integer k

First pass, run OLS regressions over each individual. Save \%XX, \%BETA and \%SEESQ. Also accumulate the sum of the coefficients and their outer product.

compute delta=%const(0.0)
compute betagls=%const(0.0)
do i=1,nindiv
  linreg(noprint) invest i//1 i//20
  # constant firmvalue cstock
  compute xxmats(i)=%xx
  compute betas(i)=%beta
  compute sigmas(i)=%seesq
  ewise delta(j,k)=delta(j,k)+beta(j)*beta(k)
  compute betagls=betagls+beta
end do i
```
Estimate the covariance matrix of beta’s

\[
\text{ewise delta}(j,k) = \frac{1.0}{(\text{nindiv}-1)} \times (\text{delta}(j,k) - \frac{1.0}{\text{nindiv}} \times \text{betagls}(j) \times \text{betagls}(k))
\]

Second pass. Compute the GLS covariance matrix. While we’re at it, replace XXMATS with the precision.

\[
\text{compute xxgls} = \%\text{const}(0.0)
\]
\[
\text{do i=1,nindiv}
\]
\[
\text{compute xxmats}(i) = \text{inv}(\text{delta} + \text{sigmas}(i) \times \text{xxmats}(i))
\]
\[
\text{compute xxgls} = \text{xxgls} + \text{xxmats}(i)
\]
\[
\text{end do i}
\]
\[
\text{compute xxgls} = \text{inv}(\text{xxgls})
\]

Final pass. Compute the grand coefficient matrix

\[
\text{compute betagls} = \%\text{const}(0.0)
\]
\[
\text{do i=1,nindiv}
\]
\[
\text{compute betagls} = \text{betagls} + \text{xxgls} \times \text{xxmats}(i) \times \text{betas}(i)
\]
\[
\text{end do i}
\]
\[
\text{linreg(create,coeffs=betagls,covmat=xxgls,form=chisquared) invest}
\]
\[
# \text{ constant firmvalue cstock}
\]

Example 14.6 Arellano-Bond GMM Estimator

This is based upon example 11.3 from Wooldridge (2002). It estimates an autoregression on panel data (seven years of data for 90 counties) for the log of the crime rate. Because of the lagged dependent variable, a fixed effects estimator will have substantial bias with \( T \) being this small, as will the first difference estimator. The model is estimated using the GMM estimator of Arellano and Bond (1991). This is an IV estimator on the first differenced data using all available instruments for all potential lags.

This is example ARELLANO.PRG.

\[
\text{cal(panelobs=7) 1981}
\]
\[
\text{all 90//1987:1}
\]
\[
\text{open data cornwell.prn}
\]
\[
\text{data(org=columns,format=prn) / county year crmrte}
\]

Transform to log first differences

\[
\text{set lcrmrte} = \text{log(crmrte)}
\]
\[
\text{diff lcrmrte / lcrmrte}
\]

The regression equation is 1st difference of the log crime rate on its lag. First differencing eliminates the individual county effect, but almost certainly produces serially correlated residuals, and thus, OLS would give inconsistent estimates because of the lagged dependent variable. The first set of estimates does instrumental variables using the second and third lags of the log crime rate as instru-
ments. (The first lag still has a correlation with the residual). This is just run over one observation per individual (the final one) to avoid having to correct the covariance matrix for serial correlation.

instruments constant lcrmrte{2 3}
set smpl = %period(t)==1987:1
linreg(instruments,smpl=smpl) clcrmrte
# constant clcrmrte{1}

Check 1st stage regression

set cllag = clcrmrte{1}
linreg(smpl=smpl) cllag
# constant lcrmrte{2 3}

The lags tend to be rather weak instruments, and using a standard instrument setup will severely restrict the number of data points or instruments which could be used.

A different approach is to create a separate instrument for each lag and for each time period and use GMM to weight them. This is the Arellano-Bond estimator. The instruments need to be constructed. This shows how do to this, however, you could more simply use the procedure ABLags:

@ABLags lcrmrte abivs

compute m=7
dec vect[series] abivs((m-2)*(m-1)/2)
compute fill=1
do period=m,3,-1
   do lag=period-1,2,-1
      set abivs(fill) = %if(%period(t)==period,lcrmrte{lag},0.0)
      compute fill=fill+1
   end do lag
end do period

The overidentification test is included in the regression output.

instrument constant abivs
linreg(title="Arellano-Bond","$
    instruments,optimalweights,lwindow=panel) clcrmrte
# constant clcrmrte{1}
Chapter 15
Spectral Analysis

RATS is one of the few econometric software programs which includes spectral analysis. We consider it to be an important tool in time-series analysis.

With RATS, you do spectral analysis using a sequence of primitive instructions such as **FFT** for Fourier transform and **WINDOW** for spectral window smoothing. This permits a great deal of flexibility, and ensures that you know exactly what is happening. On the other hand, the proper use of the instructions demands a somewhat higher level of sophistication on your part than from some other programs.

However, we have put together procedures which handle many of the key applications. These operate much like standard RATS instructions. You can rely upon them if you are unsure of your skills.

While there are some uses for direct spectral and cross-spectral analysis, the most important use of spectral methods is as a computational device. Many techniques are actually applications of the frequency domain filtering (15.8).
Chapter 15: Spectral Analysis

15.1 Complex-Valued Expressions

Mixing Modes

Complex is a more general type than real or integer. If needed, RATS will automatically convert an integer or a real to complex (with zero imaginary part). The explicit type conversion functions are:

- \%real(z) returns the (real-valued) real part of z
- \%imag(z) returns the (real-valued) imaginary part of z
- \%cmplx(x1,x2) creates the complex x1 + ix2 from the real numbers/expressions x1 and x2.

Using Operators with Complex Values

The following operators can be used in complex-value calculations:

- +, - and / function as expected.
- * multiplies the first number by the conjugate of the second number, that is, \( A \times B = A\overline{B} \).
- ** functions as expected. The only operation not supported is raising a negative real number (with zero imaginary part) to a power.

Relational Operators

You can use the == and <> operators to test equality or inequality of two complex numbers. You cannot use any of the other relational operators ( >, <, >=, <=).

Predefined Variables

\%pi is the constant \( \pi \) (REAL variable)

Functions

The functions of particular interest are (x’s are real, z’s are complex, t’s are integer):

- \( \sin(x), \cos(x), \tan(x) \) Sine, cosine, tangent (real-valued)
- \%asin(x), \%acos(x), \%atan(x) Inverse sine, cosine, tangent
- \%cmplx(x1,x2) Complex number x1 + ix2
- \%real(z), \%imag(z) Real and imaginary parts (real-valued)
- \%conjg(z), \%csqrt(z) Complex conjugate and complex square root
- \%cabs(z), \%arg(z) Absolute value and argument. The polar decomposition is \%cabs(z)*exp(i\%arg(z))
- \%clog(z), \%cexp(z) Complex natural logarithm and antilogarithm
- \%unit(x), \%unit2(t1,t2) Unit circle values \( e^{ix} \) and \( e^{i2\pi(t_1-1)/2} \)
- \%zlag(t1,t2) Unit circle value \( e^{i2\pi t_2(t_1-1)/N} \)
15.2 Complex Series

Complex Series vs Real Series

Complex series and real series are separate types. You can’t substitute one where RATS expects the other. While there are many similarities in structure between the two types of series, there is little similarity in their use. This section describes the differences.

The data type name of a complex series is \texttt{SERIES[COMPLEX]}, or \texttt{CSERIES}, for short.

Creating Series

As with real series, you can create complex series by name when needed. However, it usually is more convenient to set up the full block of series needed (using \texttt{FREQUENCY}) and to refer to the series by numbers rather than by names. There are several reasons for this:

- Most frequency domain procedures are short and straightforward and are adapted easily for use with different data sets. Referring to series by number simplifies this process.
- A single series usually is subjected to several transformations in succession. In most instances, you can keep transforming the series onto itself. Giving a single name to the series would be misleading.

These points will become more obvious as you look at the examples.

CSET Used Less Often

You will note a similarity between the frequency domain instructions and the time domain instructions. However, a different set of instructions is important. While there is an instruction \texttt{CSET} (the complex analog of \texttt{SET}), specialized transformation instructions, such as \texttt{CMULTIPLY} and \texttt{CLN} (complex log) are used much more often.

FREQUENCY Instruction

The \texttt{FREQUENCY} instruction sets up a block of complex series of a specific length. Each use of \texttt{FREQUENCY} eliminates the previous block of complex series. This makes it possible (and desirable) to write self-contained procedures for your analysis, as each procedure can have its own \texttt{FREQUENCY} instruction to meet its needs.

No “CSMPL” Instruction

There is no “CSMPL” instruction corresponding to \texttt{SMPL}. Since most series are defined over the full \texttt{FREQUENCY} range and most transformations use all entries, the use of the defined range as the default is almost always adequate.
Chapter 15: Spectral Analysis

15.3 Complex-Valued Matrices and FRML’s

Data Types

The two main types of matrices of complex numbers are the VECTOR [COMPLEX] (or CVECTOR for short) for a one-dimensional array and RECT [COMPLEX] (CRECT) for a general two-dimensional array. There’s also a SYMMETRIC [COMPLEX], but symmetric arrays aren’t as useful as they are with real-valued arrays. (The true analogue of the symmetric real array is a conjugate symmetric or self-adjoint matrix). You declare and these with instructions like DECLARE, LOCAL, TYPE and OPTION just as you would real-valued arrays. And you can dimension them with DIMENSION, or on DECLARE or LOCAL.

You can define series of complex matrices: SERIES [CRECT], for instance, is a series of complex 2-dimensional arrays. And you can define FRML’s which evaluate to complex numbers or complex matrices with the types FRML [COMPLEX] (formula evaluating to complex numbers), FRML [CVECTOR] and FRML [CRECT].

Operators and Functions

When applied to complex matrices, the operators +, - and * have the expected meanings. Note that * is just a standard matrix multiply. You can use * to multiply complex matrices by real-valued matrices or real or complex scalars as well.

There’s a much more limited set of functions for complex matrices:

%cxinv(Z)  Matrix inverse
%cxadj(Z)  Adjoint (conjugate transpose)
%cxsvd(Z)  Singular value decomposition
%cxeigdecomp(Z)  Eigen decomposition

%CXSVD and %CXEIGDECOMP each create a VECT [CRECT] with the components of the decomposition.

Most calculations with complex-valued arrays are done with COMPUTE or with GSET. The latter is used when working with series of complex arrays, when you have one for each frequency.

Complex Eigenvalues/vectors of Real-valued Matrices

The instructions EIGEN (for eigen decomposition) and QZ (for generalized Schur decomposition) can both create complex vectors for their eigenvalues. For both instructions, you use the option CVVALUES = VECTOR [COMPLEX] for eigenvalues. With EIGEN, you can also get the complex matrix for the (column) eigenvectors with the CVECTORS = RECT [COMPLEX] of eigenvectors.
15.4 Fourier Transforms

Description

The Finite Fourier transform of the series \( \{X(t), t = 1, \ldots, T\} \) is

\[
\tilde{X}(2\pi j/T) = \sum_{t=1}^{T} X(t) \exp(-2\pi i (t - 1)/T)
\]

The frequencies run from 0 to \( 2\pi(T - 1)/T \) by increments of \( 2\pi/T \).

Note that this is sometimes defined using a \( 1/\sqrt{T} \) multiplier (and \( 1/\sqrt{T} \) also on the inverse transform). We have found that the formula above (with a \( 1/T \) multiplier on the inverse) is more convenient. The instruction \texttt{FFT} computes the Fourier transform and \texttt{IFT} computes its inverse.

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.

RATS also uses a general complex \texttt{FT}. You can apply it to any series, not just real-valued series.

Frequency and Entry Mappings

Different ways of mapping entries to frequencies and back are used in spectral analysis packages. We have chosen the above because our primary interest is in (one-sided) time series data. It means, however, that if you are interested in two-sided sequences (such as the covariogram, or a two-sided distributed lag), you have to do some rearranging.

The two-sided finite sequence

\[
\{x_{-N}, x_{-N+1}, \ldots, x_{-1}, x_0, x_1, \ldots, x_N\}
\]

is represented in this mapping as the one-sided series

\[
\{x_0, x_1, \ldots, x_N, x_{-N}, x_{-N+1}, \ldots, x_{-1}\}
\]

This is true whether you are inputting the sequence to \texttt{FFT} or getting output from \texttt{IFT}. For instance, if you compute a covariogram by inverse transforming the periodogram, lag 0 (the variance) will be in entry 1, lag 1 in entry 2, etc., and lag -1 (which is identical to lag 1) will be in the last entry, lag -2 in the second to last, etc.

The following, for instance, creates a symmetrized window with 320 ordinates, note that the value used for \( t \) on one end goes into entry \( 321-t \) on the other.

\[
cset 2 = \%if(t<=k, 1-(t-1)/k, \%if(t>320-k, 1-(321-t)/k, 0.0))
\]
15.5 Preparation and Padding

Preparation of Data

Most data used in spectral analysis are prepared in the time domain and copied to the frequency domain using the instruction RTOC (Real TO Complex). Frequency domain techniques are among the fussiest of all time series techniques concerning data preparation. Applying FFT and IFT in succession will reproduce any series, no matter how non-stationary, but almost any real work done in the frequency domain will require some calculations which are sensitive to improper differencing or detrending.

There are two technical reasons for this:

- Finite Fourier methods treat the data (in effect) as being periodic with period $T$, where $T$ is the number of data points. Thus, the point “before” entry 1 is entry $T$. Clearly, if a series has a trend, it can’t be reasonably represented in such a fashion.
- The statistical properties of spectral estimators break down when the series does not have a well-behaved spectral density.

Thus, whenever the method you are using requires you to take the spectrum of a series, you should make sure:

- the series is properly differenced (usually done with DIFFERENCE or SET) or detrended.
- the series has a mean of zero, or close to it.

Missing values in a data set are replaced by zeros when they are copied to the frequency domain. Thus, you don’t have to concern yourself about undefined elements at the ends of your series, since they will just become part of the padding.

RATS provides the instruction TAPER for applying a taper to a data set prior to Fourier transforming it. A taper scales the ends of the actual data (not the padded series) with a graduated series so that it merges smoothly with the zeros in the padding.
Padding

The length on the FREQUENCY instruction is usually chosen to be a convenient number of frequencies for Fourier analysis. The extra length beyond the actual number of data points is the padding. These extra entries are set to zero when you transfer data using RTOC. There are two considerations in choosing length:

- The Fourier transforms (instructions FFT, IFT and TRFUNC) are much faster for series lengths which have small prime factors (2, 3 and 5) than for lengths which are products of large primes. For instance, applying FFT to length $1024 = 2^{10}$ is roughly 40 times faster than applying it to $1023 = 3 \times 11 \times 31$. Given the speed of modern computers, this is only a minor consideration if you are just applying the Fourier transform a few times. However, if the Fourier transform is part of a function evaluation for non-linear estimation, a timing gap like that can be quite significant.

- The exact seasonal frequencies are included if the number of frequencies is a multiple of the number of periods per year.

You can use the function %FREQSIZE(n) to get a recommended size for n actual data points. It returns the smallest integer of the form $2^mS$ which is greater than or equal to n. (S is the number of periods per year).

For procedures that require filtering in the frequency domain (Section 15.7), the series must be padded, preferably to at least double length. To get a recommended length in that case, use $2 \times %FREQSIZE(n)$.

Example

```
linreg logx 1970:1 2007:7 resid
# constant trend
frequency 1 768  # Pad 449 monthly to 768=(12)(64)
rtoc
# resid
# 1
```
15.6 Getting Output

Tools

Use **CPRINT** to print complex series. You can also use **DISPLAY** or **WRITE** to print individual complex numbers or expressions. Other types of output (such as graphics) will require you to transfer data to real series (see below).

Reducing Ordinates

**CPRINT** can produce a vast amount of output when you have a long series or have padded it substantially. Since spectral densities and related functions are usually quite smooth, you lose little by printing just a subset of the frequencies. You can use the **INTERVAL** option on **CPRINT** to pick entries at a regular interval.

Also, because a spectral density is symmetric about 0 and a cross-spectral density is conjugate-symmetric, you really only need to look at frequencies 0 to $\pi$. In RATS, these are entries 1 to $(T/2)+1$. If you cut the frequencies down to half in this way, use the option **LENGTH=$T$** on **CPRINT** so the entries will be labeled properly.

This cuts 288 ordinates by printing just half (to 144) then picking every 4th one.

```
cprint(lc=freq,length=288,interval=4) 1 144 3
```

Using **GRAPH** and **SCATTER** for Complex Series

Neither **GRAPH** nor **SCATTER** is designed to handle complex series directly, but you can use them to graph a real series which has been sent from the frequency domain using **CTOR**.

You can use either instruction to graph a spectral function. Each has minor advantages and disadvantages—we will demonstrate both.

**GRAPH** is good for quick views. The only drawback for this is that **GRAPH** will want to label the horizontal axis as if you were graphing a time series and not a function of frequencies. The option **NOTICKS** is handy to suppress the misleading labeling.

If you graph a spectral density estimate, the values will usually have a huge range. The peak values can easily be many orders of magnitude larger than most. Remember that this is a decomposition of the **variance**, so size differences get squared. In most cases it is more useful to graph the log of the spectrum rather than the spectrum. You can do this either by taking the log first, or by using the **LOG** option on **GRAPH**, such as **LOG=10**. For **SCATTER**, the analogous option is **VLOG=10**.
SCATTER is better for “production” graphs. It can do a better job of handling the “frequency” axis, though that comes at the cost that it can’t be used without some additional instructions to create a series to represent the frequencies. This is an instruction of the form

\[
\text{set frequencies} = \frac{(t-1.0)}{N}
\]

where “N” depends upon how much of the frequency range you’re graphing. The scaling doesn’t affect the appearance of the graph itself, just the labeling on the x-axis. For instance, suppose you are graphing half the frequencies (thus 0 to \(\pi\)). If you use \((T-1.0)\) divided by the number of frequencies, the x-axis will run from 0 to 1.0 and is thus showing the fractions of \(\pi\). A slight change to

\[
\text{set frequencies} = 6.0 \times \frac{(t-1.0)}{N}
\]

would map the x-axis to 0.0 to 6.0. The integer x-values would be multiples of \(\pi/6\), which are the harmonics for monthly data. The ultimate in fancy labeling uses the XLABELS option. This allows you to specify full strings which are equally spaced across the x-axis. Using, for instance,

\[
\text{XLABELS}=|0, \pi/6, \pi/3, \pi/2, 2\pi/3, 5\pi/6, \pi|
\]

will, when combined with the instruction

\[
\text{grparm(font="symbol") axislabels 12}
\]

give you labels of 0, \(\pi/6\), \(\pi/3\), etc.

The following example is taken from the SPECTRUM.SRC procedure. \texttt{HALF} is half the number of frequencies. The spectral density is plotted on a log scale.

\begin{verbatim}
ctor 1 half
# 2
#  spect
set frequencies 1 half = (t-1.0)/half
scatter(style=lines,header=header, $
  hlabel="Fractions of Pi", vlog=10.0) 1
# frequencies spect 1 half
\end{verbatim}
15.7 Computing Spectra and Cross Spectra

Periodograms

The periodogram of a series is its squared Fourier transform. The cross-periodogram of two series is the product of one FT by the other’s conjugate. You compute both of these using FFT followed by CMULTIPLY.

\[
\text{fft 1} \\
\text{fft 2} \\
\text{cmultiply(scale=1./(2*%pi*points))) 1 1 / 3} \\
\text{cmultiply(scale=1./(2*%pi*points))) 1 2 / 4}
\]

The correct scale factor is \(1/(2\pi N)\) where \(N\) is the number of actual data points. We represent \(N\) above as \(\text{points}\). This might be a number or a variable previously computed.

The raw periodogram is used directly in the cumulated periodogram test (see CACCMULATE), and can be inverse transformed to get the covariogram (see description of lag windows on the next page). However, usually it is smoothed to get a consistent estimate of the spectral density or cross-spectral density.

Smoothing

While the periodogram gives an unbiased estimate of the spectrum, it is inconsistent, as the variance of the estimators does not go to zero as the number of data points grows.

RATS provides the method of spectral window smoothing for producing consistent estimates. The alternative procedure is the lag window or weighted covariance method, described on the next page.

The instruction WINDOW carries out the smoothing process. This estimates the spectral (cross-spectral) density at an ordinate by taking a weighted average of the periodogram at neighboring ordinates. For a continuous spectral density, there are two requirements for this to produce consistency:

- The window width must go to infinity as the number of data points increases (ensuring that the variance goes to zero). In a practical sense, this means that a window which is too narrow will produce imprecise estimates.
- The window width must increase at a rate slower than the increase in the number of data points (ensuring that the bias goes to zero). Translation: a very wide window will flatten the peaks and troughs too much.

Continuing from above, the following smooths 3 and 4 with a flat window of width 13, producing series 5 and 6.

\[
\text{window(width=13) 3 / 5} \\
\text{window(width=13) 4 / 6}
\]
Tapers

The smoothing method, particularly with the default flat window, can produce some spurious ripples in the estimate due to “data window leakage.” The data set is, theoretically, just a piece of an infinite data set, and we cut it off sharply at both ends. Data window leakage is the result of Fourier transforming this sharp-edged data window.

A taper reduces this by scaling the ends of the data so they merge smoothly with the zeros on either side. Apply the RATS instruction TAPER to the unpadded part of the series prior to the FFT instruction. Tapering has little effect when you use WINDOW with TYPE=TENT. When you apply a taper, you need to change the scale factor on CMULTIPLY to 1.0/(2*%PI*%SCALETAP), where %SCALETAP is computed by TAPER.

Lag Windows

Lag window or weighted covariance estimators were the favored method for computing spectral densities before the Fast Fourier Transform was developed. They are described in Koopmans (1974) on pages 325-6. A simple form of this is used in modern econometrics in the Newey-West covariance estimator.

The periodogram and the covariogram are inverses of each other under Fourier transformation. The periodogram gives inconsistent estimates of the spectrum because it weights equally (in Fourier transformation) the covariances with a small number of lags, which are estimated with almost a full set of data, and those at high lags, which are computed with almost none. In lag window estimation, we drop a window over the covariogram which gives zero weight to the high lags. The requirements for consistency here is that the number of covariances given non-zero weight goes to infinity at a rate slower than the increase of the number of data points.

If you want to employ the lag window method, the simplest way to compute the covariogram is to inverse transform the periodogram. You must pad to at least double the number of data points. The “windowing” process is done by setting up a weight series with the desired form. Make sure to symmetrize it by making the end terms match up with the starting ones. This does a Bartlett window of 13 lags applied to a data set with 130 data points (padded to 320). Series 2 will be the estimate of the spectral density. We sneak the 1/2π factor in when we multiply 1 by 2.

```plaintext
compute k=13.0
frequency 2 320
rtoc
#  X
#  1
fft 1
ctmul(scale=1./130) 1 1
ift 1
ctset 2 = %if(t<=k, 1-(t-1)/k, %if(t>320-k, 1-(321-t)/k, 0.0))
ctset 2 = (1.0/(2*%pi))*%z(t,1)*%z(t,2)
fft 2
```

These three lines transform series 1 into the covariogram of the input series.
Cross-Spectral Statistics

The examples at the start of the section constructed an estimate of the cross-spectral density. Usually, we are interested not in the density itself (which is complex-valued) but statistics derived from it:

- **Phase lead**: the fraction of a cycle by which one series leads (lags) the other at each frequency.
- **Coherence (squared)**: the proportion of the variance of either series which can be explained (linearly) by the other, again frequency by frequency.

RATS provides the instruction `POLAR` for computing these statistics. It does a polar decomposition of a series. The phase can be obtained directly from the cross-spectral density, but the coherence requires that we normalize the density by the product of the square roots of the spectral densities of the series. Continuing the example:

```plaintext
cmultiply(scale=1./(2*%pi*points)) 2 2 / 7  need spectrum of 2
window(width=13) 7 / 8
cset 7 = %z(t,7)/%csqrt(%z(t,6)*%z(t,8))
polar(periods) 7 / 8 9
```

Series 8 is the coherence, 9 is the phase lead of series 1 over 2 (since we `CMULTIPLY`ed in that order). The `PERIODS` option converts the phase lead from radians to periods. Note that the phase lead is almost meaningless (and very poorly estimated) when the coherence is small.

Procedures

The procedures `SPECTRUM` and `CROSSPEC` (on the files `SPECTRUM.SRC` and `CROSSPEC.SRC`) compute and (optionally) graph a spectral density, and coherence and phase, respectively.

```plaintext
@crosspec( options ) series1 series2 start end
@spectrum( options ) series start end
```

The series input to the procedures are REAL series. Most of the options are common to both:

- `ordinates=number of ordinates [depends upon length,seasonal]
- `window=[flat]/tent/quadratic`
- `width=window width [depends upon ordinates]`
- `taper=trapezoidal/cosine/[none]  (spectrum only)`
- `wtaper=taper width as a fraction of length [.25] (spectrum only) [graph]/nograph`
- `header=header for high-resolution graph`
- `footer=footer for high-resolution graph`
15.8 Frequency Domain Filtering

Uses of Filtering

Many applications of spectral analysis use some form of frequency domain filtering:

• Sims’ seasonal adjustment procedure uses a “filter” which can only be represented in the frequency domain.
• Hannan’s efficient estimator does GLS for arbitrary (stationary) serial correlation structures by computing in the frequency domain a filter to whiten the regression residuals.
• The spectral forecasting procedure uses a transfer function computed in the frequency domain.
• Maximum likelihood in the frequency domain relies upon frequency domain filtering to estimate time domain models.

Technical Information

Filtering is a convolution operation, which can be represented easily in the frequency domain. If

\[ y_t = \sum_{s=-\infty}^{\infty} a_s x_{t-s} , \text{ then} \]

\[ \tilde{y}(\omega) = \tilde{a}(\omega) \tilde{x}(\omega) \]

that is, the complicated convolution operation (1) is converted into simple multiplication in the frequency domain. The filtering procedure is thus:

1. Apply **FFT** to \( x \) to get \( \tilde{x} \).
2. Compute \( \tilde{a} \). You can use the instruction **TRFUNC** if it is a standard finite filter, **CMASK** for a filter to shut out frequency bands, and **CSET** or some other combination of instructions for other filter types.
3. Multiply the two Fourier transforms. Remember that RATS conjugates the second item in a **CMULTIPLY** or * operation.
4. Inverse transform the product using **IFT**.

Frequency domain filtering is most useful when the filter \( a \) is theoretically infinite but has a relatively simple Fourier transform. If \( a \) has a small number of coefficients, direct calculation with **FILTER** or **SET** is quite a bit faster.
Wraparound Effect

Consider the output obtained by passing a series \( y(t) \) through the filter \( A(L) \). If you do this in the time domain using \texttt{FILTER}, there are entries on one or both ends which cannot be computed for lack of lags or leads of \( y \). If you filter in the frequency domain, the effect is as if three copies of \( y \) were laid end to end. The filter \( A \) is applied to this new series, with the middle section extracted as the output.

\[
\begin{array}{c|c|c}
1, & 2, & ... , N-1, N, \text{ } 1, & 2, & ... , N-1, N, \text{ } 1, & 2, & ... , N-1, N \\
\hline
A(L)y(N)
\end{array}
\]

If we look at the entries which we cannot compute with \texttt{FILTER}, we see that they get computed in the frequency domain, but use data from both ends of the series. For instance, a filter which lags back to the non-existent entry 0 will take entry \( N \) in its place. This \textit{wraparound effect} hits only those entries which we could not compute anyway. The other entries are unaffected.

However, the time domain filter equivalent to most important frequency domain filters is an \textit{infinite} lag polynomial. Thus all entries will be affected by wraparound to some extent. To reduce the seriousness of this, the series should:

- have mean zero
- have no trend
- be well-padded

That way, the wrapping will be picked up by the zero padding at the ends of the data. With a mean zero trendless series, these zeros will not create the magnitude of bias that would be created if, for instance, there were a decided trend in the data.

Our recommendation is that you pad to at least double the length of the data.
Example 15.1 Deseasonalization

Sims (1974b) describes a procedure to deseasonalize data by removing all power from frequencies in a band about the seasonals. This is a frequency domain filtering procedure where the transfer function is a seasonal mask constructed using \texttt{CMASK}. The width of the seasonal bands depends upon the width of the peaks in the spectrum. The data in this example are Canadian retail sales. The example file is \texttt{DESEASON.PRG}.

\begin{verbatim}
open data oecdsample.rat
calendar(m) 1960

    Data are Canadian retail sales

data(format=rats) 1960:1 2007:3 canrett canretts

    Transform to logs and take the trend out

set logret = log(canrett)
filter(type=hp) logret / removed
set prepped = logret-removed
*
compute nords=2*%freqsize(2007:3)
frequency 2 nords
*
rtoc
# prepped
#    1
fft 1

    Create bands of width $\pi/16$. Leave the low frequencies in.

cmask 2 / (nords/12) (nords/32) (nords/12)+1
cmult 1 2
ift 1

    Send the deseasonalized data back, add back the removed trend and exponentiate.

ctor 1960:1 %allocend()
# 1
# deseason
*
set deseason = exp(deseason+removed)

    Graphs the deseasonalized series with the "officially" adjusted series.

graph 2
# canretts
# deseason
\end{verbatim}
Example 15.2 Hannan Efficient

Hannan’s Efficient (Hannan (1963)) is a frequency domain procedure for serial correlation correction. It allows for arbitrary patterns of serial correlation as long as the disturbance process is covariance stationary. It may be applied to any regression for which GLS is consistent. The idea is this: for the model

\[ y_t = X_t \beta + u_t \]

we need a filter which transforms the residuals \((u)\) to white noise. In the frequency domain, this means we are seeking a filter \(a(L)\) whose transfer function \(\tilde{a}(\omega)\) satisfies:

\[ |\tilde{a}(\omega)|^2 f_u(\omega) = \text{constant} \]

Any filter which satisfies this will have the desired effect. For convenience, we can choose the real-valued solution:

\[ \tilde{a}(\omega) = \sqrt{1/f_u(\omega)} \]

Note that the “filter” used is very different from the standard autoregressive filters used for serial correlation correction in the time domain. Because it has a real-valued transfer function, the filter is, in fact, symmetric and two-sided. Thus, the filtered series will look very different from series quasi-differenced in the time domain.

To implement Hannan’s Efficient estimator, we need to compute the reciprocal square root of the spectral density of the residuals, and filter the dependent and independent variables using it. The actual regression is still done in the time domain. Because the number of variables which must be filtered depends upon the regression, this is not as readily converted into a standardized procedure as the other techniques. This example demonstrates the process: (example file HANNAN.PRG).

```
cal(m) 1977:3
open data haversample.rat
data(format=rats) 1977:3 2007:4 fcm30 ftbs3

Do OLS regression. Save the lag coefficients in a series

linreg fcm30
# constant ftbs3{0 to 12}
set olslags 1 13 = %beta(t+1)

Do the Hannan filter on the independent and dependent variables.

inquire(lastreg) n1 n2
compute nords = 2*%freqsize(%nobs)
freq 3 nords
rtoc
# %resids fcm30 ftbs3
# 1 2 3
```
Chapter 15: Spectral Analysis

Compute the transfer function of the whitening filter

\[ \text{fft 1} \]
\[ \text{cmult 1 1} \]
\[ \text{window 1} \]
\[ \text{cset 1} = 1./%csqrt(%z(t,1)) \]

To make the HE regression more comparable (in terms of such things as sum of squared residuals), normalize the transfer function so it represents a filter with a 1 at 0 lag. Because the IFT has a $1/N$ divisor, we need to rescale to make the sum of the entries of the IFT to be $N$. Note that this will have no effect on the lag coefficients, since both the filtered dependent and explanatory variables are scaled by the same factor.

\[ \text{sstats 1 nords %real(%z(t,1))>>sumtr} \]
\[ \text{cset 1} = %z(t,1)*nords/sumtr \]
*  
\[ \text{dofor i = 2 3} \]
\[ \text{fft i} \]
\[ \text{cmult i 1} \]
\[ \text{ift i} \]
\[ \text{end dofor} \]

Send the filtered variables back to the time domain and run the GLS regression.

\[ \text{ctor} \]
\# 2 3
\# lfilt sfilt
*  
\[ \text{linreg(title="Hannan Efficient") lfilt n1 n2} \]
\# constant sfilt{0 to 12}
\[ \text{set hannanlags 1 13} = %beta(t+1) \]
*  
\[ \text{graph(number=0,header="Distributed Lag Coefficients","key=below,klabels=||"OLS Estimates","Hannan GLS Estimates"||)} 2 \]
\# olslags
\# hannanlags

Band Spectrum Regression

Another type of frequency domain regression procedure is Engle’s band spectrum regression (see Engle (1974)). In this procedure, the “filtering” procedure is to zero out certain bands in the Fourier transforms of the dependent variable and regressors. This is basically identical to Hannan efficient, except that a masking series is used instead of the one computed there. For instance, to include just the lowest $3/4$ of frequencies, use

\[ \text{cset 1} = t<=nords*3/8.or.t>nords*5/8 \]

in place of the FFT, CMULT, WINDOW and CSET instructions.
Spectral methods can be used to forecast univariate time series. The `SPECFORE` procedure (available in the file `SPECFORE.SRC`) implements the algorithm described here, which follows a suggestion made by John Geweke.

The basis of the method is the moving average representation:

\[ X_t = c(L) \varepsilon_t, \text{ where } c(0) = 1 \text{ and } \varepsilon \text{ is fundamental for } X \]

Box-Jenkins techniques attempt to represent \( c \) as a rational function: the numerator is the moving average part, the denominator is the autoregressive. Spectral methods allow us to compute an estimate of the Fourier transform of \( c \), which can be used to compute forecasts. Their advantage over Box-Jenkins is that the same “model” applies to all data series, so the forecasting process is automatic. That, unfortunately, is one of the disadvantages as well—see the cautions below.

The following derivation can be found in Koopmans (1974), pp. 235-237. Write the spectral density of \( X \) as the \( z \)-transform

\[ f_X(z) = c(z) c(z^{-1}) \sigma^2 \]

Under reasonable regularity conditions on \( c \), \( \log(f_X) \) has a Laurent expansion:

\[ \log f_X(z) = d(z) + d(z^{-1}) + d_0 \]

where \( d \) is a one-sided polynomial in positive powers of \( z \). If we take the \( \exp \) of both sides of (3), and combine it with (2), we get:

\[ c(z) c(z^{-1}) \sigma^2 = e^{d(z)} e^{d(z^{-1})} e^{d_0} \]

To get the Fourier transform of \( c \) we need to

1. compute the log spectral density of \( X \)
2. mask the negative and zero frequencies to get the Fourier transform of \( d \)
3. take its \( \exp \), frequency by frequency (again, omitting the negative and zero frequencies)

We now have a “non-parametric” estimate of the moving average representation of the series. Once we have an estimate of the FT of \( c \), we divide the FT of \( X \) by the FT of \( c \) to get the FT of \( \varepsilon \) process, set \( \varepsilon \) to zero outside of the range of actual data, then filter \( \varepsilon \) by \( c \) (in the frequency domain) to get the forecasts.

The code from the procedure which accomplishes this is provided on the next page.
Cautions

As with other frequency domain filtering techniques, the series must be well–padded. Also, please note the following:

- Data series which can be adequately represented by a Box-Jenkins model with few parameters will usually be better forecast using that technique, as the model is more sharply estimated.
- Spectral methods require more data (100+ data points to be safe) than other techniques, since they do not use it as efficiently.
- The series you run through the model must be stationary: proper differencing is vital.

Code

Compute the log spectral density. Scale factors will wash out.

```
fft 1
cmult 1 1 / 2
window 2 / 3
cln 3

Back to sequence-space to shut out the negative powers in the Laurent expansion.
FT back and exponentiate to get FT of exp(D(Z)).

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

IFT to get the sequence c. Normalize so that C(0)=1

ift 3
cset(scratch) 3 = %z(t,3)/%z(1,3)

Filter the input series by 1/C(L) to get residuals. Mask it outside the data range.

fft 3
cset 1 = %z(t,1)/%z(t,3)
ift 1 / 2
cset 2 = %z(t,2)*(t>=startl.and.t<=endl)

Filter the residuals E by C(L) in the frequency domain

fft 2 / 1
cset 1 = %z(t,1)*%conjg(%z(t,3))
ift 1
```
15.10 Fractional Integration

Fractional integration (or fractional differencing) was devised independently by Granger and Joyeux (1980) and Hosking (1981) to provide a way to model a process with “long-memory”—defined as autocorrelations which damp off at a slow enough rate that they sum to infinity. Standard ARIMA models can only approximate this type of behavior, as a non-integrated ARMA model has autocorrelations that are exponentially damped, and are thus summable, and any integrated model has autocorrelations which don’t damp at all.

Fractional integration is modelled with the help of the specialized lag function \((1 - L)^d\) where \(d\) is non-integer. This can be expanded in the time domain using a generalized binomial expansion. In fact, in RATS, the coefficient on lag \(k\) can be obtained as \%BINOMIAL(d, k) * (-1)**k. This is an infinite one-sided lag polynomial.

There has been only a modest amount of empirical work done in econometrics using fractional integration, particularly when compared with the amount done using unit root statistics. Part of this is clearly due to the fact that a first difference is easy to compute and has a relatively simple economic interpretation, while a fractional difference can’t even be computed exactly in the time domain—the filter always has to be truncated. It is also true that fractional integration is another entrant in the set of models proposed to explain long term behavior of data sets which in many cases simply aren’t long enough to separate one clearly from the others.

Fractional integration is a natural candidate for attack via the frequency domain since the lag function \((1 - L)^d\) can be computed easily there. For instance, Geweke and Porter-Hudak (1983) proposed a method for estimating \(d\) which involves analyzing the behavior of the periodogram of the data for low frequencies. The procedure GPH.SRC provided with RATS implements this. A more recent refinement of this by Andrews and Guggenberger (2003) is available on AGFRACTD.SRC.

If all you need to do is fractionally difference a series with a known value of \(d\), you can use the \texttt{DIFFERENCE} instruction with the option \texttt{FRACTION} (and possibly \texttt{DIFFERENCE}). The frequency domain computations described here are used by \texttt{DIFFERENCE}.

The program below estimates an ARFIMA model (AutoRegressive, Fractionally Integrated, Moving Average) model using frequency domain techniques. This is done using the instruction \texttt{MAXIMIZE} applied to a criterion function which is defined frequency by frequency. This function, which can be used more broadly than just for ARFIMA models, looks similar to those used in time domain likelihoods. However, where you would usually see a squared residual divided by a variance, it uses the ratio between the periodogram of the data and the spectral density implied by the model and its parameters. Fox and Taqqu (1986) work through the theory of this method of estimation. It’s possible to concentrate out the innovation variance, leaving a simpler function than the one shown here, but we’ve found that leaving the function in its original form works better.
An ARFIMA model takes the form

\begin{equation}
(1 - L)^d A(L)y_t = B(L)u_t
\end{equation}

where \(A\) and \(B\) are standard autoregressive and moving average polynomials.

Solving this out gives

\begin{equation}
y_t = (B(L)/A(L))(1 - L)^d u_t
\end{equation}

The square of the transfer function of the lag polynomial on the right is the spectral density function that we want (except for scale by the variance of \(u\)). The easiest way to compute this is to use a complex-valued \texttt{FRML} to give us the transfer function itself. This makes it simpler to change from one model to another, since the model is entirely described by the transfer function.

If you use this method of estimation, there are two things you must be sure to do:

1. Run the \texttt{MAXIMIZE} instruction across the full range of frequency domain ordinates. This is, almost certainly, longer than your actual data set.
2. Include the extra factor of actual observations divided by the number of frequency ordinates. Because of the procedure’s reliance on frequency domain filtering, it only works acceptably if the data are padded. This factor corrects for the effect of padding.

\textbf{Example 15.3 Fractional Integration}

This example is available on the file \texttt{FRACTINT.PRG} included with RATS.

```
open data oecdsample.rat
calendar(q) 1960
data(format=rats) 1960:1 2007:2 usaunempsurqs

The data series is the US unemployment rate. The mean is removed after transformation to square roots.

set usasur = sqrt(usaunempsurqs)
diff(center) usasur / unemp0

declare frml[complex] transfer
nonlin a b d ivar

Construct the transfer function of the moving average polynomial. This example includes one MA lag and one AR lag in addition to the fractional difference. The \%CONJG function on the final term is necessary in the next function since the * operator conjugates the right operand, and we want a straight multiply.

frml transfer = (1+b*%zlag(t,1))/$
((1-a*%zlag(t,1))*%conjugate((1-%zlag(t,1))**D))
```
Run an AR(1) to get a guess for the innovation variance.

```
linreg(noprint) usasur
# constant usasur{1}
compute a=.9,b=0.0,d=.5,ivar=%seesq

Use double the recommended number of ordinates. Send the data to the frequency domain and compute the periodogram.

compute nobs = %allocend()
compute nords = 2*%freqsize(nobs)
freq 3 nords
rtoc
# unemp0
# 1
fft 1
cmult(scale=1.0/nobs) 1 1 / 2
set periodogram 1 nords = %real(%z(t,2))
frml likely = trlambda=transfer, %logdensitycv($ivar*%cabs(trlambda)**2,periodogram,(float(nobs)/nords))
maximize(pmethod=simplex,piters=5,method=bfgs) likely 1 nords

Dividing the FT of the series by the MAR transfer function gives the FT of the residuals. Inverse transform and send the relevant part back to the time domain.

cset 3 = %z(t,1)/transfer(t)
ift 3
citor 1 nobs
# 3
# resids

Check out the correlation of residuals. We skip the first few residuals in this as they are likely to be very large with a correspondingly large variance.

corr(qstats,span=24) resids 3 *

Compute the (approximate) moving average representation. The transfer function doesn’t exist at zero frequency for d>0. (The 0**0 calculation in computing “transfer” produces a 0). Since the MAR is known to have a unit lead coefficient, we know what we need to add to the inverse transform to achieve that. We just add that same adjustment term to all the lags to produce (approximately) the correct result.

cset 3 1 nords = transfer
ift 3
compute adjust=1-%z(1,3)
cset 3 1 nords = %z(t,3)+adjust *
set mar 1 100 = %real(%z(t,3))
graph(style=bargraph,number=0,$
   header="Moving Average Representation")
# mar 1 100
```
Chapter 16
Advanced Programming Tools

While not the longest chapter in the book, this final chapter may be the most important. It’s the programming tools described here that give RATS its tremendous power. Even if you don’t use these yourself, you are an indirect consumer every time you use a procedure written by someone in the RATS community.

RATS provides an environment in which a user can, in effect, design his or her own instructions to do statistics in the manner desired. With the more advanced techniques, it’s possible to create a fully menu-driven application.
16.1 Program Control and Compile Mode

Loops, Conditional Blocks, and Procedures

RATS offers several tools for program control. You can

- execute instructions only if a certain condition holds (or fails to hold) using **IF** and **ELSE**.
- loop over a range of values or a list with **DO**, **DOFOR**, or **LOOP**.
- execute a loop under the control of a condition using **WHILE** or **UNTIL**.
- write entire subprograms with **PROCEDURE** and **FUNCTION**.
- use the **{** and **}** symbols (or the equivalent keywords **BEGIN** and **END**) to define a “block” of instructions. Usually, these are used to execute a block of several instructions following a conditional statement when that conditional statement is itself part of a compiled section (such as a loop or procedure).

Procedures offer so much power, and are so useful, that we have put a discussion of their special properties in a separate section (16.2).

Compile Mode

Ordinarily, RATS executes instructions in order, processing one instruction completely before going on to the next. However, loops, conditional blocks, procedures (and the **{** and **}** symbols) put RATS into *compile mode*: RATS reads in and interprets an entire block of instructions. Rather than executing them immediately, it converts them into an internal code and stores this code. When RATS reaches an **END** instruction, signalling the end of the structure, it begins executing the compiled code (except for a **PROCEDURE** or **FUNCTION**, which are executed only on command).

Instruction Locations

When RATS processes a compiled structure, it prefixes the actual code on each line by a pair of numbers in parentheses, which can be very helpful in debugging problems with compiled sections. (Note: these numbers normally aren’t displayed when running in interactive mode, but you can see them by using a **SOURCE(ECHO)** command to execute your program file.) The first number is the structure level *at the end of the previous line*. The second number is the location of the instruction within the compiled section. Consider the following example, taken from the **DFUNIT** procedure:

```
(01.0155) inquire(series=series) start1<<start endl<<end
(01.0207) *
(01.0207) set sdiff start1+1 endl = series-series{1}
(01.0271) set strend start1+1 endl = t
```

If an error occurs during execution of the compiled section, the error message will include a line indicating the location of the error. For example, if you were to try to apply **DFUNIT** to a series that contained no data, you would get the message:
## SR10. Missing Values And/Or SMPL Options Leave No Usable Data Points

The Error Occurred At Location 0206 of DFUNIT
Line 17 of DFUNIT

The first line in the traceback says the occurred at location 0206. That tells us the error was generated by the INQUIRE instruction, which occupies memory locations 155 through 206. The second traceback line gives the same information, but tells what line in the procedure caused it. *Note that this is the line number from the start of the actual procedure or loop*. Most procedure files have a header of comments at their top, but the line count used here doesn’t start until the PROCEDURE statement.

The structure level is helpful in locating block nesting errors. Consider this code from later in the procedure:

```plaintext
(02.1417) if lags {
(03.1432)   summarize(noprint)
(03.1437)   # sdiff{1 to lags}
(03.1493)   compute fiddle=1.0/(1.0-%sumlc)
(03.1524) }  
(02.1524) else
```

Suppose that we forgot the `}` before the ELSE. The ELSE location would be prefixed with 03, instead of 02 which is what we would expect (remember that the number prefixing a line is the level at the end of the previous line).

### Loops

RATS offers five different looping instructions. Two of these (DO and DOFOR) were first introduced in Section 1.9. (A %DO function is also available, but it is not used for program control—see Section 2, DO, or EWISE in the Reference Manual).

**DO** is a standard loop instruction which loops over an integer index. It is like a FORTRAN DO or a BASIC FOR.

**DOFOR** is less standard but often very useful. It loops over a list of items: series, matrices, numbers, etc.

**WHILE** loops as long as an expression is true.

**UNTIL** loops until an expression is true. It always executes the enclosed instructions at least once; by contrast, WHILE can fail to loop a single time.

**LOOP** loops unconditionally. You need to use a BREAK instruction to get out of the loop.

With any of the loops, you can use NEXT to skip directly to the next pass or BREAK to break out of the loop entirely. Note that, if you have nested loops a NEXT or BREAK applies only to the loop in which the instruction is located. DO, DOFOR and LOOP have a somewhat different setup from WHILE and UNTIL. They all loop over the instructions down to a matching END. Some examples follow on the next page.
do end=1999:1,2006:12
   linreg y 1970:1 end
   # constant y{1}
end do

This does regressions over a changing range of entries. All the regressions have the same starting date (1970:1), while the ending periods range from 1999:1 through 2006:12. This takes advantage of the fact that dates in RATS are handled as integer entry numbers.

dofor i = realgnp to realbfi
   set(scratch) i = log(i{0}/i{1})
end dofor

This does a percent change calculation for series from REALGNP to REALBFI. We use I{0} rather than just I because I (an INTEGER variable) is perfectly legal by itself in an expression. Including the lag notation tells RATS to use series number I.

**WHILE** and **UNTIL** loop only over the next instruction or block of instructions. When you want the conditional loop to include more than one instruction, you must enclose them within braces { and }.

```plaintext
compute total=0.0, count=0
while total<100.0 {
   compute count=count+1
   compute total=total+series(count)
}
end
```

The **END** is needed if this is *not* within a larger compiled section. It puts RATS out of compile mode, so it can execute the instructions. If the conditional instruction is located inside a larger compiled section, the matching **END** instruction is omitted. For example:

```plaintext
if docount==1 {
   compute total=0.0, count=0
   while total<100.0 {
      compute count=count+1
      compute total=total+series(count)
   }
} end if
```

The **IF** command puts RATS in compile mode, and thus a matching **END** is required to close the **IF** block. However, because the **WHILE** statement occurs inside a compiled section, no matching **END** instruction is required for the **WHILE** block.
16.2 Procedures and Functions

What Are They?

The most powerful compiler structures are procedures and functions. In effect, procedures allow you to create your own “instructions” from a sequence of other RATS commands, while functions define callable functions similar to built-in functions like inv and %density.

We include many pre-written procedures with RATS, and dozens of others are available for downloading from our web site and other sources. These allow you to perform many complex tasks without having to code them yourselves. In the following pages, we’ll tell you how to make use of these existing procedures. Then, we’ll discuss the various elements of RATS procedures and guide you through the process of writing a complete procedure so you can get a feel for how it’s done. We’ll focus mainly on procedures. Functions are similar but can’t use all the features of a procedure. We’ll often use the word subprogram when discussing information that applies to both procedures and functions.

First, a look at the basic elements of a procedure: A procedure begins with a procedure statement and ends with a matching end. A procedure may have some, all, or none of the following special features:

- **Parameters**, the names of which are listed following the procedure name on the procedure statement. Use these to pass information to the procedure. Their types are set using the statement type.
- **Local variables**, recognized only within the procedure. They are set up by the statement local.
- **Options**, which operate much like those for a standard RATS instruction and are set up using the statement option.
- **Supplementary cards**, which are defined with the instruction enter.

A function begins with function, ends with a matching end, and can include parameters and local variables, but not options and supplementary cards.

Comparison With Other Structures

Unlike other types of compiled structures, these are not executed automatically when the processing of its code is finished. Instead, RATS saves the generated code internally and only executes it when it receives an execute instruction (or the @procedure shortcut) for a procedure, or sees function name(arguments) for a function. They can be invoked any number of times, again differing from the other structures, which execute only once.

Because a subprogram is not a loop, you cannot use break to leave it before the end. Use return instead. Also, if you have calls to a subprogram nested inside another subprogram, you must compile the code for the inner subprogram first, so that RATS will recognize the procedure or function call when you compile the main routine.
16.2.1 Using Existing Procedures

Storing Procedures in External Files

Procedures and functions do not need to be in the same file as the program which uses them. There are several advantages to keeping them on separate files:

- If you have a subprogram which can be used by more than one program, you only need to maintain one copy of the subprogram.
- Keeping subprograms on separate files means that your main program files will be shorter and easier to edit and debug.
- You can call in a subprogram even when you’re working interactively.

The RATS software package includes a number of pre-written subprograms which extend the capabilities of the program by making it possible to perform complex tasks via a single call. Dozens of other procedures, written by Estima and by other RATS users around the world, are available for downloading from Estima’s Web site, as well as other sites maintained by other users and organizations.

Before you can execute a subprogram stored on a separate file, RATS needs to execute (or “compile”) the instructions on that file. In many cases, RATS can locate and execute the appropriate file automatically. Otherwise, you can use SOURCE to tell RATS which file to execute. See below for details.

First, Open and Read the File!

If you want to use one of the procedures supplied with RATS or downloaded from a website, the first thing to do is to open the file and read through the comments at the top. These are ordinary text files, so you can open them by doing File–Open in RATS, or any other text-editor or word-processor. All of the files we supply with RATS or make available on our web site include comments describing the procedure syntax, options available, and, usually, technical details and references as well.

If you create a procedure yourself that you wish to share with others, make sure you include similar information, so that the future users will know how to use it, and will know whom to thank.

Compiling the Procedure

To use a procedure or function that is stored on an external file, RATS will first need to execute the commands on the file. This “compiles” the procedure or function code, and stores it in memory. Procedures can then be executed using the EXECUTE command or (more commonly) @procedure. For functions, you simply use the function name (and any parameters) in an expression.

For procedures, the easiest approach is to let RATS locate and compile the procedure file automatically, by simply executing the procedure. If the procedure has not already been compiled, RATS will search the hard drive for a filename matching the name of the procedure, with an extension of .SRC. For example, if you do @DFUNIT,
RATS will search for a file called `DFUNIT.SRC`. It will start searching in the “Procedure Directory” specified in the `File–Preferences` dialog. If necessary, RATS will then search the current directory and the directory containing the RATS executable file. If it still fails to find a matching file, you will see a dialog box prompting you to locate the file.

If you prefer, you can use an explicit `SOURCE` instruction to read in and execute the commands on an external file. This is required for functions (RATS won’t search for those), for procedure files where the procedure name does not match the file name, or for procedures located in a directory not included in the search path described above.

Suppose you are writing a program that will require the `INTERPOL` procedure (supplied with RATS on the file `INTERPOL.SRC`) to interpolate low-frequency data to a higher frequency. You might do:

```r
source c:\rats\interpol.src
calendar(m) 1990
all 1999:12
open data quarterly.rat
data(for=rats) / x1
@interpol x1 xm
```

The `SOURCE` instruction includes an `ECHO/NOECHO` option, which controls whether or not the instructions being executed are displayed in the output window or file. When trying to debug an error, or when using a procedure for the first time, you may find it handy to use the `ECHO` option to see the instructions echoed to the output window. If the file is located in a directory whose name includes blank spaces, enclose the whole path and name in quotes:

```r
source "c:\Program Files\Estima\WinRATS 7.0\interpol.src"
```

Another option is to use Procedure Libraries, as described below.

## Procedure Libraries

Procedure libraries are collections of procedures and functions which are brought in at the start of the program, and also whenever the program is reinitialized, using, for instance, `File–Clear Program`. This makes sure the procedures included in the library are always available. There are several ways to specify a procedure file:

1. The RATS instruction `ENV PROC=name of procedure library`
2. The `/PROC=name of procedure library` option on the command line.
3. Through setting the procedure library in the preferences.

You can form the library either by copying the full text of the desired procedures to a single file, or by setting up a file with a group of `SOURCE` instructions for the procedures you want.
Chapter 16: Advanced Programming Tools

16.2.2 Writing Your Own Procedures

Next, we’ll take a closer look at the elements of a procedure, and take you through the process of writing a complete procedure.

Instruction Order

The instructions that make up the subprogram must be in the following order:

- `PROCEDURE` or `FUNCTION` statement
- `TYPE` and `OPTION` instructions
- other RATS instructions
- `END`

The first line provides the subprogram name and lists the names of the formal parameters. `TYPE` sets the variable types of the parameters and the function, while `OPTION` instructions define any options for a procedure. `DECLARE` and `LOCAL` statements may be scattered among the other instructions, but it’s usually a good idea to keep them together before the executable instructions.

Learning How to Write Procedures

Start small!! If the first procedure you try to write uses 100 RATS instructions to do the latest wrinkle in econometric theory, you are likely to be working at it for a very long time. Begin with a more modest (or even trivial) operation and work up from there. We would suggest that you start with parameters only, then add local variables and finally options (if they make sense for your application). Get it to work correctly with proper input before putting in checks for garbage input.

Learn from our procedures!! We have provided many procedures which use the most advanced techniques that RATS has to offer.

Parameters

Parameters allow you to pass information stored in variables (data series, scalar variables, etc.) from the main program to a subprogram. You list the parameters on the `PROCEDURE` or `FUNCTION` statement itself following the subprogram name, with the parameters separated by blanks. We refer to these as `formal` parameters. You pass information to these formal parameters by specifying a matching set of `actual` parameters on the `EXECUTE` instruction or function reference. Within the subprogram, you manipulate the data using the formal parameter names. If you pass a parameter by address (see next segment), you can either use or set it within the procedure. If, however, you pass a parameter by value, you may only `use` it within the procedure.

By default, formal parameters of a `PROCEDURE` are all type `INTEGER`, passed by value, and those of a `FUNCTION` are `REAL`, again passed by value. The `FUNCTION` name itself is `REAL` by default. You will need to use one or more `TYPE` statements if you want them to take different types.
Passing Parameters: By Value vs. By Address

Parameters can be passed to a procedure in one of two ways: by address, or by value.

- Passing by address means that the actual memory address of a variable is passed to the procedure. If you change the value of the formal parameter inside the procedure, you will change the value of the actual parameter.
- Passing by value means that the only information made available to the procedure is the value of the actual parameter. The procedure is not allowed to manipulate the value of the formal (or actual) parameter. In fact, you will get a syntax error if you try to reset the value.

You should pass a parameter argument by address only if it is being set by the procedure. Anything that is only an input to the procedure should be passed by value.

You distinguish between the two by prefixing the parameter name with * on a TYPE statement if you want it called by address. Without the *, it is called by value.

```
proc forecast depvar start end forecast
type series depvar *forecast
```

Parameters DEPVAR and FORECAST are both type SERIES. DEPVAR, however, is called by value, and FORECAST by address. START and END are INTEGERS, passed by value. So, DEPVAR, START, and END will be inputs to the procedure and FORECAST will be the output that is computed and returned to the main program.

```
procedure condition nvar nfore nbeg vmat nconstr fcstout
type integer nvar nfore nbeg nconstr
  type vect[series] *fcstout
  type symmetric vmat
```

Parameters NVAR, NFORE, NBEG and NCONSTR are INTEGER passed by value, FCSTOUT is a VECT[SERIES] passed by address and VMAT is a SYMMETRIC passed by value. We could actually omit the first TYPE since INTEGER by value is the default for all parameters.

```
function sum num1 num2
type integer sum num1 num2
  compute sum = num1+num2
end
  compute a=5, b=10
  disp a "+" b "=" sum(a,b)
```

This computes the sum of two integer values, and returns the result as an integer. NUM1 and NUM2 are passed by value, so the function can use these variables, but it cannot change their value. The function name itself is always considered to be passed by address (there wouldn’t be much point otherwise), so you don’t need a * on it.
Local Variables

The variables in a RATS program are global unless you state otherwise—that is, you can use them within any part of the program, including procedures. There can be only one global variable with any one name.

A local variable (defined by LOCAL) can only be used in a subprogram, and only within the subprogram which defines it. RATS keeps global and local variables completely separate in memory, so a local variable can have the same name as a global one. If a name conflict arises, the local variable takes precedence.

If you plan to use a subprogram in many different programs, you should write it using only parameters, local variables, and variables defined internally by RATS. That way there can be no possibility of conflict with the variable names in the main program. If you want some variables computed within a procedure to be accessible to the user later (the way that %RSS and %BETA are for an instruction like LINREG), it is a good idea to use a special prefix (such as %%) on the variable names, to avoid the possibility of conflict with the user's own variable names, or with RATS reserved names in future releases.

LOCAL takes the same form as DECLARE.

procedure bayestst series start end
  type series series
  type integer start end
  local series sdiff strend
  local integer startl endl
  local real teststat margalph alphtest

The procedure BAYESTST has several local variables. Note particularly the use of local series. If you use a series locally, it is very important to declare it as such.

Checking Local Versus Global Declarations

When writing and testing a new procedure, you may find the instruction DEBUG(SYMBOL) helpful. If you do DEBUG(SYMBOL) and then compile your procedure code, RATS will report any variables that are created as global variables, or that are declared as local but aren't actually used. For example:

debug(symbol)
source kfilter.src

produces output like this:

KFILTER, line 53 Using TIME as INTEGER

indicating that TIME is being defined as a global integer, on line 53 of the procedure.
Chapter 16: Advanced Programming Tools

Options

The **OPTION** statement defines options for the procedure. (They can’t be used with a function). When you call the procedure, you can choose options in a field immediately after the procedure name, much like options on regular RATS instructions. You can also define default values or settings for procedure options. Like parameters, options can be passed by value or by address. To have it passed by address, prefix the formal parameter name with *.

See **OPTION** in the *Reference Manual* for a complete description—we just give some quick information here. For the three types (switch, choice and value), use

<table>
<thead>
<tr>
<th>Option Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch</td>
<td><code>option switch option name Default (1 for ON, 0 for OFF)</code></td>
</tr>
<tr>
<td>choice</td>
<td><code>option choice option name Default number List of Choices</code></td>
</tr>
<tr>
<td>data type</td>
<td><code>option data type option name Default value</code></td>
</tr>
</tbody>
</table>

- Option names are local to the procedure.
- You must use the full name for `option name` in your instructions within the procedure.
- When you invoke the procedure using `EXECUTE` or `@procedure`, only the first three characters of the option name are significant (just like standard RATS options). Make sure you do not use conflicting option names. For instance, `RESIDUALS` and `RESTRICT` both abbreviate to `RES`.
- *Do not use a name which begins with the letters NO*. This is reserved for turning off switch options.

Passing Information to RATS Instructions

RATS allows you to pass information to instruction options in a more flexible fashion than was described in Chapter 1. With all types of options (including switch and choice), you can use the syntax

```
instruction(option=expression or variable)
```

```
procedure nonsense
  option switch print 1
  option symmetric *vcvmat
  vcv(print=print,matrix=vcvmat)
end
```

In this example, the PRINT switch is on by default, so the PRINT option on **VCV** will also be on by default. If, however, you use the option NOPRINT when you execute **NONSENSE**, the PRINT variable will be “off” (value 0) which will turn off the PRINT option on **VCV**.

The **VCVMAT** option has no default. If you don’t use the option when you execute the procedure, the **MATRIX** option on **VCV** will behave as if you didn’t use it at all.
Chapter 16: Advanced Programming Tools

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.

Supplementary Cards

RATS allows you some flexibility in dealing with supplementary cards for RATS instructions within the procedure. You can either:

- code them immediately after the instruction in the procedure which requires them (if you know what the values on the card will be), or
- put them after the `EXECUTE` instruction which invokes the procedure. If an instruction with an omitted supplementary card is within a loop, `EXECUTE` must be followed by one supplementary card for each pass through the loop.

The latter method allows you to act as if the supplementary cards are for the `PROCEDURE` rather than for instruction within it which requires them. When a single instruction requires several supplementary cards, you must treat them all in the same fashion—you cannot include three in the procedure and leave out the fourth.

If you need a list of regressors, the best way to get them is by using an `EQUATION` instruction applied to a local equation. For instance:

```plaintext
local equation xvar
equation xvar *
```

This will process the regressor list that follows the line executing the procedure to define the variables list for the equation. For instance, this is from the ChowLin procedure, so

```plaintext
@chowlin loggdp
# constant logip logsales
```

will define `XVAR` (within the procedure) as an equation with `CONSTANT`, `LOGIP` and `LOGSALES` as its explanatory variables. You can then use the `%EQNxxxx` functions to work with this. For instance, `%EQNSIZE(XVAR)` will be the number of variables in the list, `%EQNTABLE(XVAR)` the table of series/lag combinations and `%EQNXVECTOR(XVAR, T)` will be the vector \( | 1, \text{LOGIP}(T), \text{LOGSALES}(T) | \)

If you need some other type of list of series (where lag notation isn’t permitted), bring it in as an `VECTOR[INTEGER]` array using `ENTER(VARYING)` and use that array when you need the list.
The INQUIRE Instruction

INQUIRE is very handy when you want to write a procedure which (perhaps optionally) determines the range itself. We use it in most of our procedures.

```plaintext
procedure stockwat series start end
type series series
type integer start end
option integer arcorr 1
*
local integer startl startm endm
inquire(series=series) startm<<start endm<<end
inquire(series=series) startl
if .not.%defined(start)
    compute startm=startl+arcorr+3
```

The first INQUIRE sets STARTM to the value of the parameter START if the user provides one, otherwise it will be the first defined entry of SERIES. ENDM is similar. The second INQUIRE just determines the first defined entry of SERIES. If there was no value for START, the COMPUTE sets STARTM to begin ARCORR+3 entries into SERIES.

The FIXED Statement-Creating Lookup Tables

FIXED can be used in a subprogram to set up a lookup table for (for instance) critical values. It creates a real-valued array (or array of real-valued arrays) whose dimensions and content are known constants. You can't use expressions in the dimensions or the values.

FIXED can set up only one array per statement, which is followed by its values. It can be placed anywhere within the subprogram, though it generally makes sense to put it near the top, typically between the other declaration statements and the first executable instructions. The array that it creates is local to the subprogram.

```plaintext
fixed rect trace95(4,10)
  4.07 12.28 24.21 40.10 59.96  83.82 111.68 143.53 179.38 219.23
  3.84 15.41 29.80 47.71 69.61  95.51 125.42 159.32 197.22 239.12
  3.84 18.15 34.56 54.11 77.79 104.76 135.75 170.15 208.53 250.53
  9.14 20.16 35.07 53.94 76.81 103.68 134.54 169.41 208.27 251.13
```
16.2.3 Regressor List Functions

RATS provides several functions for working with regressor lists, and these can be helpful in writing procedures, particularly when you need to use lags. A regressor list is represented by a VECTOR[INTEGER] which codes information about the series and lags. Use these functions to manipulate the list; don’t try to do it yourself.

%rladdone(R,S)  Add a regressor to a regressor list
%rladdlag(R,S,L)  Add a lagged regressor to a regressor list
%rladdlaglist(R,S,L)  Add a list of lags to a regressor list
%rlconcat(R1,R2)  Concatenate two regressor lists
%rlcount(R)  Return the number of regressors in a list
%rlempty()  Creates an empty regressor list
%rlfromtable(Table)  Convert a regression table to a regressor list
%rlverify(R)  Verify a regressor list

In earlier versions of RATS, procedures often used ENTER(VARYING) instructions to build up regressor lists. The regressor list functions are somewhat cleaner and they can handle lag lists more flexibly. The following builds up a list of the dependent variables and regressors for a cointegration procedure. The original dependent variables are in the VECT[SERIES] V, while the differenced versions are in DV. The list needs to include:

1. The current differenced variables (added with %RLADDONE)
2. The first lag of the original variables (added with %RLADDLAG)
3. L – 1 lags of each differenced variable (added with %RLADDLAGLIST)
4. The deterministic variables, if needed (%RLADDONE)

First, we define an empty regression list called REGLIST. We then use other regressor list functions to add additional terms to the existing list and save the result back into REGLIST.

compute reglist = %rempty()
do i=1,numvar
   compute reglist=%rladdone(reglist,dv(i))
end do i
do i=1,numvar
   compute reglist=%rladdlag(reglist,v(i),1)
end do i
if lags>1
   do i=1,numvar
      compute reglist=%rladdlaglist(reglist,dv(i),$
                     %seq(1,lags-1))
   end do i
if determ>=2
   compute reglist=%rladdone(reglist,constant)
if determ>=3
   compute reglist=%rladdone(reglist,trend)
16.2.4 Putting It All Together: An Example

We will now develop a **PROCEDURE** for implementing a General LM test for serial correlation from Section 6.5. We will start with a very simple procedure, concentrating upon getting the calculations correct, and then we refine it.

We need to input the following information to the procedure:

- The dependent variable
- The regressors
- The number of lags to test

To make this look as much like a RATS instruction as possible, the dependent variable should be a parameter, the regressors should come from a supplementary card, and the number of lags should be an option.

The only tricky part about our first attempt is the handling of the list of regressors. We need it in two places: the initial regression and the auxiliary regression. We pull it in with the **EQUATION** instruction, then extract that list with the `%RLFROMTABLE` and `%EQNTABLE` functions. (%EQNTABLE(REGEQN) is the table of regressors (series/lag combinations) from REGEQN).

```plaintext
procedure sctest depvar
  type series depvar
  *
  option integer lags  1
  local equation regeqn
  local vect[integer] reglist
  *
  equation regeqn depvar
  linreg(equation=regeqn)
  compute reglist=%rlfromtable(%eqntable(regeqn))
  *
  linreg(noprint,dfc=%nreg) %resids
  # reglist %resids{1 to lags}
  disp "LM statistic" %trsq "Signif Level" %chisqr(%trsq, lags)
end
```
We should now check against the direct code in Section 6.5 to make sure the calculations are correct. After this, we can make refinements. We will make four changes:

- We will improve the output. This is a straightforward change to the `DISPLAY` instruction at the end of the procedure.
- We will add a `PRINT` option, to allow the user to suppress (with `NOPRINT`) the output from the first regression.
- We will have `SCTEST` define the standard variables `%CDSTAT` and `%SIGNIF`.
- We will allow the user to specify a start and end range.

The last is the only difficult part of this. The range is easy to handle if we either require the user to give an explicit range, or if we always use a default range. It gets complicated when we let the user do either. This is where `INQUIRE` comes in. We can determine the range of the initial regression by using an `INQUIRE` instruction with the `REGRESSORLIST` option. If the user passes values to the `START` or `END` parameters, they override the values from `INQUIRE`. Once we have values for the range on the first regression, the auxiliary regression just starts `LAGS` entries farther up.

```plaintext
procedure sctest depvar start end
  type series depvar
  type integer start end
  *
  local vect[integer] reglist
  local integer startl endl
  local equation regeqn
  *
  option integer lags 1
  option switch print 1
  *
  equation regeqn depvar
  compute reglist=%rlfromtable(%eqntable(regeqn))
  inquire(regressorlist) startl<<start endl<<end
  # reglist depvar
  linreg(print=print,equation=regeqn) depvar startl endl
  *
  linreg(noprint,dfc=%nreg) %resids startl+lags endl
  # reglist %resids{1 to lags}
  compute %cdstat=%trsq , %signif=%chisqr(%trsq, lags)
  disp "LM test for Serial Correlation of Order " lags
  disp "Test Statistic " %trsq "Signif Level " #.##### %signif
end
```

Suppose we want to use the procedure to regress $Y$ on three regressors (a constant, $X_1$ and $X_2$) and test the residuals for serial correlation with a lag length of 4. We can do that with the following instruction:

```plaintext
@sctest(lags=4) y
# constant x1 x2
```
16.3 Repetitive Analyses: Cutting Your Workload

Background
Suppose that you need to do an analysis of 200 stock prices, or 25 countries, or some similarly gruesome task. If you're like us, you want to get it accomplished with as little work as possible. The brute force method—cloning code for each series or country and hand editing it—is a time-consuming, and worse, very inflexible way to handle this.

Fortunately, RATS provides a kit of tools for dealing with these situations. The more structured the analysis, the easier it is to write a short program.

Organizing Your Data
The first step is to store your data in a RATS format file or in a spreadsheet. RATS format is better, because you are likely to need to dip into the file many times. The reason for keeping the data in one of these forms is simple: with those formats, DATA looks on the file for the series with a label matching a requested name. We will be changing the name (label) of a series or set of series with each pass, and re-executing DATA, so we can keep refilling the same memory space with new data.

Numbered Series and Arrays of Series
The other key step is to use numbered data series or arrays of series. (See Section 4.4 for more details). For numbered series, you need to set aside a block of series either with ALLOCATE (using the series parameter), with SCRATCH, or with DECLARE VECT[SERIES]. For instance,

allocate 5 2007:2

creates series 1 to 5. We can reference them as 1, 2, 3, 4 and 5 or with variables such as DO loop indexes which take those values. Note that to use these series we must refer to them explicitly with numbers, integer variables, or names assigned to them with the instruction EQV.

However, VECTOR of SERIES are generally easier to work with and more foolproof than series numbers. For example, instead of the above, do:

allocate 2007:2
dec vector[series] vs(5)

You can refer to specific series using the subscript on VS (and you can use a loop index for that subscript to easily loop over the set of series). You can also use the name of the vector by itself in place of a list of series when you want an operation (such as PRINT or COPY) to apply to all series in the VECTOR. For example:

print / vs(1) \hspace{1cm} Prints the series stored in element 1 of VS
print / vs \hspace{1cm} Prints all 5 series stored in VS
Using LABEL Variables and the %S Function

LABEL variables and VECTORS of LABELS play an important role in these examples. The instruction LABELS resets the label of a series, which, as mentioned above, is what the DATA instruction uses to locate data on a RATS format or spreadsheet file.

“Labels” are strings of up to sixteen characters. You can concatenate them using either the explicit function %CONCAT(a, b), or simply by using a+b. The a+b notation also works if b is an integer or integer-valued expression. The function %S(label) can be used to refer to (or create) a series with the given label. Examples of the use of labels:

```plaintext
declare label lab lab1 lab2
compute lab="abcde"
compute lab1=lab+lab  
     lab1="abcdeabcde"
compute lab2=lab+3  
     lab2="abcde3"
set %s(lab2) = %s(lab)**3  
     Series ABCDE3 = ABCDE**3
```

Examples

VECTORS of LABELS are particularly useful for reading in or storing lists of variable or file names. For example, suppose you have a text file containing a list of (an unknown number of) series names, and you want to read those series in from a data file. You could do something like:

```plaintext
declare vector[label] names
open data namelist.txt
read(varying) names
compute num = %rows(names)
declare vector[series] vars(num)
labels vars
# names
open data data.xls
data(format=xls, org=cols) / vars
```

This pulls in all the variable names from NAMELIST.TXT, determines how many names were read in, creates a VECOR of SERIES of that size, and assigns the labels to the series in the vectors. It then reads in those series from a spreadsheet (RATS looks for series on the file whose names match the labels of the VARS series).

This next example computes beta values for a group of stocks. First, consider computing a beta value for a single security, which requires regressing the return for the security on the market return. Let’s let MARKET be the market return and series 1 be the return on the single security. We just regress series 1 on MARKET and report the result:

```plaintext
linreg(noprint) 1
     # constant market
display(unit=copy) %l(1) @20 ##.##### %beta(2)
```
Chapter 16: Advanced Programming Tools

Note that %L(1) is the label of series 1, %BETA(2) is the second coefficient. We use NOPRINT on LINREG and instead list the important information to the COPY unit.

Now all we need to do is to write a loop around this which resets the labels with each pass. Here we will handle the security labels by reading them from a file. The file consists of the symbols separated by blanks.

cal(m) 1986
allocate 1 1996:12
open data ticker.lst
declare vector[label] tickers
read(varying) tickers
compute ntickers=%rows(tickers)
open data returns.rat
open copy betas.lst
data(format=rats) / market
do i=1,ntickers
  labels 1
  # tickers(i)
clear 1
data(format=rats) / 1
linreg(noprint) 1
  # constant market
display(unit=copy) tickers(i) @20 ##.##### %beta(2)
end do i

This next program does an overlay graph of unemployment and inflation for seven countries, using a VECTOR of SERIES. The labels here are generated by appending SUR (Standardized Unemployment Rate) and DEFLS suffixes to three letter country prefixes. The INQUIRE instruction determines the maximum range over which both inflation and unemployment are defined.

open data oecddata.rat
cal(q) 1960
all 1998:4
declare label country
declare vector[series] vars(2)
dofor country = "can" "deu" "fra" "gbr" "ita" "jpn" "usa"
  labels vars(1) vars(2)
  # country+"sur" country+"defls"
clear vars
data(format=rats) / vars
set inflation = log(vars(2){0}/vars(2){1})
inquire(regressorlist) n1 n2
  # vars(1) inflation
graph(header="Inflation + Unemployment for "+country,$
  overlay=line) 2
  # vars(1) n1 n2
  # inflation n1 n2
end dofor
16.4 Interactive Procedures

RATS provides a variety of tools for writing highly interactive programs and procedures. You can construct programs that allow the end-user to control the program using pull-down menus, view messages and progress bars via pop-up dialog boxes, select series from drop-down lists, and more.

Tools

RATS offers several ways to control an interactive procedure: a set of steps controlled by a user through dialogs and prompts. These range from using simple prompts and informational dialogs to simplify routine or oft-repeated tasks, to implementing complex menu-driven procedures like the CATS cointegration analysis package.

- The **SOURCE** instruction can be extremely helpful when you are designing procedures where the user picks one of a set of (RATS) programs to run. Put the programs on separate files and use separate **SOURCE** instructions for each.
- **MESSAGEBOX** and **INFOBOX** display information in pop-up dialog boxes. **MESSAGEBOX** is used for messages which require a user-response (such as Yes, No, or Cancel), while **INFOBOX** is used to display informational messages (which can be updated) and graphical “progress” bars.
- **USERMENU** defines a pull-down menu, which can be used to control the program. The CATS cointegration analysis procedure (available separately from Estima) is the most sophisticated implementation of **USERMENU**.
- **SELECT** displays a dialog box that allows the user to select from a list of series, strings, or integers.
- **QUERY** obtains information (as a line of text) from the user. The input text is then interpreted.
- **MEDIT** displays the contents of an array in a spreadsheet style window, which makes it easy to view, enter, or edit the values in the array.
- **MENU** and **CHOICE** display a dialog prompting the user to choose one action from a single list of choices.
- **DBOX** allows you to create custom user-defined dialog boxes. It provides all the capabilities of **MENU/CHOICE, SELECT, QUERY** and a lot more.

Using MESSAGEBOX

**MESSAGEBOX** displays one of a limited set of standard dialog boxes to give information to the user or request answers to simple “Yes/No” type questions.

```
messagebox(style=alert) "Matrix is Singular"
messagebox(style=yesno,status=ok) "Save to File?"
```

The first of these is an “alert,” which simply displays the message and waits for the user to dismiss the dialog before continuing. The second includes “Yes” and “No” buttons. The variable **OK** will be 1 if the user clicked “Yes” and 0 if “No.”
Using USERMENU and SELECT

The example below uses the USERMENU and SELECT instructions to create a menu-driven program for specifying and estimating a least squares regression model. The USERMENU instruction creates the menu and controls program flow, while the SELECT command allows the user to select dependent and independent variables from all series currently available.

Before the user can run a regression, she must select a dependent variable and a set of regressors. Thus, we disable the “Run Regression” item until those two steps have been done. We use check marks to indicate which of the first steps the user has done. A version of this program is included with RATS, in the file OLSMENU.PRG.

Note, by the way, that we started with a much more modest version of this (without the check marks and item disabling and enabling), working up to the final version.

The first few lines of this program are specific to the data set being used. They define the frequency, starting and ending date, read in the data, and do the necessary transformations. To use the OLS menu program with your own data, simply modify or add to these lines as needed for your data set

calendar(a) 1922
allocate 1941:1
open data fooddata.dat
data(format=free,org=obs) /
   foodcons prretail dispinc foodprod prfarm
set trend = t
set avgprices = (prfarm+prfarm{1})/2.0
set logconsump = log(foodcons)
set logincome = log(dispinc)
set logprice = log(prretail)

The following code is the main part of the OLSMENU program. After executing these lines, a menu called “OLS” is added to the menu bar. The user must select a dependent variable and a list of independent variables before they can choose the “Run Regression” operation, which does the OLS regression. The user can change the variables and re-run the regression as desired. Select “Done” to remove the menu from the menu bar and return control to “ready” mode.

compute depvar=0,regressors=0
compute dep_item = 1, ind_item = 2, run_item = 3, done_item = 4
*
usermenu(action=define,title="OLS") $
  1>>"Select Dependent Variable" $
  2>>"Select Regressors" $
  3>>"Run Regression" $
  4>>"Done"
Disable the ‘Run Regression’ item:

```
usermenu(action=modify,enable=no) run_item
```

Start a Loop and activate the menu

```
loop
usermenu
if %menuchoice==1 {
    With one parameter, SELECT allows only one choice, returning it here in the variable DEPVAR.
    select(series,status=ok) depvar
    If user made a selection, check the DEP_ITEM:
    if ok {
        usermenu(action=modify,check=yes) dep_item
        If we have both a dependent variable and regressors enable the ‘Run’ item:
        if depvar>0.and.regressors>0
            usermenu(action=modify,enable=yes) run_item
    }
}
if %menuchoice==2 {
    With two parameters, SELECT allows multiple selections, returning the number of choices in REGRESSORS and the list of choices in REGLIST.
    select(series,status=ok) regressors reglist
    if ok {
        usermenu(action=modify,check=yes) ind_item
        if depvar>0.and.regressors>0
            usermenu(action=modify,enable=yes) run_item
    }
}
if %menuchoice==3 {
    linreg depvar
    # constant reglist
}
if %menuchoice==4
    break
end loop
usermenu(action=remove)
```

This will allow the user to run regressions with various combinations of regressors and dependent variables until they select the “Done” item from the OLS menu.
Using QUERY and MEDIT

QUERY and MEDIT are useful for obtaining numerical or character information from a user. QUERY is used to get a limited amount of information, usually a single number or string. MEDIT gets an entire array. MEDIT is also useful for displaying a set of results in matrix form. See Section 16.5.

QUERY pops up a dialog box with a prompt for the user. This is a “modal” dialog, so the program will wait until the user is done with it before continuing. QUERY can be set to accept real values, integers or strings. You need to make sure that any variable that you want filled in has the type that you want. If you need an integer, DECLARE it as an INTEGER. With the VERIFY option, you can check that the user has entered valid information, and prevent him from exiting the dialog until the input is corrected, or he cancels. You should always include a STATUS variable, and check it to see if the user has cancelled.

MEDIT pops up a “spreadsheet” with cells for a (real-valued) matrix. If you are using it for input, you should use the options EDIT and MODAL to ensure that the user fills in the matrix before the program continues. The matrix should be DECLARE’ed and DIMENSION’ed first. You can’t request an array with variable dimensions—they must be known in advance. It’s usually a good idea to zero out the array first—in most situations, most of the cells will be zero anyway.

The following takes the most recent regression and requests information to do a set of linear restrictions on it. (Similar to what the Regression Tests wizard does). The QUERY asks for the number of restrictions. The user then fills in the restrictions in rows of a matrix using MEDIT. The HLABELS option on the MEDIT is set up to label each column with the regressor label. The final column is labeled with =.

```plaintext
procedure TestRestrict
local integer nrestr i
local vect[strings] hlabels rlabels
local rect r capr
local vect lowr
local integer i

* Ask for the number of restrictions, make sure the
* value is positive and no bigger than %NREG.
*
query(prompt="How Many Restrictions?",status=ok,$
    verify=nrestr>0.and.nrestr<= %nreg,errmsg="Illegal Value") $
    nrestr
if .not.ok
    return

dim r(nrestr,%nreg+1)
compute r=%const(0.0)
dim hlabels(%nreg+1)
```
compute rlabels=%eqnregrlabels(0)
do i=1,%nreg+1
   if i<=%nreg
      compute hlabels(i)=rlabels(i)
   else
      compute hlabels(i)="="
   end
end do i
medit(modal,edit,hlabels=hlabels) r
compute capr=%xsubmat(r,1,nrestr,1,%nreg)
compute lowr=%xcol(r,%nreg+1)
mrestrict nrestr capr lowr
end test

Using MENU and CHOICE

MENU was originally designed to handle program flow under user control, but USER-
MENU now does that in a more flexible way. However, MENU still has its uses. It
displays a dialog box with radio buttons for selecting one and only one of the choices
provided by the CHOICE blocks. The string on the MENU instruction is the overall
prompt for the choices. Each CHOICE instruction is followed by the instruction or
block of instructions to be executed if that is chosen. If there is more than one in-
struction controlled by a CHOICE, enclose the group within { }.

Combining MENU with SOURCE is an easy way of controlling a set of steps in a proce-
dure. For example:

loop
   menu "What now?"
   choice "Identification"
      source identify.src
   choice "Estimation"
      source estimate.src
   choice "Diagnostics"
      source diagnost.src
   choice "Forecast"
      source forecast.src
   choice "end"
      halt
   end menu
end loop

will keep executing the menu until END is chosen, with each other choice running the
instructions on a source file.
Using DBOX

The **QUERY** command discussed earlier is useful for getting a single piece of user input via a simple dialog box. The **DBOX** command is a much more powerful instruction for creating custom dialog boxes. It can generate dialogs with multiple fields, and supports many different types of dialog box elements, including simple text labels (used to identify fields or present information), text fields to input numbers or strings, check boxes to turn settings on or off, radio buttons to select from a list of choices, and several kinds of “list boxes” allowing the user to select one or many items from lists of series or other information.

To create a dialog box, you need at least three **DBOX** instructions. The first command uses the option **ACTION=DEFINE** to begin the definition of the dialog box:

```
dbox(action=define)
```

This is followed by one or more additional **DBOX** commands with the (default) **ACTION=MODIFY** option to add the desired fields to the dialog. For example, this adds a checkbox field titled “Graph?”. The returned value (1 for on, 0 for off) will be stored in **GRAPHYESNO**:

```
dbox(action=modify, checkbox="Graph?") graphyesno
```

Finally, use the **ACTION=RUN** option to display the dialog box:

```
dbox(action=run)
```

Below is a simple example, taken from the **RegHeteroTests** procedures, available on the file **RegHeteroTests.SRC**:

```
procedure RegHeteroTests
  local integer lm exlm
  local vect[integer] selreg
  local series usq logusq
  local rect[integer] fulltable seltable
  local integer nselect i j

  Initiate the dialog box, and supply a title:
  dbox(action=define,title="Regression-Based Tests")

  Add a static text field to the dialog (note that we omit the ACTION option because ACTION=MODIFY is the default).
  dbox(stext="Which Tests Do You Want?")

  Add two check boxes, one for each of the two available tests. The user can turn on either or both boxes. The settings of the check boxes (1 for on, 0 for off) will be stored in LM and EXLM, respectively.
  dbox(checkbox="u**2 form") lm
  dbox(checkbox="log u**2 form") exlm
```

Now add a title for the list of regressors field. The SPECIAL option puts an extra space between the previous field and this new text field.

dbox(stext="Select Regressors",special=gap)

Display a scrolling list of the regressors from the most recent regression. The user can select one or more of these.

dbox(scroll,regressors) selreg

Activate the dialog box. If the user selects “Cancel” in the dialog box, the status check routine will exit from the procedure.

dbox(action=run,status=ok)
if .not.ok
  return

The remaining commands use the information collected from the dialog box to perform the selected tests.

* How many regressors were selected?
compute nselect=%rows(selreg)

* Pull them out from the regressor table
compute fulltable=%eqntable(0)
dim seltable(2,nselect)
ewise seltable(i,j)=fulltable(i,selreg(j))

* Create the transformed residuals series
set usq = %resids**2
set logusq = log(usq)
if lm {
  linreg(noprint) usq
  # %rlfromtable(seltable)
  cdf(title="LM Test on U**2") chisqr %trsquared nselect-1
}
if exlm {
  linreg(noprint) logusq
  # %rlfromtable(seltable)
  cdf(title="LM Test on log(u**2)") chisqr %trsquared nselect-1
}
end RegHeteroTests

See the DBOX section of the Reference Manual for complete details on this instruction.
16.5 Handling Output

Using REPORT

REPORT is a very useful tool for building up tabled output. It’s covered in Section 4.8.

Using OPEN(WINDOW)

The OPEN instruction with the option WINDOW creates a new text window into which you can direct output. With the CHANGE instruction, you can direct all subsequent output to this window, or, with UNIT options on instructions like DISPLAY and COPY, you can put selected information into it. For example:

```
open(window) tempout "Second Window"
change output tempout
display "This will go to the tempout window"
change output screen
display "This will go to the main output window"
display(unit=tempout) "This will also go to tempout"
```

The OPEN instruction creates a window whose unit name is tempout. The window title (which is what appears on the title bar and in the Window menu) is “Second Window.” The first CHANGE OUTPUT switches output to that window; the second switches it back to the standard output window (whose unit name is always SCREEN).

Graph Window Titles

By default, RATS simply labels new graph windows by number, for instance, “Graph.01” and “Graph.02”. The WINDOW options on SPGRAPH, GRAPH, SCATTER, and GCONTOUR allow you to assign your own titles to the graph windows. These titles appear on the title bar of the window, and in the Window menu.

For procedures which produce several graphs, assigning meaningful window titles to each graph can be very helpful to the user, particularly for selecting a graph from the Window menu.

```
graph(window="Kalman filter/smooth") 3
# lgnp
# hpsmooth
# hpfilter
```

Spreadsheet Windows

Some instructions (PRINT and most of the forecasting instructions) include a WINDOW option, which allows you to direct their output to a spreadsheet style window, rather than a standard text window. This will give you a separate window on the screen for each of these operations. While these windows are “read-only,” the information in them can easily be transferred into spreadsheet programs or word processors by using File-Export... or using the Cut and Paste operations.
MEDIT

We looked at MEDIT (Matrix EDIT) in the last section as an instruction for getting information into RATS. It is also handy for displaying information from RATS. If you want to display the contents of a matrix, use MEDIT with the options NOEDIT and NOMODAL. With the NOMODAL option, MEDIT simply displays the information and continues. To help the user find the desired information, MEDIT has a WINDOW option for labelling the window on the title bar and in the Window menu. From the MEDIT window, the user can export the data, or cut and paste it into another program.

MEDIT is most useful when you need to display a single array. REPORT is more flexible when the data are generated across several instructions.

Using INFOBOX

INFOBOX allows you to keep the user informed about the progress of a time-consuming operation. While it can be used to post a simple message, its main purpose is to display a progress bar. Note, by the way, that INFOBOX behaves quite differently from MESSAGEBOX. An INFOBOX displays a window on the screen which requires no response from the user—it removes itself from the screen under the control of your program. A MESSAGEBOX, on the other hand, stops the execution of the program until the user answers the question it asks, or, in the case of an alert box, acknowledges having seen it by clicking OK.

The following displays a progress bar for an operation which takes NDRAWS draws.

infobox(action=define, progress, lower=1, upper=ndraws) $  
 "Gibbs Sampler"

do draw=1, ndraws
   ( instructions which actually do things)
   infobox(current=draw)
end do draw
infobox(action=remove)
Bibliography


Bibliography


Index

Symbols

$ line continuation, UG55, UG63.
%ABS function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%BETA vector, RM504.
%BOXCOX function, RM294, UG287.
%CMPLX function, UG558.
%CSQRT function, UG558.
%CONJG function, UG558.
%CONCAT function, UG154, UG596.
%COLS function, UG159.
%CMOM array, UG156.
%CLOSESTWEEKDAY function, UG168.
%CLOSESTDATE function, UG168.
%CLOG function, UG558.
%CHISQR function, UG220.
%CEXP function, UG558.
%CDSTAT variable, UG222.
%CABS function, UG558.
%BQFACTOR function, UG359.
%BOXCOX function, RM294, UG287.
%BETA vector, UG156, UG172.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%ARG function, UG558.
%AVG function, UG160.
%BESTREP function, RM113.
%AVG function, UG160.
%TODAY function, UG168.
%TRACE function, UG160.
%TRADEDAY function, RM22, UG168.
%TRSQ variable, UG224.
%TRQUARED variable, UG224.
%STATS vector, UG172.
%TEST function, UG220.
%UNIFORM function, UG160, UG478, UG483.
%UNITY function, UG159.
%UZWZU variable, RM276, UG280.
%VALUE function, UG154.
%VARLAGSUMS array, UG358.
%VEC function, UG159.
%VESTORECT function, UG159.
%VETTOSYMM function, UG159.
%X array, RM5.
%XCOL function, UG161.
%XDIAG function, UG161.
%XRW function, UG161.
%XSUBMAT function, UG161.
%XX matrix, UG156, UG243.
%YEAR function, UG168.
%Z matrix, RM71, RM195.
%ZERO function, UG159, UG160.
%ZTEST function, UG220.
%ZEROS function, UG159, UG160.
%Z matrix, RM71, RM195.
%YEAR function, UG168.
%ZFRACTILES function, UG481.
%WMATRIX matrix, UG279.
%WFRACTILES function, UG481.
%WEEKDAY function, UG168.
%XSUBVEC function, UG161.
%XSUBMAT function, UG161.
%XROW function, UG161.
%XDIAG function, UG161.
%XCOL function, UG161.
%XSUBVEC function, UG161.
%XSUBMAT function, UG161.
%XROW function, UG161.
%WEEKDAY function, UG168.
%WFRACTILES function, UG481.
%WMATRIX matrix, UG279.
%WFRACTILES function, UG481.
%WEEKDAY function, UG168.
%WMATRIX matrix, UG279.
%WFR value, UG220.

A

Abbreviating instruction names, UG60.
option names, UG61.
ABLAGS procedure, UG555.
Absolute value function, RM520.
complex numbers, RM523.
integers, RM535.
ACUMULATE instruction, RM2.
ADAPTIVE.PRG example, UG462.
Add factors, UG414.
Adjusted R-squared, UG177.
AIC. See Akaike criterion.
Akaike criterion, UG203, UG320.
BJEST procedure, UG332.
example, UG204.
for VARs, UG344, UG348.
Akaike, H., UG203.
ARELLANO.PRG example, UG555.
ARCH/GARCH Wizard, RM206.
Markov switching, UG458.
ARCH/GARCH Wizard, RM206.
Archiving data, RM364.
ARCHTEST procedure, UG227.
Arellano-Bond estimator, UG555.
ARELLANO.PRG example, UG555.
ARCH/GARCH Wizard, RM206.
Markov switching, UG458.
ARCH/GARCH Wizard, RM206.
Archiving data, RM364.
ARELLANO.PRG example, UG555.
ARCH/GARCH Wizard, RM206.
Markov switching, UG458.
ARELLANO.PRG example, UG555.
ARCH/GARCH Wizard, RM206.
Markov switching, UG458.
ARELLANO.PRG example, UG555.
ARCH/GARCH Wizard, RM206.
Markov switching, UG458.
ARELLANO.PRG example, UG555.
ARCH/GARCH Wizard, RM206.
Markov switching, UG458.
Complex numbers
    absolute value of, RM523.
    argument of, RM520.
    conjugate, RM525.
    creating, RM525.
    exponentiation, RM523.
    expressions using, UG558.
    imaginary part of, RM535.
    log function, RM524.
    matrix adjoint, RM527.
    matrix inverse, RM527.
    operators on, RM571.
    real part of, RM554.
    singular value decomposition, RM527.
    square root, RM526.

Complex series, UG559.
    %Z variable, RM71.
    converting from real, RM400.
    converting to real, RM73.
    creating, RM71, RM400.
    displaying, UG564.
    Fourier transforms, RM171.
    graphing, RM73, UG564.
    labeling, RM39.
    padding, UG563.
    preparation, UG562.
    printing, RM64.
    reading in, RM401.
    COMPUTE instruction, RM50, UG36, UG136, UG145.
    alternative to INPUT, RM252.
    ASSIGN variable, UG137.
    alternative to ASSIGN, RM252.
    END option, RM489.
    reading in, RM401.
    CONDITION instruction, RM73.
    CONDITION procedure, UG398, UG400.
    CONDITION.PRG example, UG400.
    CONDITIONAL option, UG12.
    CONDITIONAL.PRG example, UG400.
    CONSTRUCT option, UG12.
    CONSTRUCT.PRG example, UG400.
    CONDITION number, RM134.
    CONDITION.PRG example, UG400.

CONDITION instruction, RM50, UG36, UG136, UG145.
    alternative to INPUT, RM252.
    assigning data types, UG137.
    expressions in, RM569.

Concatenation
    of matrices, RM522, UG158.
    of strings, RM525, UG118, UG154.
    Condition number, RM134.
    CONDITION procedure, UG398, UG400.
    CONDITION statement, UG400.
    CONDITION instruction, RM50, UG36, UG136, UG145.

Confidence bands
    for impulse responses, UG365.
    CONJUGATE instruction, RM41.
    Consistent covariance matrices
        hypothesis tests with, RM297.
        with MCOV instruction, RM295.
        CONSTANT series, UG25.
        CONSTANT.PRG example, UG238.
        Constants, UG52.
    Constrained optimization
        non-linear, UG264, UG271.
        CONSUMER.PRG example, UG285.
        Continuation lines, UG63.
        with expressions, UG55.
        with FORTRAN format, UG95.
        Contour plots, RM212.

Convergence
    non-linear estimation, UG256.
    setting criterion with NLPAR, RM318.
    simultaneous equation models, UG412.
    COPY instruction, RM54, UG70, UG82.
        with binary format, UG103.
        with FORTRAN format, UG96.
        with free-format, UG93.
        with PORTABLE format, UG102.
        with spreadsheets, UG87.
    COPY unit, RM339.
    CORRELATE instruction, RM58.
    CORRELATE instruction, RM58.
    Correlation
        auto, RM58.
        of series, RM66.
        Cosecant, RM526.
        Cosine
            function, RM526.
            hyperbolic, RM526.
            inverse, RM520.
            taper, RM471.
        COTANGENT instruction, RM526.
    Count data model, RM92.
    Covariance
        of arrays, RM526.
        auto, RM58.
        correlations from, RM527.
        of series, RM66.
    Covariance matrix
        modelling, RM75.
        of coefficients, RM276, UG211.
        consistent estimate of, RM295.
        UG211, UG214, UG266.
        of series, RM496.
        VAR (via KFSET), RM263.
        Covariance Matrix wizard, RM496.
        Covariance.correlation matrix, UG179.
        Cox, D. R., RM294.
        CPRINT instruction, RM64.
        CREATE option
            with LINREG, UG212.
            with RESTRICT and MRESTRICT, UG206.
        Cross Correlations wizard, RM66.
        CROSS instruction, RM66.
        Cross sectional data, UG524.
        Cross.correlations, RM66.
        Cross-equation restrictions, RM457.
        Cross-moment matrix
            computing, RM44.
        Cross-periodograms, RM502.
        Cross-spectral analysis, RM352.
        CUBIC instruction, RM56.
        CVMODEL.PRG example, UG370.
        CVMODEL instruction, RM75, UG355, UG360.
        CVMODEL.PRG example, UG370.
        CXTREMUM instruction, RM26.
        RM79.

D

Daily data, RM28.
    removing gaps, RM402.

DAMP option, UG185.

Data. See also Series.
    archiving with PRTDATA, RM364.
    changing frequency of, RM32.
    editing, RM130.
    interpolating, RM32.
    opening files containing, UG10.
    reading from a file, UG10, UG70.
    setting frequency of, RM27.
    typing in, UG72.
    writing to a file, UG70.
    Data (RATS Format) wizard, UG81.
    DATA I/O unit, RM339.
    DATA instruction, RM80, UG10, UG39, UG70.
    binary data, UG103.
    changing data frequencies, UG76.
    CITIBASE format, UG99.
    DRI format, UG99.
    FORTRAN (text) format, UG96.
    free-format (text) files, UG92.
    HAVER format, UG99.
    with holidays omitted, UG80.
    PORTABLE format files, UG102.
    spreadsheet files, UG87.
    text files, UG93.
    troubleshooting, UG89.
    Data transformations, RM420, UG17.
    Data types, UG136, UG143.
    basic, UG146.
    declaring variables, RM99.
    Data wizards, RM80, UG12.
    Date/time stamp, RM527.
    Dates. See also Calendar functions.
        arithmetic using, UG137.
        of entries, RM528.
        entry number of, RM523.
        in expressions, UG137.
        functions, RM524.
        label function, RM528.
        specifying, RM29.
        on spreadsheet files, UG86, UG89.
    Davidson-MacKinnon test, UG241.
    Davis, R. A., UG319.
    Day of week, RM529, RM567.
    dBase files, UG73, UG90.
    DBOX instruction, RM83, UG145.
Index

DO instruction, RM124, UG56, UG581.
DO loops
EWISE as alternative to, RM163.
Doan, T., RM264, UG343, UG392, UG393.
DOFOR instruction, RM126, UG56, UG581.
Dot product, RM529.
DRI database, UG74, UG99.
Dummy observation prior, UG396.
Dummy variables, UG49.
Chow tests, UG237.
panel data, UG545.
seasonal, RM413.
for subsamples, RM427, UG181.
Duration analysis, UG540.
Durbin h statistic, UG230.
Durbin, J., RM26, UG230.
Durbin-Watson statistic, UG27, UG178, UG230.
%DURBIN variable, RM276.
with missing observations, UG182.
Dynamic forecasts, RM188, UG34, UG299.
Dynamic Linear Models. See DLM.

Easter
date of, RM529.
Echoing of input lines, RM140.
ECT instruction, UG374.
ECT.PRG example, UG376.
EDIT instruction, RM130, UG82.
DELTA method
with SUMMARIZE, RM455.
DeLurgio, S., UG319, UG337.
Density
beta distribution, RM540.
chi-squared, RM523.
non-central, RM524.
empirical, RM105.
GED distribution, RM541.
generalized Pareto, RM541.
GEV distribution, RM541.
negative binomial, RM541, RM546.
normal, RM528.
multivariate, RM540.
at random draw, RM552.
t distribution, RM563.
multivariate, RM541.
non-central, RM563.
DENSITY instruction, RM105.
UG460.
Dent, W., UG232.
Dependent variables
of equation, RM142, RM530.
of MODEL, RM544.
in a VAR, RM495.
DESEASON.PRG example, UG571.
Determinant, RM528, UG160.
DETERMINISTIC
instruction, RM108, UG342, UG387.
DFC option, UG217.
DFUNIT procedure, UG249, UG254.
Diagonal matrix, RM529.
Dialog boxes
INFOBOX, RM248.
MESSAGEBOX, RM304.
user-defined, UG598.
Dickey, D., UG249.
Dickey-Fuller unit root test, UG249.
Diebold-Mariano test, UG309.
DIEBP210.PRG example, RM488.
DIEBP228.PRG example, UG125.
DIFF files, UG85.
DIFFERENCE instruction, RM109.
Differencing, RM109, UG50.
matrix operator, RM185.
prior to forecasting, UG315.
prior to VAR models, UG343, UG387.
seasonal, RM109.
Differentiating wizard, RM109, UG19.
Digamma function, RM529.
DIMENSION instruction, RM112.
UG145, UG148.
Dimensioning arrays, RM112.
Discrete dependent variables, RM92,
UG525.
DDV instruction, RM92.
DISPLAY instruction, RM113, UG36,
UG140, UG145.
Displaying
arrays, RM505.
calculations, UG140.
complex series, UG564.
data series, RM356.
series, UG16.
DISTRIB procedure, RM32, UG77.
Distributed lags, UG201.
graphing, UG201.
lag length, UG203.
polynomial, RM135, UG207.
sums of coefficients, RM453.
Distributions
chi-squared, RM36.
non-central, RM36.
RM530
extreme value, RM93, RM360, UG525, UG526.
F, RM36.
GED (generalized error), RM206, UG419.
logistic, RM360.
logit, RM93.
normal, RM36.
prob, RM93.
t, RM36, UG419.
Wishart, UG492.
Division
modulo, RM543.
DLM, RM116, UG436.
forecasting with, UG314.
stationary covariance matrix, RM549.
DLM instruction, RM116.
DMARIANO procedure, UG309.

RATS User's Guide 619
convergence criterion, RM318.
discrete dependent
variables, RM92.
distributed lags, UG201, UG207.
fixed effects, RM347, UG549, UG552.

GARCH, RM206.
generalized method of
moments, RM295, RM320,
UG282.
HAC, RM295.
Hannan efficient, UG572.
Heckit, UG537.
heteroscedasticity
consistent, RM295.
Hildreth-Lu, UG189.
iterated least squares, UG537.
Kalman filter, RM259.
least absolute deviations
(LAD), RM396, UG214.
limited dependent variable, RM268.
linear regression, RM272, UG176.
with linear restrictions, RM135,
RM307, RM385, UG223.
logit, RM92, UG525.
maximum likelihood, RM290,
UG286.
minimum absolute deviations
(MAD), RM396.
mixed, UG208.
multinomial logit, UG528.
multivariate non-linear least
squares, RM320.
nerval network, RM328, UG469.
non-linear, UG45.
non-linear least squares, RM313,
UG276, UG278.
non-linear systems, UG282.
non-parametric, RM336.
orinary least squares
(OLS), UG25, UG43, UG183.
panel data, RM353, UG548.
parameters for non-linear, RM334.
probit, RM92, UG525.
quantile regression, RM396.
random effects, UG549, UG552.
recursive least squares, RM392.
ridge, RM44, UG208.
robust, UG214, UG215.
seemingly unrelated regressions
(SUR), RM457, UG197, UG200.
simultaneous equation
models, UG405.
stepwise regression, RM449.
systems of equations, RM457.
three-stage least squares, RM457,
UG197, UG405.
three-stage non-linear least
squares, RM320, UG282, UG405.
tobit, UG533.
truncated samples, UG533.
two-stage least squares, RM257,
UG313.

Estrella, A., RM97.

F
F distribution
inverse tail probability, RM337.
tail probability, RM534.
F-tests, RM366, UG27, UG31, UG142,
UG118.

CDF instruction, RM36.
for zero coefficients, RM164.
Factor analysis, RM75, UG357.
FACTOR option, UG362.
Factorial function, RM532.
Fair, R. C., RM6, UG377, UG404.
Fair's procedure for AR1, RM6,
UG404.
FAME format, UG74.
Fan chart, UG134.
FE estimator. See Fixed effects.
FFT instruction, RM171, UG561, UG566.

FGLS estimator. See Generalized least squares.

Filename of I/O Unit, RM565.

Files appending data to, RM339.
  binary, UG103.
  closing, RM42.
  closing RATS format, RM370.
  dBase, UG73.
  FAME format, UG74.
  formats, UG11, UG39.
  FORTRAN formats, RM56.
  free-format text, RM56, UG92.
  graphics, UG110.
  list of available, RM422.
  opening, RM339, UG10.
  operations on, RM592.
  organization of data on, UG11.
  outputting scalars and arrays to, RM113.
  PORTABLE format, UG102.
  PRN format, UG85.
  reading data from, RM80, RM374.
  reading strings from, UG118.
  rewinding, RM391.
  spreadsheet, UG73, UG85.
  temporary, RM42.
  text (ASCII), UG74, UG95.
  translating data formats, UG71.
  WKS, UG85.
  writing series to, RM54, RM356.

FILTER instruction, RM173.


Filtering, RM173.
  frequency domain, UG569.
  Henderson, RM173.
  Hodrick-Prescott, RM173, UG446.
  Spencer, RM173.

FIND instruction, RM179, UG293.

Fitted values, RM358, UG34, UG43.

FIX function, UG137.

Fixed effects estimators, RM347.
  UG549, UG552.
  with PREGRESS, RM353.

Flannery, B. P., RM60.

Fluctuations test distribution, RM546.

FMATRIX instruction, RM185.
  UG157, UG210.

FOLD Instruction, RM187.

Fonts
  for graphs, RM225, UG119.
  FORCEDFACTOR procedure, UG361.
  FORECAST instruction, RM188.
  UG34, UG300, UG320, UG482.
  saving results, UG303.
  Forecasts, UG47, UG298.
  ARIMA models, RM17, UG314.
  UG319.
  example, UG320.

Box-Jenkins models, UG314, UG319.
  building models, RM224.
  conditional, RM188, UG397.
  UG400.
  dynamic, RM188, UG34, UG299.
  error statistics, RM443.
  exogenous variables in, RM188.
  exponential smoothing, RM153.
  UG314, UG317.
  graphing, UG34, UG132, UG304.
  identities, UG302.
  instructions for, RM589.
  of MODELS, UG409.
  out-of-sample, UG412.
  performance statistics, RM476.
  UG307, UG308.
  rolling regressions, UG305.
  with shocks, RM188.
  spectral methods, UG314, UG334.
  UG574.
  standard errors of, RM147, UG307.
  static, RM358, UG34, UG299.
  time series methods, UG34, UG299.
  UG314.
  troubleshooting problems, UG310.
  univariate, RM486.
  with VARS, UG377, UG385.
  FORM=CHISQUARED, UG211.
  UG213.

FORM=CHISQUARED, UG211, UG213.

FORM=CHISQUARED, UG211, UG213.

Formatting output
  %BESTREP function, RM521.
  %MAXIMAL function, RM543.

Formulas
  calendar/date, RM32.
  created, RM483.
  calendar, RM32.
  complex/spectral, UG558.
  list of, RM511.
  random draws, UG483.
  using, RM511, UG36, UG53, UG138.
  Functions wizard, UG138.

Gain
  cross spectral, RM483.
  Gamma distribution
  random draws, RM552, UG483.
  Gamma function, RM534.
  digamma, RM529.
  incomplete, RM534.
  log of, RM539.
  trigamma, RM564.

GARCH instruction, RM206.

GARCH models, RM206, RM290, UG417.
  Bayesian analysis, UG502, UG512.
  bootstrapping, UG518.
  example, UG427.
  multivariate, UG428, UG434.

GARCHBOOT.PRG example, UG518.
  GARCHGIBBS.PRG example, UG512.
INDEX

GARCHIMPORT.PRG example, UG502.
GARCHMV.PRG example, UG434.
GARCHUV.PRG example, UG427.
Gardner, E. S., RM156.
Gauss-Newton algorithm, UG260.
used by NLLS, RM313.
Gauss-Seidel algorithm, UG410, UG411.
used by INITIAL, RM250.
Gaussian distribution. See Normal distribution.
Graph wizard, RM214, UG23.
GRAPH instruction, RM212.
GRAPH.PRG example, UG128.
GED distribution
CDF, RM534.
inverse CDF, RM537.
log density, RM541.
Gelfand, A. E., UG505.
Generalized instrumental variables. See Generalized Method of Moments.
Generalized inverse, RM534, UG159.
Generalized least squares for heteroscedasticity, UG186.
joint (Zellner), UG197.
panel data transformations, UG548.
Generalized Method of Moments, UG548.
for serial correlation, UG189.
Generalized residual, UG535.
Genetic algorithm, RM318, UG263.
GEV distribution
CDF, RM534.
density, RM541.
inverse CDF, RM537.
Geweke, J., UG232, UG576.
Gibbs sampling, UG232, UG507.
GIBBS.PRG example, UG507.
GIV.PRG example, UG281.
Global Insight database, UG74.
Glosten, L., UG420.
Godfrey, L. G., UG230.
Godfrey-Breusch test, UG230.
Goldfeld-Quandt test, UG228.
Goodness of fit, UG26.
Granger, C. W. J., UG232, UG507.
UG319, UG576.
Granger-Sims causality tests, UG232, UG346.
GRAPH instruction, RM214, UG20, UG108.
with complex series, UG564.
with SPGRAPH, RM434.
Graph wizard, RM214, UG23.
Graphs, RM214, UG20.
adding text to, RM228.
appearance options, RM225.
autocorrelations, UG129.
contour plots, RM212.
copying and pasting, UG113.
displaying, RM140, UG109.
exporting, UG111.
fan charts, UG134.
fonts on, UG119.
high-low-close, UG130.
highlighting entries, UG131.
impulse response, UG365.
instructions for, UG108.
keys, UG118.
labeling, RM225, RM228, UG117.
multiple per page, RM434, UG123.
opening, UG110.
orientation of, RM225, UG114.
overlay/two-scale, RM220, UG121.
preparing for publication, UG113.
printing, RM140, UG112.
saving, RM140, RM339, UG205.
scatter (X-Y) plots, RM405, UG21.
shading, UG131.
size of, RM225.
special, RM434.
Griddy Gibbs, UG515.
Griffiths, W. E., UG209.
GARCHUV.PRG example, UG427.
GARCHMV.PRG example, UG434.
GARCHIMPORT.PRG example, UG502.
Hausman specification test, UG243, UG247.
HAUSMAN.PRG example, UG247.
Haver Analytics database, UG73.
Haver Analytics data, UG99.
Hazard models, UG540.
Heckit models, UG533, UG537.
Heckman, J., UG211, UG537.
Henderson filter, RM173.
Hessian matrix, UG258, UG261, UG266.
HETERO.PRG example, UG188.
Heteroscedasticity, UG186.
consistent covariance matrix, RM295.
testing for, UG226, UG227, UG229.
HETTEST.PRG example, UG229.
Hildreth-Lu, UG6, UG189, UG404.
Hill, R. C., UG209.
Hill-climbing methods, UG259.
Histograms, RM105.
Historical decompositions, RM233, UG372.
Historical simulations, UG413.
HISTORY instruction, RM233, UG346, UG372.
HISTORY.PRG example, UG373.
Hodrick, R., UG446.
Hodrick-Prescott filter, RM173, UG446.
Hoerl, A. E., UG208.
Holidays, RM31.
on data file, UG80.
Easter, RM529.
finding date of, RM524, RM533.
Holt-Winters model, RM154.
Hosking, J. R. M., UG576.
HPFILTER.PRG example, UG446.
Hsiao, C., UG548.
Hyperbolic trig functions
cosine, RM526.
sine, RM558.
tangent, RM563.
Hypothesis tests, RM307, UG31.
applicability, UG222.
with BOXJENK, RM22.
coefficient values, RM473.
with consistent covariance matrix, RM297.
exclusion restrictions, RM164.
P-test, UG27.
formulas for, UG223.
instructions for, RM590, UG220.
linear restrictions, RM385.
for logit and probit, RM97.
with NLLS, UG277.
with SUR, UG199.
variables defined, UG222.
with VARs, RM372, UG347.
wizard for, UG33.

622 RATS User’s Guide
Index

I/O units, RM339.
  closing, RM42.
  filename functions, RM565, RM566.
  reserved, RM339.
  standard, RM37.
  switching, RM37.
  tips on, UG104.
  user-defined, RM339.
Identities, RM224.
  adding to a model, UG410.
  Identity matrix, RM535.
  IF instruction, RM237.
  %IF function as alternative to, RM237.
  IFT instruction, RM171, UG561.
  Imaginary part, RM525, RM535.
  Importance sampling, UG500.
  Importing data, UG71.
  IMPULSE instruction, RM240, UG346, UG364.
  alternative to, RM148.
  Impulse responses, RM240, UG364, UG368.
  confidence bands for, UG365.
  cumulative, RM2.
  fractiles, UG497.
  graphing, UG365.
  graphical, UG365.
  INPUT I/O unit, RM339.
  INPUT instruction, RM251, UG145.
  with complex series, RM401.
  Individual effects, RM366, UG546.
  means, RM346.
  Inequality constraints
    linear/quadratic programming, RM283.
    non-linear estimation, RM334, UG264.
  INFOBOX instruction, RM248, UG598.
    example using, UG478.
  Information criteria, UG203. See also Akaike criterion.
  INITIAL instruction, RM250.
  Innovations process, UG351.
  Input
    changing source of, RM37.
    echoing of, RM140.
    SOURCE instruction, RM429.
  INPUT I/O unit, RM339.
  INPUT instruction, RM251, UG145.
    with complex series, RM401.
  Inputting
    arrays, RM251.
    variables, RM251.
  INQUIRE instruction, RM254, UG591.
  Instructions
    changing default options, RM103.
    continuing on another line, UG55.
    echoing, RM140.
    executing, UG2.
    input files of, RM339.
    listed by general function, RM589.
    locations in compiled code, UG580.
    multiple per line, UG63.
    secondary input files, RM429.
    splitting up long, UG63.
    syntax of, UG60.
  INSTRUMENT.PRG example, RM194.
  Instrumental variables, RM257.
    RM464, UG192.
    generalized, UG281.
    with LINREG, RM272.
    non-linear least squares, UG277, RM313.
    simultaneous equation models, UG404.
  INSTRUMENTS instruction, RM257, UG192, UG404.
  Integer
    absolute value, RM534.
    from real, RM533.
    to real, RM533.
    to string, RM559.
    values in expressions, UG137.
  INTEGER data type, UG137, UG143, UG146.
  Integral
    Simpson's Rule, RM538.
    trapezoidal rule, RM538.
  Interactive mode, UG2.
  Interactive procedures, UG598.
    dialog boxes, RM248, RM304, UG598.
    input from user with QUERY, RM368.
    menus, RM303, UG598.
    menus with USERMENU, RM491.
    selecting from a list, RM418.
    using PROCEDURE, RM362.
    viewing/editing arrays, RM300.
  INTERPOL procedure, RM32, UG77.
  Interpolating data, RM32, UG77.
  Interval estimation, UG535.
  INTERVEN.PRG example, UG337.
  Intervention models, RM17, UG336.
  Intraday data, RM29.
  INV function, UG159.
  Inverse
    autocovariances, RM31.
    generalized, RM534, UG159.
    matrix, RM536, UG159.
    iterative least squares, UG537.
  Iterations
    non-linear estimation, UG256, UG262.

J
  Jacquier, E., UG489.
  Jarque, C. M., RM439.
  Jarque-Bera Normality test, RM439.
  Jenkins, G. M., RM250, UG319.
  Johansen cointegration test, UG253.
  JOHMLE procedure, UG253, UG254.
  JOHNP121.PRG example, RM394.
  Joyeux, R., UG576.
  JROBUST option, UG280.
  Judge, G. G., UG209.
  Julian date, RM538.

K
  Kalman filter, RM259.
  equation variances, RM265.
  estimating parameters, RM179.
  initial values, RM158, UG389.
  likelihood function, RM264.
  recursive residuals, UG390.
  sequential estimation, UG389.
  setting up, RM263, RM466.
  state-space models, RM116, UG439.
  time-varying coefficients, RM484, UG391, UG394.
  updating formula, UG388.
  VAR models, UG388.
  KALMAN instruction, RM259.
  Kalman smoothing, RM116, UG439, UG446.
  Kendall, M. G., RM441.
  Kennard, R. W., UG208.
  Kernel methods
    computing density, RM105.
    density estimation, UG460.
    non-parametric regression, RM336, UG460.
  KEYBOARD I/O unit, RM339.
  KFSET instruction, RM263, UG391.
  Kim, S., UG489.
  KLEIN.PRG example, UG192.
  Koopman, S. J., UG441.
  Koopmans, L. H., UG185, UG574.
  Kroner product, RM538, RM539, UG161, UG164.
  Kroner, K. F., UG428.
  Kurtosis, RM439, UG13.

L
  LABEL data type, UG143, UG146.
  Labels, RM539.
  of complex series, RM59.
  creating series from, RM557.
  on data files, UG86.
  expressions using, RM52, RM571.
  for graphs, UG117.
  series, RM266.
  of variables, RM539.
  LABELS instruction, RM266.
  vs. EQV instruction, RM146.
  LAD estimator, UG214.
  Lag length
    methods for choosing, UG203.
    VAR
      choice of, UG379.
      testing, UG348.
  Lagrange multiplier
    %LAGRANGE variable, RM326.
    constrained optimization, RM335, UG264.
Index

tests, UG224, UG226, UG230, UG278.
OPG (outer product of gradient), RM299.
Lags
(\) notation in expressions, UG17.
in regressor lists, UG42.
non-consecutive, with BOXJENK, RM18.
with panel data, UG545.
LAGS instruction, RM287, UG342.
LAGS option for serial correlation, UG189.
Lancaster, T., UG540.
Landscape mode, RM225, UG114.
LDV. See Limited dependent variables.
LDV instruction, RM268, UG533, UG534, UG539.
Leamer, E., UG554.
Least Absolute Deviations. See LAD estimator.
Least squares iterated, UG537.
with LINREG, RM272.
non-linear, RM313.
recursive, RM392.
three-stage, RM457, UG405.
two-stage, UG230.
Lo, A., RM201, UG416.
LOCAL instruction, RM278, UG363.
LOCALDLM procedure, UG446.
Local optimia, UG257.
local variables, UG583, UG588.
LOG instruction, RM281.
Log likelihood %LOGL reserved variable, RM276.
Logical expressions, RM538.
with IF function, RM535.
with IF instruction, RM237.
Logical operators, RM569, UG53.
creating dummies using, UG49.
with complex values, UG558.
Logistic distribution CDF, RM539.
Logistic function, RM541.
Logit models, RM92, UG252, UG531, UG532.
fitted values for, RM358.
samples with repetition, UG530.
Long memory models estimation, UG449.
ARCH, UG458.
MARKOV.SRC procedure, UG449.
Marquardt algorithm, RM319.
Matrix \[ (\ldots) \] notation, UG157.
absolute value of, RM520.
average value of, RM521.
banded, RM185.
block diagonal, RM522.
calculations, RM50.
Choleski factorization, RM185.
creating dummies using, UG49.
with complex values, UG558.

RATS User’s Guide
Index

RATS User's Guide

625

inverse, RM536, UG159.
generalized, UG159.
Kronecker product, RM538, RM539, UG161.
using in lists, UG149.
maximum value of, RM542.
minimum value of, RM543.
number of columns, RM525.
number of rows, RM556.
one-dimensional, UG148.
optimizing computations, UG164.
outer product, RM546.
partitioning, RM522.
printing, RM505.
QR decomposition, RM551.
quadratic form, RM545.
QZ decomposition, RM371.
ranks of elements, RM552.
RATS instructions for, RM592.
reading data into, RM251, RM374.
reading/writing complex series as, RM401.
rectangular, UG143.
RECTANGULAR data type, UG148.
reshaping, RM566.
scalar, RM546.
Schur decomposition, RM371.
of series, RM550, UG151.
setting, RM287, UG145.
sorting, RM536, RM558, UG162.
sparse, RM542.
submatrix setting, RM549.
subvector setting, RM550.
summing, RM560.
SV decomposition of, RM553.
sweep function, RM560, RM561.
USERMENU, UG598.
vector, UG143.
VECTOR data type, UG148.
viewing/editing with MEDIT, RM300, UG145, UG598, UG601, UG606.
Mean
individual (panel data), UG546.
moving window, RM309.
of a series, RM439, RM468.
MEANGROUP.SRC procedure, UG554.
Median. See also Fractiles.
moving window, RM309.
of a series, RM439.
MEMORY available, RM422.
clearing, RM137.
freeing, RM377.
MENU instruction, RM303, UG602.
Menus (user-defined), RM303, UG599.
pull-down, RM491.
USERMENU, UG598.
Moving average, UG18.
adding lags to equation, RM494.
alternative to, RM309.
representation, RM240, UG351.
serial correlation correction, RM295.
Moving Window Statistics wizard, RM309.
MONTHLY instruction, RM307, UG206, UG220.
MSVARSETUP procedure, UG454.
Multipliers, UG413.
MVSTATS instruction, RM309.

N
Names
for numbered series, RM146, RM266.
legal form, UG4, UG144.
Negative binomial distribution density, RM541, RM546.
Nelson, C. R., UG344.
Neural networks, UG469.
convergence of, UG473.
fitted values/predictions, RM333.
limit values of, UG474.
tips on fitting, UG472.
training/fitting, RM328.

Model selection
stepwise regressions, RM449.
Models
building with GROUP, RM224.
coefficients of, RM544.
combining with +, RM53.
companion matrix, RM543.
defining with SYSTEM, RM466.
dependent variables, RM444.
dynamic linear, RM116.
equations
extracting, RM544.
locating, RM544.
replacing, RM544.
forecasting, RM188, UG409.
instructions for, RM590.
simulations, RM423.
size of, RM545.
state-space, RM116.
MODIFY instruction, RM306, UG410.

Modular arithmetic
%CLOCK, RM524.
%MOD, RM543.
Monte Carlo
antithetic acceleration, UG496.
bookkeeping for, UG478.
importance sampling, UG500.
impulse response standard errors, UG497.
integration, UG495.
tests, RM423, UG486.
MONTEARCH.PRG example, UG486.
MONTEVAR.PRG example, UG497.
Monthly data, RM28.
Morgan, I. G., UG423.

Moving average
adding lags to equation, RM494.

Moving Window Statistics wizard, RM309.

MRESTRICT instruction, RM307, UG206, UG220.
MSVARSETUP procedure, UG454.
Multipliers, UG413.
MVSTATS instruction, RM309.

N
Names
for numbered series, RM146, RM266.
legal form, UG4, UG144.
Negative binomial distribution density, RM541, RM546.
Nelson, C. R., UG344.
Neural networks, UG469.
convergence of, UG473.
fitted values/predictions, RM333.
limit values of, UG474.
tips on fitting, UG472.
training/fitting, RM328.
abort editing/close file, RM370.
adding replacing data, RM490.
deleting series from, RM104.
editing series on, RM130.
editing creating, RM101.
finding series length, RM254.
instructions for, UG82.
listing contents of, RM35.
older versions of, UG84.
panel data, UG544.
printing series, RM364.
RATS instructions for, RM592.
reading data from, RM80.
renaming series, RM378.
saving data to, RM246, RM404, RM146.
RATS Variables wizard, UG142.
RATS DATA program
renaming series, RM378.
RE estimator. See Random effects.
READ instruction, RM374, UG145.
with binary format files, UG103.
with complex series, RM401.
with FORTRAN format, UG96.
Ready/Local modes, UG2.
REAL data type, UG137, UG143, UG146.
Real numbers
from complex, RM554.
 fractional part, RM533.
from integers, RM533.
rounding, RM566.
RECTANGULAR arrays, UG143, UG148.
Recursive least squares, RM392, UG195.
tests using, UG238.
Recursive residuals, RM259, UG195, UG238.
graphing example, UG125.
VARS, UG390.
REGCORRS procedure, UG231, UG332.
REGCRITS procedure, UG203.
REGRESET procedure, UG240.
Regression format, UG142.
Regression tables, RM562.
Regression Tests wizard, RM164, RM307, RM385, RM473, UG33, UG222.
Regressions. See also LINREG instruction.
AR1
with panel data, UG550.
with AR1 errors, RM6.
band spectrum, UG573.
F statistic, UG178.
fitted values, RM358.
forecasting, UG298.
instructions for, RM589.
with LINREG, UG25.
non-parametric, RM336.
ordinary least squares, RM272.
output, UG226, UG176.
panel data, RM353.
quantile, RM396.
recursive least squares, RM392.
reporting results, UG169.
restricted, RM135, RM307, RM385, UG205, UG220.
ridge, UG208.
robust, RM396.
rolling, UG138.
seemingly unrelated, RM457, UG197.
standard Normal linear model, UG183.
statistics
accessing, UG142.
defined by LINREG, RM276.
stepwise, RM449.
testing restrictions, RM164.
using SWEEP, RM464.
Regressions wizard, RM6, RM272.
tests, UG220.
with panel data, UG550.
another to, RM307.
Restrictions
across equations, UG199.
coefficient vector, UG223.
covariance matrix, UG223.
exclusion, RM164, UG220.
on non-linear parameters, UG264, UG271.
regressions, RM135, RM307, RM385, RM590, UG205, UG220.
tests, UG220.
RETURN instruction, RM390, UG583.
REWIND instruction, UG391.
Ridge regression, UG208.
RLS instruction, RM392.
Robins, R. P., UG421.
Robust estimation, UG214.
ROBUST PRG example, UG215.
ROBUSTERRORS option, RM295, UG184, UG211.
for heteroscedasticity, UG187.
for serial correlation, UG189.
ROBUSTLMTEST procedure, UG225.
Rolling regressions, UG138, UG305.
Root mean square error
for forecasts, RM476.
Roots
using FIND, RM179, UG293.
Rossi, P., UG489.
Rounding function, RM556.
Rounding function, RM556.
Sample selection bias, UG537.
Sample range, RM427.
Sample selection bias, UG537.
Sample range, RM427.
SCATTER instruction, RM405, UG21.
Savin, N. E., UG182.
Scalars
calculations, RM50, UG36.
variables, UG136.
SCATTER instruction, RM405, UG21, UG108.
with complex series, UG564.
with SGRAPH, RM434.
Scatter plots, RM405, UG21.
multiple per page, RM434.
multiple scales, RM434.
Schwarz criterion, UG203, UG204, UG320.
Schwarz, G., UG203.
Schwert, G., UG249.
SCREEN I/O unit, RM339.
Seasonal
differencing, RM109.
dummies, RM413, UG49.
forecasting models, UG316.
masks, RM43.
removing with linear regression, RM173.
Seasonal adjustment
Census X11, RM507.
exponential smoothing, RM153.
frequency domain, UG571.
SEASONAL instruction, RM413.
SEASONALDL procedure, UG446.
Serial correlation
Sequential probit, UG530.
Sequence
Selectivity bias, UG533.
SELECT instruction, RM418, UG598.
Selectivity bias, UG533.
Series
Seemingly unrelated
SEED instruction, RM416.
Secant function, RM557.
making an array from a set
looping over list of, UG581.
listing all, RM422.
listing over list of, UG581.
making an array from a set
of, RM287.
maximum of, RM439.
mean of, RM309, RM439, RM468.
median of, RM439, RM468.
minimum of, RM439.
missing values, UG17, UG50, UG78.
names, RM146, RM266, UG4, UG11.
numbers, RM266, RM150, RM582, RM595.
panel data, RM346, RM348.
printing, RM54, RM356, UG16.
from a RATS file, RM364.
ranks, RM344.
reading from a file, RM80, UG10, UG39, UG70.
referencing from a label, RM557.
renaming on a RATS format file, RM378.
sampling at an interval, RM402.
scalar computations, UG153.
scatter plots of, RM405.
selecting from list, RM418, UG598.
setting by elements, RM52, RM4240.
skipping observations, UG180.
SMPL, UG180.
sorting, RM344.
storing on a RATS format files, RM101, RM246, RM446.
transformations, RM420, UG17, UG48.
trend, UG17, UG48.
writing to files, RM54, UG70.
Series Window
generating statistics, UG13.
SET instruction, RM420, UG17, UG48.
Shapiro, N., UG489.
SHOLLER.PRG example, UG210.
Shocks
adding to forecasts, RM192.
responses to, RM240.
SHOCKS option example, UG414.
SHORTANDLONG procedure, UG359.
Show Series Window wizard, RM246, RM356, RM446, RM496.
Significance levels
computing, RM36, UG220.
definition, UG227.
interpreting, UG179.
Simplex method
non-linear optimization, UG262.
Sims, C. A., RM264, RM373, UG202, UG232, UG251, UG343, UG347, UG351, UG353, UG365, UG392, UG393, UG571.
Sims exogeneity test, UG232.
SIMULADD.PRG example, UG414.
SIMULATE instruction, RM423, UG478, UG482.
Simulations, RM423, UG47, UG482.
bookkeeping for, UG478.
computing a probability, UG479.
historical, UG413.
using FORECAST, RM188.
SIMULEST.PRG example, UG406.
SIMULFOR.PRG example, UG412.
SIMULMUL.PRG example, UG413.
Simultaneous equation models
accuracy statistics, UG413.
add factors, UG414.
algorithm for solving, UG411.
constructing models, RM224.
convergence, UG412.
example, UG406.
formulas, UG409.
historical simulations, UG413.
identities, RM196.
multipliers, UG413.
out-of-sample forecasts, UG412.
preliminary transformations, UG404.
three-stage least squares, UG197.
Sine
function, RM558.
hyperbolic, RM558.
inverse, RM521.
Single-Equation Forecasts
wizard, RM368, RM486, UG300.
Singleton, K.J., UG281.
Singular value decomposition, RM560, UG162.
Skewness, RM439, UG13.
SKIPSAVE option, RM193.
Smith, A. F. M., UG505.
Smoothing
complex series, RM502.
exponential, RM153, UG314.
Kalman, UG437.
SMPL instruction, RM427, UG64, UG65.
current setting, RM254.
SMPL option, UG180.
using Kalman filter, RM259.
using SMPL instruction, RM427.
Solving
linear equations, RM558.
models, UG411.
Sorting
arrays, RM558.
data series, RM344.
vectors, RM536.
SOURCE instruction, RM429, UG58, UG584, UG598.
Spearman’s rho, RM344.
SPECFORE procedure, UG334, UG574.
SPECFORE.PRG example, UG335.
Specification tests, UG232, UG242, UG281. See also Tests.
Harvey-Collier, UG196.
SPECIFY instruction, RM430, UG380.
Spectral analysis, UG562.
  complex expressions, UG558.
  density, UG566, UG568.
  forecasting, UG314, UG334.
  window, RM502, UG566.
Spectral density, RM352.
Spectral density function, RM187.
SPECTRUM procedure, UG568.
Spencer filter, RM173.
SPGRAPH instruction, RM214, RM434, UG108, UG123.
Splines, UG202.
SPREAD option, UG186.
  with panel data, RM347.
Spreadsheets, UG73.
  adding dates to files, UG89.
  reading data from, UG85.
  troubleshooting, UG89.
Square root function, RM559.
  complex, RM526.
  elementwise, RM559.
SSTATS instruction, UG153.
Statistical tests, UG234, UG238.
See also Tests.
CUSUM, RM393.
STABTEST procedure, UG234.
STABTEST.SRC procedure, UG234, UG238.
Standard error
  of autocorrelations, RM61.
  of coefficients, vector of, RM276.
  of estimate, UG178.
  of forecast, RM150, UG307.
  of projection, RM358.
  of series, RM468.
STARTUP option, UG541.
State space model. See DLM.
Static forecasts, UG34, UG299.
Statistics
  forecast performance, RM476.
  instructions for computing, RM591.
  marginal significance levels, RM36.
  moving window, RM309.
  regression, RM276, UG176.
  for series, RM439, RM468, UG13.
STATISTICS instruction, RM437, RM439, UG13.
STEPS instruction, RM443, UG300.
  saving results, UG303.
Stepwise regressions, RM449, UG43.
Stereo projection, RM559.
Stochastic volatility model, UG448, UG489.
Stock, J., UG252.
STORE instruction, RM446, UG82.
STRING data type, UG143, UG146.
Strings
  comparison, RM559.
  concatenating, UG118.
  expressions with
    COMPUTE, RM52.
extracting substring, RM539, RM542, RM555.
  functions, UG155.
  from integer, RM559.
  length of, RM559.
  line breaks in, UG118.
  to lower case, RM560.
  numerical value from, RM566.
  operations with, RM571.
  reading from files, UG118.
  from real value, RM560.
  selecting from list, RM418.
  to upper case, RM560.
  variables, UG154.
Structural break tests. See Stability tests.
Structural breaks
  in ARIMA models, UG336.
  Structural VAR, RM75, UG353, UG362, UG370.
  Sub-FRML’s, RM199.
Subprograms, UG583.
Sum
  of an array, RM560.
  of coefficients, RM453.
  partial, RM2.
  of squared residuals, UG178.
SUMMARIZE instruction, RM453, UG201, UG220.
  Supplementary cards, UG20, UG61.
  ENTRIES option, UG217.
  long, UG61.
  in procedures, RM138.
  sets of, RM278.
SUR instruction, RM457, UG197.
  example of, UG200.
  technical information, UG197.
  three stage least squares, UG405.
  with VAR’s, UG454.
SUR.PRG example, UG200.
SV.PRG example, UG448.
Swamy, P. A. V. B., UG554.
SWAMY.PRG example, UG554.
SWAMY.SRC procedure, UG554.
Swamy’s random coefficients, UG554.
SWARCH.PRG example, UG458.
Sweep functions, RM560.
SWEEP instruction, RM464, UG549.
  Switch options, UG61.
SYMMETRIC data type
  aggregate, UG143, UG148.
Symmetric matrix
  packing/unpacking
    functions, RM561.
Syntax errors, UG66.
SYSTEM instruction, RM466, UG342.
  Systems of equations
    estimating, RM457.
    forecasting, RM188.
    linear estimation, UG197.
T
distribution
  CDF instruction, RM36.
  density, RM563.
  inverse CDF, RM538.
  inverse tail probability, RM538.
  multivariate density, RM541.
  multivariate random draw, UG484.
  non-central
    CDF, RM563.
    density, RM563.
    random draws, RM553, UG483.
    tail probability, RM564.
T variable for SET, RM421, UG17.
t-statistics, RM439, UG27.
  vector of coefficient, RM276.
TABLE instruction, RM468, UG13.
  from Series Window, UG13.
Tangent
  function, RM563.
  hyperbolic, RM563.
  inverse, RM521.
TAPER instruction, RM471, UG567.
  Tapers, RM471, UG567.
Taquq, M. S., UG576.
TAR model, UG196.
Taylor, W. E., UG552.
TEST instruction, RM473, UG31, UG220.
Tests
  analysis of variance, RM366.
  applicability, UG222.
  ARCH, UG227.
  Bayesian odds ratio (unit root), UG251.
  Breusch-Pagan, UG226.
  causality, UG232.
  Chow, UG234, UG236.
  coefficient values, RM473.
  cumulated periodogram, RM26, UG231.
  Diebold-Mariano, UG309.
  Engle-Granger cointegration, UG253.
  exogeneity, UG232.
  Goldfeld-Quandt, UG228.
  Hansen specification, UG245.
  Harvey-Collier misspecification, UG196.
  Hausman specification, UG243.
  heteroscedasticity, UG226, UG229.
  Johansen cointegration, UG253.
  lag length, UG348.
  Lagrange multiplier, UG224.
  likelihood ratio, RM372.
  linear restrictions, RM307, RM385.
  MacKinnon-White-Davidson, UG241.
  marginal significance levels, UG220.
  Monte Carlo, RM423.
Index

normality, RM439.
overidentifying restrictions. See
  Overidentifying restrictions. RESET, UG240.
serial correlation, RM61, UG230.
specification, UG242.
structural stability, UG234.
unit roots, UG249.
White heteroscedasticity, UG227.

Teukolsky, S. A., RM60.
Text files, UG39, UG74, UG95.
Theil, H., UG200, UG208, UG285,
  UG345, UG382.
THEIL instruction, RM476, UG327,
  UG383, UG413.
Theil U statistic, RM476, UG308,
  UG383.
Three-stage least squares, RM457,
  UG405, UG406.
non-linear, RM320.
Threshold autoregression, UG196.
Time effects, RM366, UG546.
computing means of, RM346.
Time-varying coefficients, RM484,
  UG391.
estimating with KALMAN, RM259,
  more general, UG394.
Time/date stamp, RM527.
TO triples, UG42.
Tobit models, UG533, UG539.
TOBIT.PRG example, UG539.
Today
  entry number of, RM564.
  Trace function, RM564.
  TRACE option, UG257.
  Trading days, RM564.
Transfer function
  models, RM17.
spectral, RM482.
Transformations wizard, RM2,
  RM281, RM420, UG19.
Transpose function, RM564, UG159.
TRAPEZOID taper, RM471.
Trend
  creating, RM420, UG17, UG48.
  and forecasts, UG315.
  removing, RM109.
  removing with HP filter, RM173,
    UG446.
  removing with linear regression, RM173.
Trend/Seasonals/Dummies
  wizard, RM413, UG19, UG49.
TRFUNC instruction, RM482, UG563.
Trigamma function, RM482.
Truncated samples, RM268, UG533.
TSAYP149.PRG example, RM488.
TVARSET procedure, UG391.
TVARYING instruction, RM484,
  UG391.
TVARYING.PRG example, UG391.
Two-stage least squares, UG192,
  UG276, UG406.
with AR1 errors, RM6.
  instruments list, RM257.
  non-linear, RM313.
  TYPE instruction, RM485, UG587.

U
UFORECAST instruction, RM486,
  UG34, UG300, UG482.
UFOREERRORS procedure, UG308,
  UG327.
Unbalanced panel data sets, RM348.
Uncentered R-squared, UG177.
Underflow, UG256.
Uniform distribution
  random draws, RM565, UG160,
  UG478, UG483.
UNION.PRG example, UG532.
Upper case
  converting string to, RM560.
USERMENU instruction, RM491,
  UG598.

V
VADD instruction, RM306, RM494.
Valid data
testing for, RM566.
Value at Risk (VaR), UG518.
Value options, UG61.
VAR
  Bayesian, RM430, UG377.
  Blanchard-Quah, UG358, UG362.
  block exogeneity tests, RM372.
  Choleski factorization, UG352,
    UG362.
  conditional forecasts, UG397,
    UG400.
  decomposition of variance, RM147.
  deterministic variables, RM108.
  differencing variables, UG343.
  dummy observation prior, UG396.
  error correction, UG374.
  estimating, RM158, UG345.
  estimating with LINREG, UG345.
  estimating with SUR, UG345.
  exogenous variables, RM108.
  forecasting, RM188, UG377.
  historical decompositions, RM233,
    UG372.
  hypothesis tests, UG347.
  impulse responses, RM148, RM240,
    UG364, UG368.
  instructions for, RM590.
  Kalman filter, UG388.
  lag length, UG344, UG348, UG379.
  lag sums, RM544.
  lags, setting list of, RM267.
  long-run restriction, UG358.
  Markov switching, UG454.
  moving average
    representation, UG351.
    near, UG342, UG345.
  orderings, UG363, UG367.
  orthogonalization, UG351.
  partial, UG386.
  preliminary
    transformations, UG343.
    prior, UG378, UG380, UG383.
    random draw for
      coefficients, UG492.
    seasonal data, UG387.
    selection of variables, UG385.
    setting up a, RM466, RM495,
      UG342.
  SPECIFY instruction, UG380.
  structural, RM75, UG353, UG362,
    UG370.
  time-varying coefficients, RM259,
    RM263, RM484, UG391.
  trends, UG344.
  variance decomposition, UG366.
  VAR (Forecast/Analyze)
    wizard, RM147, RM188, RM233,
      RM240, UG346.
  VAR (Setup/Estimate)
    wizard, RM108, RM128, RM158,
      RM267, RM430, RM466, RM495,
      UG342.
  VARCAUSE.PRG example, UG350.
Variables
  accessible, UG142.
  data types, UG143.
  declaring, RM99, UG144.
  defined by RATS
    instructions, UG142.
  displaying values of, RM113,
    UG140.
  global, RM280.
  input/output of, UG145.
  label of, RM539.
  local in procedures, RM280, UG583,
    UG588.
  names (form of), UG144.
  reading values into, RM251,
    RM374.
  reserved, RM573.
  scalar, UG136.
  setting with COMPUTE, RM50.
  user input for, RM368.
Index

VARIABLES instruction, RM495, UG342.

Variance
- components model, RM366, UG552.
- decomposition of, RM147, UG366.
- for equation, RM12.
- for FRML, RM197.
- moving window, RM309.
- of series, RM439.

VARIREF procedure, UG368.

VARLAG.PRG example, RM384, UG348.

VARLAG.PRG example program, UG348.

VCV instruction, RM496.

VCV option
- output from, UG43, UG179.

Vector
- rectangular from a, UG159.
- symmetric from a, UG159.

Vector Autoregressions. See VAR.

VECTOR data type, RM100.

aggregate, UG143, UG148.

Vettering, W. T., RM60.

VREPLACE instruction, RM306, RM499, UG410.

Wald test, UG223.

- for logit, probit, RM97.

Weekly data, RM28.

Weight matrix
- instrumental variables estimator, UG279.

Weighted least squares, RM272, UG186.
- with panel data, UG548.

Weiss, L., UG367.

West, K., UG185.

WHILE instruction, RM501, UG56, UG581.

White
- covariance matrix, RM295, UG187.
- heteroscedasticity test, UG227.

White, H., UG227.

White, K. J., UG182.

Window
- data, RM471.
- directing output to, RM339.
- displaying data in, UG145.
- displaying information in, UG598.

Window (spectral)
- leakage, RM471.
- smoothing, RM502.

WINDOW instruction, RM502, UG566.

- MASK option, RM43.

WINDOW option
- PRINT instruction, RM497.
- saving forecasts, UG304.

Windows Metafile format, RM140.

Wishart distribution
- random draws, RM554, UG485.

Wizards
- ARCH/GARCH, RM206.
- Box-Jenkins (ARIMA models), RM17.
- Calendar, RM28, UG7.
- Covariance Matrix, RM496.
- Cross Correlations, RM66.
- Data, RM80, UG12.
- Data (RATS Format), UG81.
- Differencing, RM109, UG19.
- Equation/FRML, RM142, RM196.
- Exponential Smoothing, RM153.
- Functions, UG138.
- Graph, RM214, UG23.

Limited/Discrete Dependent Variables, RM92, RM268.

Moving Window Statistics, RM309.

overview, UG5.

Panel Data Regressions, RM353.

RATS Variables, UG142.

Regression Tests, RM164, RM307, RM385, RM473, UG33, UG222.

Regressions, RM6, RM272, RM396, RM449, UG28.

Show Series Window, RM246, RM356, RM446, RM496.

Single-Equation Forecasts, RM358, RM486, UG300.

Transformations, RM2, RM281, RM420, UG19.

Trend/Seasonals/Dummies, RM413, UG19.


VAR (Forecast/Analyze), RM147, RM188, RM233, RM240, UG346.

VAR (Setup/Estimate), RM108, RM128, RM158, RM267, RM430, RM466, RM495, UG342.

X11, RM507.

WKS files, RM80, UG39, UG73, UG85.

WLS. See Weighted least squares.

WMATRIX option, UG279.

Wooldridge, J. M., UG423.

WRITE instruction, RM505, UG145.

with binary-format files, UG103.

with complex series, RM401.

with FORTRAN format, UG96.

X
- X11 instruction, RM507.
- X11 wizard, RM507.

Y
- Yule-Walker equations, RM60, RM250.

Z
- Zellner GLS, RM457.
- Zha, T., UG365.
- ZUMEAN option, UG280.