Economic analysis with RATS 4.30

Christopher F Baum

Department of Economics and

Faculty Microcomputer Resource Center

Boston College

September 30, 1998

Faculty
Microcomputer
Resource
Center

Introduction

- **RATS**: an econometric programming language with particular strengths in the analysis of time series data in time and frequency domains
- Most similar in capabilities to Eviews/MicroTSP
- More of a programming language than a ‘canned’ econometrics program
- Less well suited to panel data or cross-section data analysis than Stata

Portability

- A portable, cross-platform-capable language
- Versions available for Windows 3.1/95/98/NT, as well as flavors of UNIX and Alpha (OpenVMS)
- Binary datafiles transportable without translation across all platforms
- RATS programs run without modification across all platforms
- Full versions available at reasonable cost to academics; student purchase plan

Extensibility

- Many RATS 'procedures' available from Estima website, including many user-contributed routines for sophisticated analysis
- As a programming language, RATS is well suited to implementation of sophisticated estimators without concern for all of the low-level details which must be handled in a matrix programming language such as GAUSS or MATLAB
- Formal syntax much cleaner than that of GAUSS

User Support

- Users and Estima developers participate vigorously in RATS-L, a moderated LISTSERV mailing list, responding rapidly to users' enquiries. Archives of the list may be searched on the BC Economics web site
- All RATS-L procedures are posted to the Boston College Statistical Software Components Archive on IDEAS, from which they may be freely downloaded. The archive is searchable.

Dataset concepts

- Data within RATS usually follows a ‘calendar’ of annual, quarterly, monthly, weekly, or daily frequency
- Each series (variable) may start and end at different points in time, subject to a maximum ALLOC value
- Binary datasets created by RATS usually are given the filetype `.rdb`; series may be retrieved from binary datasets via random access methods

‘Batch’ mode

- RATS may be run interactively or in ‘batch’ mode. It is always a good idea to write a RATS program and execute it, so that you can replicate your results.
- In UNIX RATS, run the program ‘`myjob.rats`’ with
  `rats  myjob.rats > myjob.out`
  which will place the output in `myjob.out`.
- To run this as a true batch job, give the UNIX command ‘`batch`’ first, type the RATS command line, and use CTRL-D to submit the batch job.

Case sensitivity

- Unlike UNIX, RATS is not case-sensitive. The variables price, PRICE, and Price are the same variable.
- However, file names must be given in the same case in which they appear in the operating system.
- UNIX commands can be given from within RATS with the ‘dos’ command. However, the commands must follow UNIX syntax rules.

Getting your data in

- RATS is quite flexible in terms of data input.
- Spreadsheet files (in several formats) can be read directly into RATS if a few conditions are satisfied. Dates in the spreadsheet will be converted into RATS' calendar.
- ASCII files can be read in either observation-wise (ORG=OBS) or variable-wise (ORG=VAR) format. Variables (which RATS calls 'series') may be of differing lengths.

Getting your data in

- If a record (or several records) on an external file contain the values of a single observation (time period), then the external file is read with the ORG=OBS option. This corresponds to the usual presentation of a ‘data table’ in a spreadsheet, with time periods labeling the rows.
- If a record (or several records) on an external file contain the values of a single variable, or series, then the external file is read with the ORG=VAR option.

Getting your data in

- RATS supports many data types, but ‘series’ (variables) must be purely floating-point numeric values. String (character) variables cannot be placed in series. However, string variables may be read as such and loaded into arrays.
- RATS implements integer, real, complex, label (for series) and string data types, as well as composite data types of vector, rectangular, symmetric, and series. Only the series data type is handled with the DATA command.

Getting your data in

- A free-format text file with space-delimited or comma-delimited numeric data may be read with the DATA command. Tab-delimited files cannot be read.
- A fixed-format text file may be read using the FORMAT= option on the DATA command, where a FORTRAN format is specified.
- The MISSING= option on the DATA command may be used to flag missing data on input.

Getting your data in

- The `DATA` command has a `UNIT=` option by which an external file may be specified. An '`OPEN DATA filename`' command preceding DATA indicates where the data are to be found. Any number of external text files may be sequentially accessed by using `CLOSE DATA` after each is read.
- Series may be written to an external file using the `COPY` command in any of several formats.

Getting your data in

- The commands
  `OPEN DATA /u/baum/macro.dat`
  `DATA(org=obs,unit=data,format=free)`
  `1964:1 1996:2 gdp rtb3 m1a`
  would read those three variables' quarterly observations, in observation-wise free format, from the specified external file.
- Neither the `UNIT=` nor the `FORMAT=` options would have to be explicitly specified, as they are the default values. `ORG=VAR` is the default.

Faculty Microcomputer Resource Center

Getting your data in

- If each of those macro series started and/or ended in different quarters, they could be read with three DATA statements, each specifying the appropriate start-end range, with the default ORG=VAR option.
- If data are to be read into the entire data space defined by ALLOC (and, if used, CALENDAR), the slash (/) may be used in place of the start-end range. This convention holds for all RATS commands in which start-end range is used.

Getting your data in

- Once you read data from an external file, generate new variables, and want to work with the data at a later date, you should convert the data to RATS' internal (binary) format of the '`.rdb`' (RATS database) file. The data may then be read much more quickly; their names and calendar range (if used) is retained; and a subsequent `DATA(FORMAT=RATS)` statement may specify any subset of the variables on the `.rdb` file (i.e. random access is supported).

Working with stored data

- The RATS database file is created with the `DEDIT(NEW) filename.rdb` command. Series are added to the file with the `STORE` command, and the file is generated with the SAVE command.
- Series may be added to the .rdb file, or existing series revised, at a later date via `DEDIT`.
- The RATS .rdb file may be transferred via binary-mode FTP to any platform which supports RATS, without any conversion process (such as that required for SAS or GAUSS binary files).

Language syntax

- Each RATS command is a three-letter ‘verb,’ possibly modified by options, followed by zero or more ‘objects.’ Options appear in a parenthesized list directly after the verb name. For instance: `stat(higher,print) gnp` executes the `statistics` command, with options `higher` and `print`, on the series gnp.
- Commands longer than one line require a ‘$’ at the end of each line to be continued.

Language syntax

- Some RATS commands require a ‘supplementary card’ following the command. This line must start with a pound sign ‘#’. For instance, a linear regression of the series y on the series x, r, q and a constant term is specified by

```
linreg y
# constant x r q
```

and the correlation of those variables via

```
cmom(corr,print)
# y x r q
```

Allocate/Calendar

- RATS uses the concept of a 'data matrix' with a specified maximum number of rows (periods) and an indeterminate number of columns. The maximum is given with ALLOCATE. If a number is given, that defines the largest number of observations which may be used in the program.
- If time series or panel data are used, a CALENDAR statement defines the periodicity of the data, and the ALL statement refers to the latest period which may be referenced in the program.

Calendar

- If the data are time series, a calendar should always be used, as output will then be labelled by the calendar periods, and instructions can refer to calendar dates (run a regression over this period, forecast over this period, etc.)
- If the data are panel data corresponding to a 'balanced' panel, calendar may be used to specify the organization of the data. If the data are an unbalanced panel, RATS is not the appropriate program for its analysis; Stata is preferred.

Calendar syntax

- `CAL n1 n2 n3`
  specifies the first year to be n1, the first period in that year to be n2, and the number of periods/year to be n3; e.g. cal 1964 2 4 specifies that the first row of the data matrix should be labelled 1964Q2.
- Data will then be specified as year:period: 1964:1 would be 1964Q1, January 1964, or the year 1964. If annual data are used, the '`:1`' <u>must</u> be used to distinguish calendar 1964 from observation 1,964!

Allocate syntax

- The `ALLOCATE` statement specifies the maximum observation number which may be referenced. For undated data, it is an integer. If a `CALENDAR` statement precedes `ALL`, then `ALL` will be in terms of dates: `ALL 1998:12` would specify December 1998 as the latest possible date, following the establishment of a monthly calendar. The `CAL/ALL` settings cannot be altered within a program.
- Do not use `ALL 1998` for annual data– `ALL 1998:1` is the appropriate command.

Defining the data

- Columns of the data matrix defined by ALL (and optionally CAL) may then be loaded by the DATA command. Any number of DATA commands may be used to combine data from different sources.
- The DATA command options COMPACT and SELECT may be used to convert data of a different timeseries frequency 'on the fly'
- New series (variables) may be created with the SET command, or as a byproduct of computation (e.g. residuals or predicted values after regression)

Checking the data

- After reading data from an external file, use the TABLE command to report descriptive statistics and ensure that the data have been read correctly.
- The STAT command generates descriptive statistics on a single series (and may specify an observation range, e.g.
  `stat gdp 1964:1 1992:4`)
- The EXTREMUM command locates extrema in a single series and optionally saves their values and locations as scalar variables

Defining the SMPL

- The SMPL command redefines the 'working sample', in terms of either observation numbers or calendar dates. The 'working sample' is that referred to by '/' on any command.
- Unlike Stata, there is no way in which the current sample may be revised to meet a logical condition (unless that condition involves contiguous observations). However, a dummy variable may be defined and a SMPL option used on many commands: e.g. STAT(SMPL=WAR) GDP.

Printing data

- To examine the data, use the `PRINT` command.
  `PRINT [range] variables`
  prints those variables, using either an explicit start-end range (such as `1964:1 1998:2`) or '`/`' to refer to the default range. The range must be given.
- If a calendar is in use, observations will be labeled by calendar date rather than their observation number.
- If you want to move the data to another program, use `COPY` rather than `PRINT`.

Data transformations

- New series are generated (or existing series revised) with the SET command.
  `SET varname [range]  =  expression`
  creates (or revises) the series varname. The range need not be given; if not, it applies to the currently defined sample (SMPL).
- Spaces <u>must</u> surround the equals sign in the SET command.

The SET command

- The expression on the RHS of the SET command may be any combination of existing series, scalars, and built-in functions.
- The value t denotes the observation number; e.g.

```
set trend  =  t
```

creates a linear trend, and

```
set ltr  =  log(t)
```

creates a logarithmic trend.

The SET command

- Think of the SET command as defining a formula to be applied to each observation in the current sample.
- The expression need not contain time subscripts, but if you want to refer to different time periods, you may via curly braces: gdp`{4}` is the fourth lag of the gdp series, whereas gdp`{-1}` is the first lead.
- The `DIFF` command may be used to generate new series which are the differences of existing series.

The SET command

- It is often unnecessary to define new variables which are lags or leads of existing variables, as they may be specified ‘on the fly’ in many commands. For instance
```
linreg dy
# dg{1 2 3 4}  dm{1 to 4}
```
specifies that four lags of each RHS variable are to be entered in the regression. If lags (leads) are contiguous, the ‘to’ may be used to avoid listing them; e.g. `{-4 to 4}` would define 9 regressors.

The SET command

- The SET command will automatically set undefined entries to the system missing value (%NA). These include lags or leads which do not exist, as well as mathematical operations which do not return valid values (log of negative datum).
- %NA values will not enter any statistical computations, and will propagate through subsequent SET statements. Use TABLE or STAT to ensure that creation of missing data values is appropriate for your sample.

Common operations

- The `DIFF series [range] newseries` command generates the first difference of the series. The range must be given, but may be '/'. The command can also generate higher-order differences, seasonal differences, centered differences, and standardized differences.
- The `EXP` and `LOG` commands will generate series equal to those transformations of the specified series. They may be applied in place.

Data transformations

- The standard operators `+, -, *, /` are augmented by `**` for exponentiation.
- Logical operators include '==', and the FORTRAN-style `.EQ., .GT., .GE., .LT., .LE.,` unary `.NOT.`, as well as conjunctives `.AND., .OR.`
- Rules of precedence are similar to those of FORTRAN, but parentheses may be used freely to ensure that the desired result is achieved.

Data transformations

- When the ‘expression’ on a SET statement generates a pure logical condition, a binary variable (or ‘dummy’) is created, taking on the values 0 (FALSE) or 1 (TRUE). Such a variable may be used in either logical or arithmetic expressions, or in statements combining the two.
- These binary variables may be used to restrict analysis with commands’ `SMPL=` option, or entered in regressor lists. Note that the mean of such a variable is a sample proportion.

Data transformations

- Note that if SET is used to define a dummy variable, two statements are generally needed:
  `SET D70 1970:1 1979:4  =  1`
  `SET D70 1980:1 1988:4  =  0`
  If only the first statement was given, `D70` would be missing (%NA) for the observations of the 1980s. This could be achieved in one statement via
  `SET D70  =  t .LE. 1979:4`
  where the ‘`t`’ variable will reference dates if a timeseries calendar is in effect.

Data transformations

- The `%IF` function may be used to define more complex results of logical conditions: e.g.
```
set uptick  = %if(price.gt.price{1},
price-price{1}, 0)
```
would define the series `uptick` as the <u>amount</u> by which the price rose, or zero.
- If 'downtick' values were to be <u>excluded</u> from further computations, setting their value to %NA rather than 0 would achieve this result.

Data transformations

- Other common functions include `ABS`, `EXP`, `LOG`, `SQRT`, `COS`, `SIN`, `TAN` for arithmetic transformations.
- Functions such as `FIX` and `FLOAT` allow for integer arithmetic, while `%VALID` permits defined values to be identified.
- Functions also exist for date arithmetic, handling of complex numbers, string manipulation, PDF/CDFs of common statistical distributions, and generation of uniform or normal random numbers.

Data transformations

- An assortment of matrix functions are available, including `INV`, `TR`, `%COLS`, `%ROWS`, `%ABS`, `%EXP`, `%LOG`, `%SQRT`, `%SCALAR`, `%MSCALAR`, `%CONST`, `%DIAG`, `%CORR`, `%COV`, `%DECOMP` (Choleski), `%DET`, `%TRACE`, `%DOT`, `%IDENTITY`, `%KRONEKER`, `%KRONID`, `%MQFORM`, `%QFORM`, `%UNIFORM`, `%RAN`, and `%SUM`.
- Matrices are defined as a byproduct of most statistical routines.

The LINREG command

- The basic OLS regression is performed with
```
linreg depvar [range] [resids]
# regressors
```
(including constant if desired)
- If the range is not specified, the default range (as defined by the SMPL command) will be used.
- If a series name is given (following '/' if a specific range is not given), OLS residuals will be stored in that series:
```
linreg gdp / gdphat
# constant g{1 to 4} m(1 to 4}
```

The LINREG command

- LINREG generates a number of scalar and matrix quantities: %NOBS, %NREG, %NDF, %RSQUARED, %SEESQ, %DURBIN, %QSTAT as scalar results, while %BETA is a vector(%NREG) of the estimated coefficients, and %XX is a symmetric matrix (the inverse of X'X). Elements of %SEESQ*%XX are estimated variances of the regression coefficients. The VCV option prints the covariance matrix.
- The ROBUSTERRORS option (with LAGS) calculates Newey-West standard errors.

The LINREG command

- Instrumental variables (two-stage least squares) estimators may be specified with `LINREG(INSTRUMENTS)`. A prior `INSTRUMENTS` command must list all exogenous variables (including the constant) to be used in the IV regression.
- Predicted values for the regression sample may be generated with `PRJ yhat` following `LINREG`. With a range, out-of-sample (static) predictions may be generated with the same command.

The LINREG command

- More complicated predictions (such as dynamic, n-step-ahead forecasts from a dynamic model) as well as predictions from a simultaneous equations model may be generated from the `FORECAST` command. The equations to be used must be defined (most readily via the `define=n` option on `LINREG`), and may be `GROUP`ed into a model. Many summary statistics of prediction accuracy may be computed from the simulation of a dynamic multiple-equation model.

Hypothesis testing

- Hypothesis testing on single equations may be conducted. The regression must first be estimated via `LINREG`. Tests may then be performed, or the regression may be reestimated subject to linear restrictions.
- The `EXCLUDE` command tests exclusion (zero) restrictions for a set of regressors, providing the value of the test statistic and p-value for its significance. Several `EXCLUDE` commands may be given following a single regression.

Hypothesis testing

- The TEST command may be used to test simple linear hypotheses. For instance,
  TEST
  ```
  # 4
  # 5.0
  ```
  tests that the fourth coefficient in the equation is equal to 5.0, while
  TEST
  ```
  #  2     3    5
  # 1.0 -2.5  3.0
  ```
  tests these joint hypotheses.

Hypothesis testing

- Hypotheses involving linear combinations of coefficients may be tested via the `RESTRICT` command:

```
REST   1
#  2  3  4
#  1.0  1.0  1.0  1.0
```

  might be used to test CRTS, that the sum of coefficients 2, 3, and 4 is 1.0.
- An equation may be estimated subject to linear restrictions via `RESTRICT(CREATE)`.

Hypothesis testing

- Nonlinear hypotheses may be tested via the `RATIO` command, which invokes a likelihood ratio test.
- The `CDF` command allows evaluation of a test statistic against one of several common distributions, with specification of the appropriate degrees of freedom parameter(s).
- All test and restriction commands produce tail probabilities for the test statistic, using the appropriate distribution and degrees of freedom.

Other estimation techniques

- Weighted least squares via SPREAD option
- AR(1) models: AR1 command
- Autocorrelation functions: CORRELATE, CROSS
- Distributed lag models (PDL, Shiller)
- Box-Jenkins models: BOXJENK command
- Nonlinear least squares/NLIV: NLLS command
- Method of moments (single & multiple equation)
- Linear systems: SUR and SUR(INST) for 3SLS
- Nonlinear systems: NLSYSTEM command

Other estimation techniques

- Generalized optimization: `MAXIMIZE` command
- ARCH and related models via procedures
- Unit root tests via procedures
- Vector autoregressions, including BVARs
- Impulse response functions
- Kalman filter techniques, time-varying parameters
- Frequency-domain techniques, including spectral analysis, cross-spectral analysis, filtering

Programming in RATS

- RATS is a full-featured programming language
- Procedures with formal parameters, local vs. global scope, full generality may be developed in the RATS language
- Procedures may make use of scalar, vector, matrix, and complex datatypes to generate built-in capability to implement any estimator or sequence of data transformations (e.g. Johansen procedure for determining rank of a cointegrating relation)

Programming in RATS

- `DECLARE` command is used to define datatypes of objects of scalar or array type
- `DIMENSION` command is executable, and is used to define size of arrays
- Datatypes may be combined: e.g. you may define a vector of symmetric matrices
- Calculations with scalars are performed with the `COMPUTE` command

Programming in RATS

- A rectangular matrix of series' contents is produced with the MAKE command
- The SET command may be used to transfer the contents of a vector (or a column of a matrix) to a series
- Most RATS commands define one or more scalars (e.g. %NOBS, %NVAR, %NDF) which may be used in further computations

Programming in RATS

- Matrix expressions may be arbitrarily complex, involving all standard operators of linear algebra, with functions available to produce standard results such as quadratic forms and Kronecker products.
- The `EWISE` command may be used to produce element-by-element matrix operations such as the Hadamard product.
- The `EIGEN` command generates the eigensystem of a real symmetric matrix.

Programming in RATS

- Results of computations with scalars may be printed with the `DISPLAY` command. A ‘picture’ format may be used to customize the output.
- The `WRITE` command is used to output vectors and matrices. It may be directed to place its output to an external file using either free or FORTRAN format.
- The `INPUT` command (which reads scalars or arrays) may read from an external file.

Programming in RATS

- A number of procedures are provided with RATS, available in `/usr/local/rats/examples` and referenced in the RATS manual. They provide an indication of what may be done in RATS programming. RATS procedures must be explicitly included in your program using the `SOURCE` command.
- Procedures may exchange formal parameters with the calling program, and may use supplementary cards to direct their execution.

```
********************************************************************************
*  kpss series  start  end
*
*    Performs unit root tests with the null being stationarity
*    of the series
*
*    References: Denis Kwiatkowski, Peter Phillips, and
*                Peter Schmidt (Michigan State University, 1990)
*
*    Options:
*     MAXLAG=number of additional lags [0]
*       MAXLAG indicates the number of lags used in calculating the variance
*              of the sum (or accumulation) of the random changes
*
*
*  Written by John Barkoulas November 1992
*
*
********************************************************************************
procedure kpss series start end
type series series
type integer start end
option integer maxlag 0

local integer startl endl
local series resids st2 trend
local real sumst2 sl2
local vec nm nt
dim  nm(maxlag+1) nt(maxlag+1)

inquire(series=series) startl>>start endl>>end
```

```
***********************************************************
*         KPSS Level Stationary                           *
***********************************************************
linreg(noprint) series startl endl resids
# constant
acc resids startl endl st2
set st2 startl endl = st2(t)*st2(t)
acc st2 startl endl st2
comp sumst2 = st2(endl)

do aa=0,maxlag
mcov(damp=1,lags=aa) startl endl resids
# constant
comp sl2 = %cmom(1,1) / %nobs
comp nm(aa+1) = sumst2 / ((%nobs**2)*sl2)
end do aa

* dis 'Test Statistics for KPSS Level Stationary Model'
do bb = 0,maxlag
dis  'KPSS [level stationary] test for lag' bb ' : ' nm(bb+1)
end do bb
```

```
***********************************************************
*        KPSS Trend Stationary Model                        *
***********************************************************
set trend startl endl = t

linreg(noprint) series startl endl resids
# constant trend
acc resids startl endl st2
set st2 startl endl = st2(t)*st2(t)
acc st2 startl endl st2
comp sumst2 = st2(endl)

do cc = 0,maxlag
mcov(damp=1.0,lags=cc) startl endl resids
# constant
comp sl2 = %cmom(1,1) / %nobs

comp nt(cc+1) = sumst2 / ((%nobs**2)*sl2)
end do cc

* dis 'Test Statistics for the KPSS Trend Stationary Model'
do dd = 0,maxlag
dis  'KPSS [trend stationary] test for lag' dd ' : ' nt(dd+1)
end do dd

end
```

Programming in RATS

- The calling sequence for a procedure, invoked by `EXECUTE procname` or `@procname`, may use the full syntax of built-in commands, including options that take on binary values (such as `print / noprint`) as well as numeric options (e.g. `lags=4`) with a specified default value.
- Procedures may be fully general in their handling of series passed to them; they may `INQUIRE` as to the current sample, and may access parameters by value or by address (by reference).

Programming in RATS

- Repetitive tasks–in your program or in a procedure–may be mitigated by the use of 'numbered series', and by several looping constructs: `DO, DOFOR serieslist, WHILE, UNTIL,` and `LOOP` with a BREAK exit.
- The `MVSTATS` and `MVFRACTILES` commands generate 'moving window' estimates of moments, medians, and percentiles of series

Graphics

- RATS contains quite sophisticated graphics commands, including the facility to combine graph types on the same graph, produce sets of graphs from a single command, and shade areas on the graph (e.g. recessions). The graph and axis labels may be generated with string variables.
- Graphics may not be viewed when working in the UNIX command-line environment, but may be generated and sent to a PostScript printer.

Graphics

- If graphics are used in the UNIX character-mode environment, the command `ENV NOSHOWGRAPHS` is required.
- The `GRAPH` command produces graphs of series, histograms, and bar graphs.
- The `SCATTER` command produces X-Y scatter plots.