Gretl Manual

Gnu Regression, Econometrics and Time-series Library

Allin Cottrell
Department of Economics
Wake Forest University

January, 2002

Gretl Manual: Gnu Regression, Econometrics and Time-series Library
by Allin Cottrell

Table of Contents

List of Tables

List of Figures

List of Examples

Chapter 1. Introduction

gretl is an econometrics package, including a shared library, a command-line client program and a graphical user interface.

Features at a glance

User-friendly

gretl offers an intuitive user interface; it is very easy to get up and running with econometric analysis. Thanks to its association with Ramanathan's *Introductory Econometrics* the package offers many practice data files and command scripts. These are well annotated and accessible.

Flexible

You can choose your preferred point on the spectrum from interactive point-and-click to batch processing, and can easily combine these approaches.

Cross-platform

gretl's home platform is Linux, but it is also available for MS Windows. I have compiled it on AIX and it should work on any unix-like system that has the appropriate basic libraries (see Appendix B).

Open source

The full source code for gretl is available to anyone who wants to critique it, patch it, or extend it. The author welcomes any bug reports.

Reasonably sophisticated

gretl offers a full range of least-squares based estimators, including Two-Stage Least Squares. It also offers (binomial) logit and probit estimation, and has a loop construct for running Monte Carlo analyses or iterated least squares estimation of non-linear models. While it does not include all the estimators and tests that a professional econometrician might require, it supports the export of data to the formats of (GNU R) and (GNU Octave) for further analysis (see Appendix C).

Accurate

gretl does well on the NIST reference datasets. For documentation of this point, see the gretl website.

Internet ready

gretl can access and fetch databases from a server at Wake Forest University. The MS Windows version comes with an updater program which will detect when a new version is available and offer the option of auto-updating.

Acknowledgements

My primary debt is to Professor Ramu Ramanathan of the University of California, San Diego. A few years back he was kind enough to provide me with the source code for his program ESL ("Econometrics Software Library"), which I ported to Linux, and since then I have collaborated with him on updating and extending the program. For the gretl project I have made extensive changes to the original ESL code. New econometric functionality has been added, and the graphical interface is entirely new. Please note that Professor Ramanathan is not responsible for any bugs in gretl.

I am also grateful to William Greene, author of *Econometric Analysis*, for permission to include in the gretl distribution some of the data files analysed in his text.

I have benefitted greatly from the work of numerous developers of open-source software: for specifics please see Appendix B to this manual. My thanks are due to Richard Stall-

man of the Free Software Foundation, for his support of free software in general and for agreeing to "adopt" `gretl` as a GNU program in particular.

Installing the programs

Linux/unix

On the Linux platform you have the choice of compiling the `gretl` code yourself or making use of a pre-built package. Ready-to-run packages are available in `rpm` format (suitable for Red Hat Linux and related systems) and also `deb` format (Debian GNU/Linux). I am grateful to Dirk Eddelbüttel for making the latter. If you prefer to compile your own (or are using a unix system for which pre-built packages are not available) here is what to do.

1. Download the latest `gretl` source package from gretl.sourceforge.net.
2. Unzip and untar the package. On a system with the GNU utilities available, the command would be `tar -xvfz gretl-N.tar.gz` (replace `N` with the specific version number of the file you downloaded at step 1).
3. Change directory to the gretl source directory created at step 2 (e.g. `gretl-0.98`).
4. The basic routine is then

   ```
   ./configure
   make
   make install
   ```

   However, you should probably do `./configure --help` first to see what options are available. One option you way wish to tweak is `--prefix`. By default the installation goes under `/usr/local` but you can change this. For example `./configure --prefix=/usr` will put everything under the `/usr` tree. In the event that a required library is not found on your system, so that the configure process fails, please take a look at Appendix B of this manual.

As of version 0.97 `gretl` offers support for the `gnome` desktop. To take advantage of this you should compile the program yourself (as described above). If you want to suppress the `gnome`-specific features you can pass the option `--without-gnome` to `configure`.

MS Windows

The MS Windows version comes as a self-extracting executable. Installation is just a matter of downloading `gretl_install.exe` and running this program. You will be prompted for a location to install the package (the default is `c:\userdata\gretl`).

Updating

If your computer is connected to the Internet, then on start-up `gretl` can query its home website at Wake Forest University to see if any program updates are available; if so, a window will open up informing you of that fact. If you want to activate this feature, check the box marked "Tell me about gretl updates" under `gretl`'s "File, Preferences, General" menu.

The MS Windows version of the program goes a step further: it tells you that you can update `gretl` automatically if you wish. To do this, follow the instructions in the popup window: close `gretl` then run the program titled "gretl updater" (you should find this along with the main `gretl` program item, under the Programs heading in the Windows Start menu). Once the updater has completed its work you may restart `gretl`.

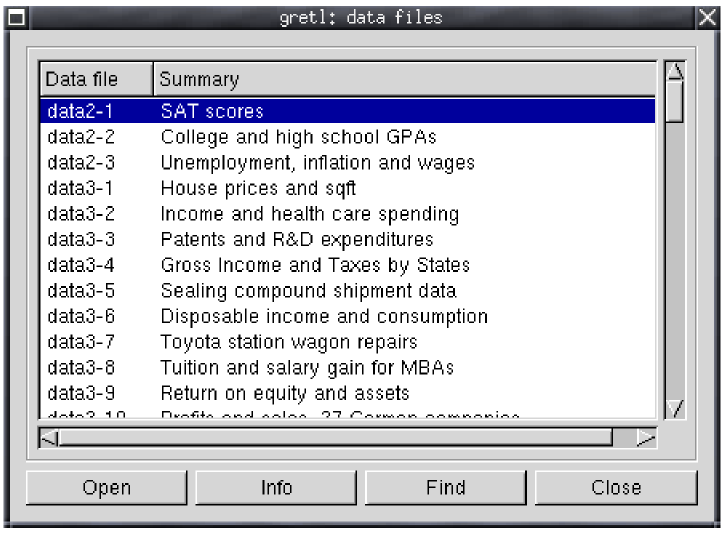
Chapter 2. Getting started

This introduction is mostly angled towards the graphical client program; please see Chapter 10 and Chapter 12 below for details on the command-line program, `gretlcli`.

You can supply the name of a data file to open as an argument to `gretl`, but for the moment let's not do that: just fire up the program.[1] You should see a main window (which will hold information on the data set but which is at first blank) and various menus, some of them disabled at first.

What can you do at this point? You can browse the supplied data files (or databases), open a data file, create a new data file, read the help items, or open a command script. For now let's browse the supplied data files. Under the File menu choose "Open data, sample file, Ramanathan...". A second window should open, presenting a list of data files supplied with the package (see Figure 2-1). The numbering of the files corresponds to the chapter organization of Ramanathan (2002), which contains discussion of the analysis of these data. The data will be useful for practice purposes even without the text.

Figure 2-1. Practice data files window



Data file	Summary
data2-1	SAT scores
data2-2	College and high school GPAs
data2-3	Unemployment, inflation and wages
data3-1	House prices and sqft
data3-2	Income and health care spending
data3-3	Patents and R&D expenditures
data3-4	Gross Income and Taxes by States
data3-5	Sealing compound shipment data
data3-6	Disposable income and consumption
data3-7	Toyota station wagon repairs
data3-8	Tuition and salary gain for MBAs
data3-9	Return on equity and assets

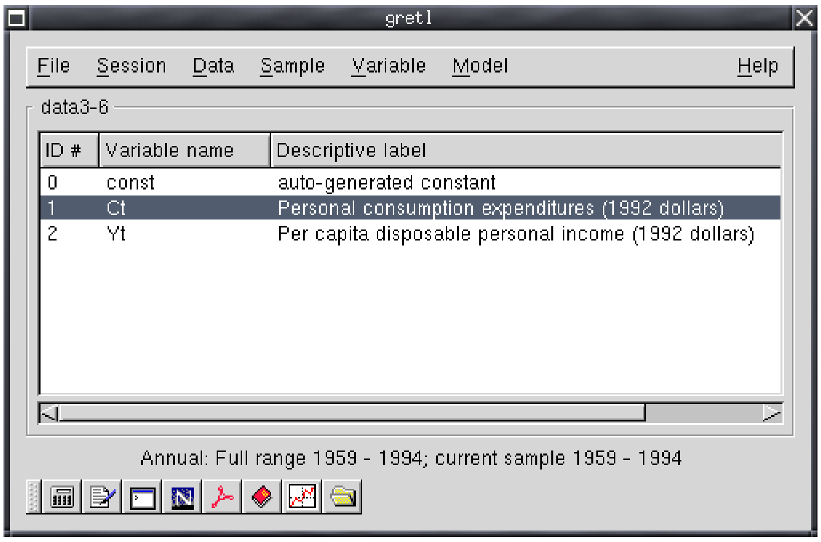
If you select a row in this window and click on "Info" this pops open the the "header file" for the data set in question, which tells you something about the source and definition of the variables. If you find a file that is of interest, you may open it by clicking on "Open", or just double-clicking on the file name. For the moment let's open `data3-6`.

In `gretl` windows containing lists, double-clicking on a line launches a default action for the associated list entry: e.g. displaying the values of a data series, opening a file.

This file contains data pertaining to a classic econometric "chestnut", the consumption function. The data window should now display the name of the current data file, the overall data range and sample range, and the names of the variables along with brief descriptive tags — see Figure 2-2.

1. For convenience I will refer to the graphical client program simply as `gretl` in this manual. Note, however, that the specific name of the program differs according to the computer platform. On unix-like systems it is called `gretl_x11` while on MS Windows it is `gretlw32.exe`. On unix-like systems a wrapper script named `gretl` is also installed — see also Chapter 9.

Figure 2-2. Main window, with a practice data file open



ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	Ct	Personal consumption expenditures (1992 dollars)
2	Yt	Per capita disposable personal income (1992 dollars)

Annual: Full range 1959 - 1994; current sample 1959 - 1994

OK, what can we do now? Hopefully the various menu options should be fairly self explanatory. For now we'll dip into the Model menu; a brief tour of all the main window menus is given in Chapter 3 below.

`gretl`'s Model menu offers numerous various econometric estimation routines. The simplest and most standard is Ordinary Least Squares (OLS).

Selecting OLS pops up a dialog box calling for a *model specification*. This takes the form of a list of variable names or numbers, separated by spaces. The first name or number represents the dependent variable, the remainder the independent variables. It is usual to include a constant (ID number 0) among the independent variables (otherwise you are forcing the intercept to equal zero).

You can put a variable's ID number into the dialog box by clicking on that variable's row in the main window.

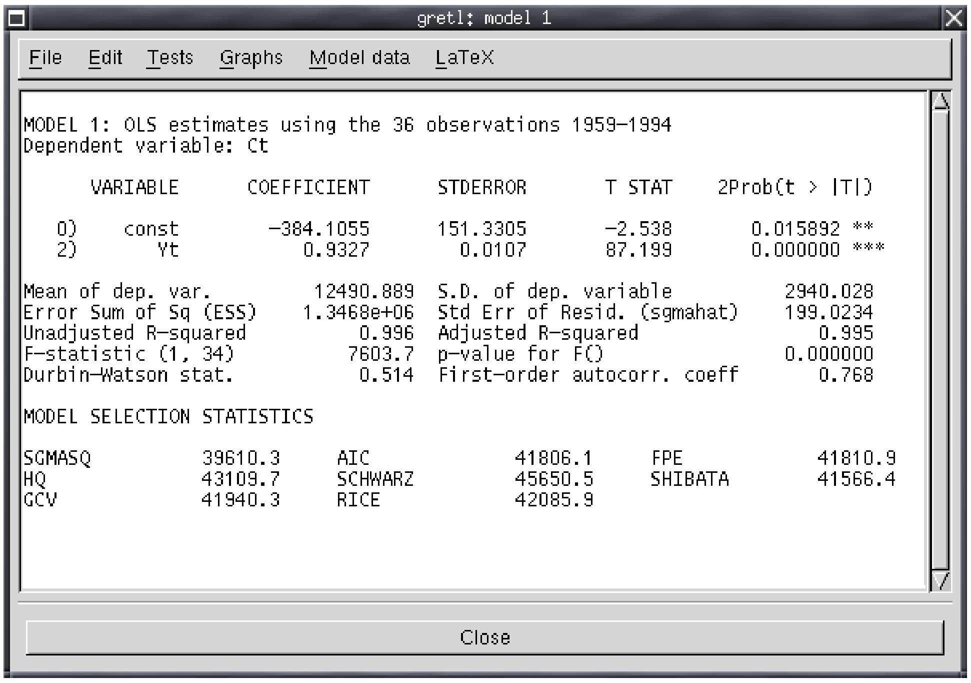
Thus, continuing the example of `data3-6`, the entry `1 0 2` (or equivalently `Ct 0 Yt`) specifies a regression of consumption (dependent) on income and a constant.

You can specify a lagged value of an existing variable without explicitly adding this to the data set first. Thus a variant on the above estimation command which includes the first lag of income on the right-hand side would be `1 0 2 Yt(-1)` The lag is selected using a negative integer enclosed in parentheses. Note that in this context the name, not the number, of the variable in question must be used.

Estimation output

Once you've specified a model, a window displaying the regression output will appear. The output is reasonably comprehensive and in a standard format (Figure 2-3).

Figure 2-3. Model output window



```
MODEL 1: OLS estimates using the 36 observations 1959-1994
Dependent variable: Ct

     VARIABLE      COEFFICIENT      STDERROR       T STAT    2Prob(t > |T|)

   0)    const       -384.1055      151.3305       -2.538       0.015892 **
   2)       Yt          0.9327        0.0107       87.199       0.000000 ***

Mean of dep. var.         12490.889  S.D. of dep. variable        2940.028
Error Sum of Sq (ESS)    1.3468e+06  Std Err of Resid. (sgmahat)  199.0234
Unadjusted R-squared          0.996  Adjusted R-squared              0.995
F-statistic (1, 34)          7603.7  p-value for F()              0.000000
Durbin-Watson stat.           0.514  First-order autocorr. coeff     0.768

MODEL SELECTION STATISTICS

SGMASQ          39610.3     AIC             41806.1      FPE            41810.9
HQ              43109.7     SCHWARZ         45650.5      SHIBATA        41566.4
GCV             41940.3     RICE            42085.9
```

The output window contains menus that allow you to inspect or graph the residuals and fitted values, and to run various diagnostic tests on the model.

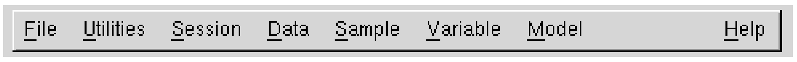
There is also an option to reprint the regression output in LaTeX format. This is not fully implemented yet, but works for OLS models. You can print the results in a tabular format (similar to what's in the output window, but properly typeset) or as an equation, across the page. For each of these options you can choose to preview the typeset product, or save the output to file for incorporation in a LaTeX document. Previewing requires that you have a functioning TeX system on your computer.

If you want to import `gretl` output into an editor or word processor there are two main options. You can simply copy and paste from an output window (using its Edit menu) to the target program, or you can save the output to a file then import the file into the target program. When you finish a `gretl` session you are given the option of saving all the output from the session to a single file.

When inserting `gretl` output into a word processor, select a monospaced or typewriter-style font (e.g. Courier) to preserve the output's tabular formatting. Select a small font (10-point Courier should do) to prevent the output lines from being broken in the wrong place.

Chapter 3. The main window menus

Reading left to right along the main window's menu bar, we find the File, Utilities, Session, Data, Sample, Variable, Model and Help menus.



§ File menu

—Open data: Open a native `gretl` data file or import from other formats. See Chapter 5.

—Save data: Save the currently open native `gretl` data file.

—Save data as: Write out the current data set in native format, with the option of using gzip data compression or storing the data in binary format. See Chapter 5.

—Export data: Write out the current data set in Comma Separated Values (CSV) format, or the formats of GNU R or GNU Octave. See Chapter 5 and also Appendix C.

—Clear data set: Clear the current data set out of memory. Generally you don't have to do this (since opening a new data file automatically clears the old one) but sometimes it's useful (see the Section called *Creating a data file from scratch* in Chapter 5).

—Browse databases: See the Section called *Binary databases* in Chapter 5 and the Section called *Creating a data file from scratch* in Chapter 5.

—Create data set: Initialize the built-in spreadsheet for entering data manually. See the Section called *Creating a data file from scratch* in Chapter 5.

—Save last graph: Just as it says.

—View command log: Open a window containing a record of the commands executed so far.

—Open command file: Open a file of `gretl` commands, either one you have created yourself or one of the practice files supplied with the package. If you want to create a command file from scratch use the next item, New command file.

—Preferences: Set the paths to various files `gretl` needs to access. Choose the font in which gretl displays text output. Select or unselect "expert mode". (If this mode is selected various warning messages are suppressed.) Activate or suppress `gretl`'s messaging about the availability of program updates. Configure or turn on/off the main-window toolbar.

—Exit: Quit the program. If expert mode is not selected you'll be prompted to save any unsaved work.

§ Utilities menu

—Statistical tables: Look up critical values for commonly used distributions (normal or Gaussian, t, chi-square, F and Durbin–Watson).

—p-value finder: Open a window which enables you to look up p-values from the Gaussian, t, chi-square, F or gamma distributions. See also the `pvalue` command in Chapter 10.

—Test statistic calculator: Calculate test statistics and p-values for a range of common hypothesis tests (population mean, variance and proportion; difference of means, variances and proportions). The relevant sample statistics must be already available for entry into the dialog box. For some simple tests that take as input data series rather than pre-computed sample statistics, see "Difference of means" and "Difference of variances" under the Data menu.

- Gretl console: Open a "console" window into which you can type commands as you would using the command-line program, `gretlcli` (as opposed to using point-and-click). See Chapter 10.
- Start Gnu R: Start `R` (if it is installed on your system), and load a copy of the data set currently open in `gretl`. See Appendix C.

§ Session menu

- Icon view: Open a window showing the current `gretl` session as a set of icons. For details see the Section called *The "session" concept* in Chapter 4.
- Add last graph: Capture the last graph shown in the form of a session icon, so that it can be revisited and manipulated.
- Open: Open a previously saved session file.
- Save: Save the current session to file.
- Save as: Save the current session to file under a chosen name.

§ Data menu

- Display values: pops up a window with a simple (not editable) printout of the values of the variables (either all of them or a selected subset).
- Edit values: pops up a spreadsheet window where you can make changes, add new variables, and extend the number of observations. (The data matrix must remain rectangular, with the same number of observations for each series.)
- Graph specified vars: Gives a choice between a time series plot, a regular X–Y scatter plot, an X–Y plot using impulses (vertical bars), an X–Y plot "with factor separation" (i.e. with the points colored differently depending to the value of a given dummy variable) and boxplots. Serves up a dialog box where you specify the variables to graph. The simplest way to fill out the dialog entry is to refer to the variables by their ID numbers (shown in the leftmost column of the main data window). Thus, having chosen the scatter plot option, an entry of "2 3" will plot variable number 2 (here, consumption) against variable number 3 (income). The last referenced variable will be on the x axis. Gnuplot is used to render the graph (except for the boxplots option).
- Multiple scatterplots: Show a collection of (at most six) pairwise plots, with either a given variable on the y axis plotted against several different variables on the x axis, or several y variables plotted against a given x. May be useful for exploratory data analysis.
- Read info, Edit header: "Read info" just displays the header file information for the current data file; "Edit header" allows you to make changes to it (if you have permission to do so).
- Summary statistics: shows a fairly full set of descriptive statistics for all variables in the data set.
- Correlation matrix: shows the pairwise correlation coefficients for the variables in the data set.
- Difference of means: calculates the t statistic for the null hypothesis that the population means are equal for two selected variables and shows its p-value.
- Difference of variances: calculates the F statistic for the null hypothesis that the population variances are equal for two selected variables and shows its p-value.
- Add variables gives a sub-menu of standard transformations of variables (logs, lags, squares, etc.) that you may wish to add to the data set. Also gives the option of adding random variables, and (for time-series data) adding seasonal dummy variables (e.g. quarterly dummy variables for quarterly data). Includes an item for seeding the program's pseudo-random number generator.

—Refresh window Sometimes gretl commands generate new variables. The "refresh" item ensures that the listing of variables visible in the main data window is in sync with the program's internal state.

§ Sample menu

—Set range: Select a different starting and/or ending point for the current sample, within the range of data available.

—Restore full range: self-explanatory.

—Set frequency, startobs: Impose a particular interpretation of the data in terms of frequency and starting point. This is primarily intended for use with panel data; see Chapter 6.

—Define, based on dummy: Given a dummy (indicator) variable with values 0 or 1, this drops from the current sample all observations for which the dummy variable has value 0.

—Restrict, based on criterion: Similar to the item above, except that you don't need a pre-defined variable: you supply a Boolean expression (e.g. `sqft > 1400`) and the sample is restricted to observations satisfying that condition. See the help for `genr` in Chapter 10 for details on the Boolean operators that can be used.

—Drop all obs with missing values: Drop from the current sample all observations for which at least one variable has a missing value (see the Section called *Missing data values* in Chapter 5).

—Count missing values: Give a report on observations where data values are missing. May be useful in examining a panel data set, where it's quite common to encounter missing values.

—Add case markers Prompts for the name of a text file containing "case markers" (short strings identifying the individual observations) and adds this information to the data set. See Chapter 5.

—Panel structureOpens a dialog box which enables you to fix the interpretation of a panel data set as either stacked time series or stacked cross sections (see Chapter 6).

§ Variable menu Most items under here operate on a single variable at a time. The "active" variable is set by highlighting it (clicking on its row) in the main data window. Most options will be self-explanatory. Note that you can rename a variable, and can edit its descriptive label. You can also "Define a new variable" via a formula (e.g. involving some function of one or more existing variables). For the syntax of such formulae, look at the online help for "Generate variable syntax" or see the `genr` command in Chapter 10. One simple example:

```
foo = x1 * x2
```

will create a new variable `foo` as the product of the existing variables `x1` and `x2`. In these formulae, variables must be referenced by name, not number.

§ Model menu This is introduced in Chapter 2. For details on the various estimators offered under this menu please consult the Section called *Estimators and tests: summary* in Chapter 10 and Chapter 10 below, and/or the online help under "Help, Estimation".

§ Help menu Please use this as needed! It gives details on the syntax required in various dialog entries.

Chapter 4. Modes of working

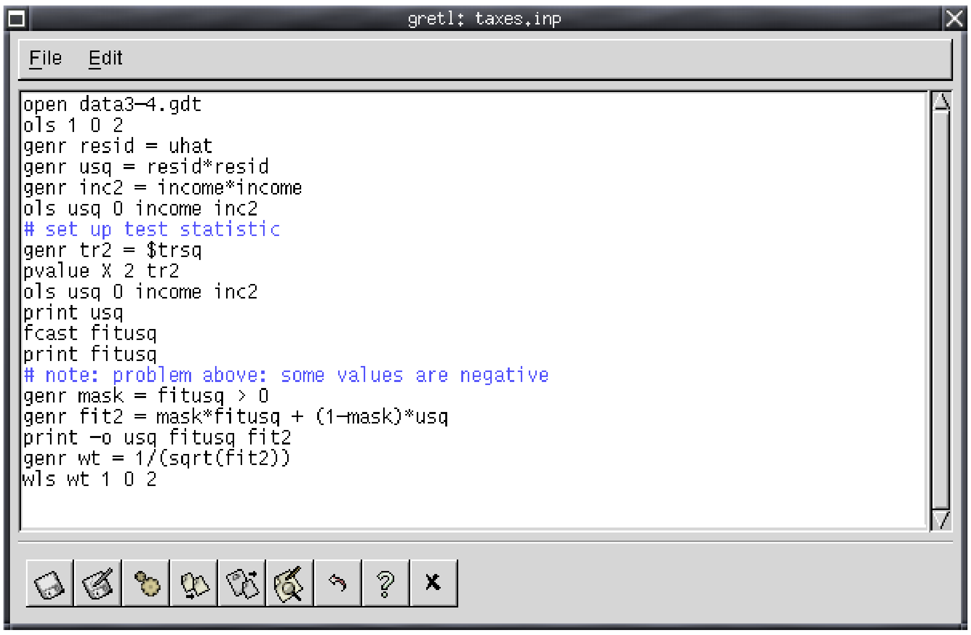
Command scripts

As you execute commands in `gretl`, using the GUI and filling in dialog entries, those commands are recorded in the form of a "script". Such scripts can be edited and re-run, using either `gretl` or the command-line client, `gretlcli`.

To view the current state of the script at any point in a `gretl` session, choose "Command log" under the File menu. This log file is called `session.inp` and it is overwritten whenever you start a new session. To preserve it, save the script under a different name. Script files will be found most easily, using the GUI file selector, if you name them with the extension "`.inp`".

To open a script you have written independently, use the "File, Open command file" menu item; to create a script from scratch use the "File, New command file" item. In either case a script window will open (see Figure 4-1).

Figure 4-1. Script window, editing a command file



```
open data3-4.gdt
ols 1 0 2
genr resid = uhat
genr usq = resid*resid
genr inc2 = income*income
ols usq 0 income inc2
# set up test statistic
genr tr2 = $trsq
pvalue X 2 tr2
ols usq 0 income inc2
print usq
fcast fitusq
print fitusq
# note: problem above: some values are negative
genr mask = fitusq > 0
genr fit2 = mask*fitusq + (1-mask)*usq
print -o usq fitusq fit2
genr wt = 1/(sqrt(fit2))
wls wt 1 0 2
```

The toolbar at the botton left of the script window offers the following functions (left to right): (1) Save the file; (2) Save the file under a specified name; (3) Execute the commands in the file; (4) Copy selected text; (5) Paste the selected text; (6) Find and replace text; (7) Undo the last Paste or Replace action; (8) Help (if you place the cursor in a command word and press the question mark you will get help on that command); (9) Close the window.

These functions (and in addition a Print option) may also be found under the "File" and "Edit" menus at the top of the script window.

When you click the Execute icon or choose the "File, Run" menu item all output is directed to a single window, where it can be edited, saved or copied to the clipboard.

To learn more about the possibilities of scripting, take a look at the `gretl` Help item "Script commands syntax," or start up the command-line program `gretlcli` and consult its help, or consult Chapter 10 in this manual. In addition, the `gretl` package includes over 70 "practice" scripts. Most of these relate to Ramanathan (2002), but they may also be used as a free-standing introduction to scripting in `gretl` and to various points of econometric theory. You can explore the practice files under "File, Open command file, practice file" There you will find a listing of the files along with a brief description of the points they illustrate and the data they employ. Open any file and run it ("File, Run" in the resulting script window) to see the output.

Note that long commands in a script can be broken over two or more lines, using backslash as a continuation character.

You can, if you wish, use the GUI controls and the scripting approach in tandem, exploiting each method where it offers greater convenience. Here are two suggestions.

§ Open a data file in the GUI. Explore the data — generate graphs, run regressions, perform tests. Then open the Command log, edit out any redundant commands, and save it under a specific name. Run the script to generate a single file containing a concise record of your work.

§ Start by establishing a new script file. Type in any commands that may be required to set up transformations of the data (see the `genr` command in Chapter 10 below). Typically this sort of thing can be accomplished more efficiently via commands assembled with forethought rather than point-and-click. Then save and run the script: the GUI data window will be updated accordingly. Now you can carry out further exploration of the data via the GUI. To revisit the data at a later point, open and rerun the "preparatory" script first.

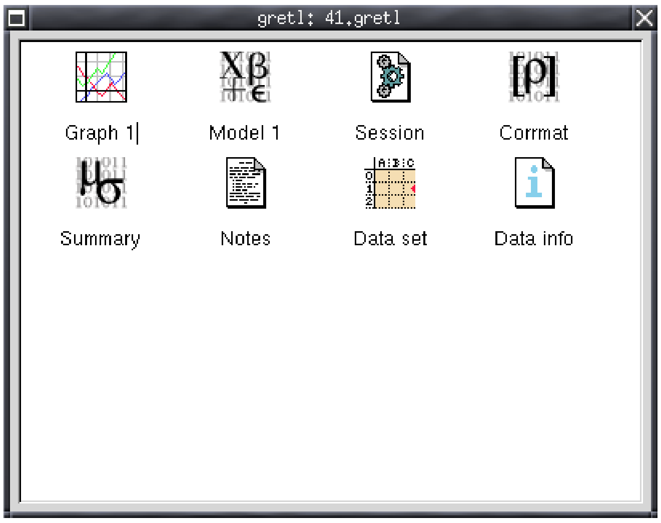
A further option is available for your computing convenience. Under `gretl`'s File menu you will find the item "Gretl console". This opens up a window in which you can type commands and execute them one by one (by pressing the Enter key) interactively. This is essentially the same as `gretlcli`'s mode of operation, except that (a) the GUI is updated based on commands executed from the console, enabling you to work back and forth as you wish, and (b) `gretl`'s Monte Carlo loop routine (see the Section called *Monte Carlo simulations* in Chapter 8) is not at present available in this mode.

The "session" concept

`gretl` offers the idea of a "session" as a way of keeping track of your work and revisiting it later. This is experimental (and at present more likely to be buggy than the rest of the program); I would be interested in hearing people's reactions.

The basic idea is to provide a little iconic space containing various objects pertaining to your current working session (see Figure 4-2). You can add objects (represented by icons) to this space as you go along. If you save the session, these added objects should be available again if you re-open the session later.

Figure 4-2. Icon view: one model and one graph have been added to the default icons



If you start gretl and open a data set, then select "Icon view" from the Session menu, you should see the basic default set of icons: these give you quick access to the command script ("Session"), information on the data set (if any), correlation matrix ("Corrmat") and descriptive summary statistics ("Summary"). All of these are activated by double-clicking the relevant icon. The "Data set" icon is a little more complex: double-clicking opens up the data in the built-in spreadsheet, but you can also right-click on the icon for a menu of other actions.

In many gretl windows, the right mouse button brings up a menu with common tasks.

Two sorts of objects can be added to the Icon View window: models and graphs.

To add a model, first estimate it using the Model menu. Then pull down the File menu in the model window and select "Save to session as icon..." or "Save as icon and close". Simply hitting the S key over the model window is a shortcut to the latter action.

To add a graph, first create it (under the Data menu, "Graph specified vars", or via one of gretl's other graph-generating commands), then choose "Add last graph" from the Session menu or click the little camera button on the toolbar (see the Section called *The gretl toolbar*).

Once a model or graph is added its icon should appear in the Icon View window. Double-clicking on the icon redisplays the object, while right-clicking brings up a menu which lets you display or delete the object. This popup menu also gives you the option of editing graphs.

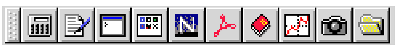
If you create models or graphs that you think you may wish to re-examine later, then before quitting gretl select "Save as..." from the Session menu and give a name under which to save the session. To re-open the session later, either

§ Start gretl then re-open the session file by going to the "Open" item under the Session menu, or

§ From the command line, type `gretl -r` *sessionfile*, where *sessionfile* is the name under which the session was saved.

The gretl toolbar

At the bottom left of the main window sits the toolbar.



The icons have the following functions, reading from left to right:

1. Launch a calculator program. A convenience function in case you want quick access to a calculator when you're working in gretl. The default program is calc.exe under MS Windows, or xcalc under the X window system. You can change the program under the "File, Preferences, General" menu, "Toolbar" tab.
2. Launch an editor or word processor. The default is winword.exe under MS Windows, emacs under X. This is configurable in the same way as the calculator launcher.
3. Open the gretl console. A shortcut to the "Gretl console" menu item (Chapter 3 above).
4. Open the gretl session window.
5. Open the gretl website in your web browser. This will work only if you are connected to the Internet and have a properly configured browser.

6. Open the current version of this manual, in PDF format. As with the previous item, this requires an Internet connection; it also requires that your browser knows how to handle PDF files.
7. Open the help item for script commands syntax (i.e. a listing with details of all available commands).
8. Open the dialog box for defining a graph.
9. Capture the last displayed graph for editing.
10. Open a window listing the datasets associated with Ramanathan's *Introductory Econometrics*.

If you don't care to have the toolbar displayed, you can turn it off under the "File, Preferences, General" menu. Go to the Toolbar tab and uncheck the "show gretl toolbar" box.

Chapter 5. Data files

Native format

gretl has its own format for data files. Most users will probably not want to read or write such files outside of gretl itself, but occasionally this may be useful and full details on the file formats are given in Appendix A.

Other data file formats

gretl will read various other data formats.

§ Comma-Separated Values (CSV) files. These can be brought in using gretl's "File, Open Data, Import CSV..." menu item, or the `import` script command. For details on what gretl expects of a CSV file, see the Section called *Creating a data file from scratch*.

§ Worksheets in the format of either MS `Excel` or `Gnumeric`. These are also brought in using gretl's "File, Open Data, Import" menu. The requirements for such files are given in the Section called *Creating a data file from scratch*.

§ BOX1 format data. Large amounts of micro data are available (for free) in this format via the Data Extraction Service of the US Bureau of the Census. BOX1 data may be imported using the "File, Open Data, Import BOX..." menu item or the `import -o` script command.

When you import data from the CSV or BOX formats, gretl opens a "diagnostic" window, reporting on its progress in reading the data. If you encounter a problem with ill-formatted data, the messages in this window should give you a handle on fixing the problem.

For the convenience of anyone wanting to carry out more complex data analysis, gretl has a facility for writing out data in the native formats of GNU R and GNU Octave (see Appendix C). In the GUI client this option is found under the "File" menu; in the command-line client use the `store` command with the flag `-r` (R) or `-m` (Octave).

Binary databases

For working with large amounts of data I have supplied gretl with a database-handling routine. A *database*, as opposed to a *data file*, is not read directly into the program's workspace. A database can contain series of mixed frequencies and sample ranges. You open the database and select series to import into the working data set. You can then save those series in a native format data file if you wish. Databases can be accessed via gretl's menu item "File, Browse databases".

For details on the format of gretl databases, see Appendix A.

Online access to databases

As of version 0.40, gretl is able to access databases via the internet. Several databases are available from Wake Forest University. Your computer must be connected to the internet for this option to work. Please see the item on "Online databases" under gretl's Help menu.

RATS 4 databases

Thanks to Thomas Doan of *Estima*, who provided me with the specification of the database format used by RATS 4 (Regression Analysis of Time Series), gretl can also handle such databases. Well, actually, a subset of same: I have only worked on time-series databases containing monthly and quarterly series. My university has the RATS G7 database containing data for the seven largest OECD economies and gretl will read that OK.

Visit the gretl data page for details and updates on available data.

Creating a data file from scratch

There are four ways to do this: (1) Use your favorite spreadsheet to establish the data file, save it in Comma Separated Values format if necessary, then use `gretl`'s "Import" option. (2) Use `gretl`'s built-in spreadsheet. (3) Select data series from a suitable database. (4) Use your favorite text editor or other software tools to create the data file independently.

Here are a few comments and details on these methods.

Using a separate spreadsheet

This may be a good choice if you're comfortable with a particular spreadsheet. If you take this option, please pay attention to the specification of what your spreadsheet should look like. `gretl` expects a file that has (a) valid variable names on the first row and (b) a rectangular block of data beneath. A valid variable name is of 8 characters maximum; starts with a letter; and contains nothing but letters, numbers and the underscore character, _. Optionally the first column may contain strings such as dates (8 characters maximum). Such a column should be headed "`obs`" or "`date`", or its first row cell may be left blank. There should be *exactly one* non-data row at the top of the file.

In the case of an import from CSV, the format described above must be respected exactly. In the case of an import from an `Excel` or `Gnumeric` worksheet you have slightly more freedom: you can select the starting row and column for the import, hence skipping any "extra" rows or columns. Within the area selected for importing, however, the above rules apply.

If you use a spreadsheet to prepare your data you are able to carry out various transformations of the "raw" data with ease (adding things up, taking percentages or whatever): note, however, that you can also do this sort of thing easily — perhaps more easily — within `gretl`, by using the tools under the "Data, Add variables" menu and/or "Variable, define new variable".

You may wish to establish a `gretl` data set piece by piece, by incremental importation of data from other sources. This is supported via the "File, Append data" menu items. `gretl` will check the new data for conformability with the existing data set and, if everything seems OK, will merge the data. You can add new variables in this way, provided the data frequency, starting observation and series length all match the existing data set. Or you can append new observations for data series that are already present; in this case the variable names must match up correctly. Note that by default (that is, if you choose "Open data" rather than "Append data"), opening a new data file closes the current one.

Using the built-in spreadsheet

Under `gretl`'s "File, Create data set" menu you can choose the sort of data set you want to establish (e.g. quarterly time series, cross-sectional). You will then be prompted for starting and ending dates (or observation numbers) and the name of the first variable to add to the data set. After supplying this information you will be faced with a simple spreadsheet into which you can type data values. In the spreadsheet window, clicking the right mouse button will invoke a popup menu which enables you to add a new variable (column), to add an observation (append a row at the foot of the sheet), or to insert an observation at the selected point (move the data down and insert a blank row.)

Once you have entered data into the spreadsheet you import these into `gretl`'s workspace using the spreadsheet's "Apply changes" button.

Please note that `gretl`'s spreadsheet is quite basic and has no support for functions or formulas. Data transformations are done via the "Data" or "Variable" menus in the main `gretl` window.

Selecting from a database

Another alternative is to establish your data set by selecting variables from a database. `gretl` comes with a database of US macroeconomic time series and, as mentioned above, the program will reads RATS 4 databases.

Begin with `gretl`'s "File, Browse databases" menu item. This has three forks: "gretl native", "RATS 4" and "on database server". You should be able to find the file `bcih.bin` in the file selector that opens if you choose the "gretl native" option — this file is supplied with the distribution.

You won't find anything under "RATS 4" unless you have purchased RATS data[1]. If you do possess RATS data you should go into `gretl`'s "File, Preferences, General" dialog, select the Databases tab, and fill in the correct path to your RATS files.

If your computer is connected to the internet you should find several databases (at Wake Forest University) under "on database server". You can browse these remotely; you also have the option of installing them onto your own computer. The initial remote databases window has an item showing, for each file, whether it is already installed locally (and if so, if the local version is up to date with the version at Wake Forest).

Assuming you have managed to open a database you can import selected series into `gretl`'s workspace by using the "Import" menu item in the database window (or via the popup menu that appears if you click the right mouse button).

Creating a data file independently

It is possible to create a `gretl` data file using a text editor or software tools such as `awk`, `sed` or `perl`. This may be a good choice if you have large amounts of data already in machine readable form. You will, of course, need to study the `gretl` data formats (XML format or "traditional" format) as described in Chapter 5.

Further note

`gretl` has no problem compacting data series of relatively high frequency (e.g. monthly) to a lower frequency (e.g. quarterly): this is done by averaging. But it has no way of converting lower frequency data to higher. Therefore if you want to import series of various different frequencies from a database into `gretl` *you must start by importing a series of the lowest frequency you intend to use.* This will initialize your `gretl` data set to the low frequency, and higher frequency data can be imported subsequently (they will be compacted automatically). If you start with a high frequency series you will not be able to import any series of lower frequency.

Missing data values

These are represented internally as -999. In a native-format data file they should be represented the same way. When importing CSV data `gretl` accepts any of three representations of missing values: -999, the string `NA`, or simply a blank cell. Blank cells should, of course, be properly delimited, e.g. `120.6,,5.38`, in which the middle value is presumed missing.

As for handling of missing values in the course of statistical analysis, `gretl` does the following:

§ In calculating descriptive statistics (mean, standard deviation, etc.) under the `summary` command, missing values are simply skipped and the sample size adjusted appropriately.

§ In running regressions `gretl` first adjusts the beginning and end of the sample range, truncating the sample if need be. Missing values at the beginning of the sample are

1. See www.estima.com

common in time series work due to the inclusion of lags, first differences and so on; missing values at the end of the range are not uncommon due to differential updating of series and possibly the inclusion of leads.

§ If `gretl` detects any missing values "inside" the (possibly truncated) sample range for a regression it gives an error message and refuses to produce estimates.

Missing values in the middle of a data set are a problem. In a cross-sectional data set it may be possible to move the offending observations to the beginning or the end of the file, but obviously this won't do with time series data. For those who know what they are doing (!), the `misszero` function is provided under the `genr` command. By doing

```
genr foo = misszero(bar)
```

you can produce a series `foo` which is identical to `bar` except that any -999 values become zeros. Then you can use carefully constructed dummy variables to, in effect, drop the missing observations from the regression while retaining the surrounding sample range.[2]

2. `genr` also offers the inverse function to `misszero`, namely `zeromiss`, which replaces zeros in a given series with the missing observation code.

Chapter 6. Panel data

Panel data (pooled cross-section and time-series) require special care. Here are some pointers.

Consider a data set composed of observations on each of n cross-sectional units (countries, states, persons or whatever) in each of T periods. Let each observation comprise the values of m variables of interest. The data set then contains mnT values.

The data should be arranged "by observation": each row represents an observation; each column contains the values of a particular variable. The data matrix then has nT rows and m columns. That leaves open the matter of how the rows should be arranged. There are two possibilities.[1]

§ Rows grouped by *unit*. Think of the data matrix as composed of n blocks, each having T rows. The first block of T rows contains the observations on cross-sectional unit 1 for each of the periods; the next block contains the observations on unit 2 for all periods; and so on. In effect, the data matrix is a set of time-series data sets, stacked vertically.

§ Rows grouped by *period*. Think of the data matrix as composed of T blocks, each having n rows. The first n rows contain the observations for each of the cross-sectional units in period 1; the next block contains the observations for all units in period 2; and so on. The data matrix is a set of cross-sectional data sets, stacked vertically.

You may use whichever arrangement is more convenient. The first is perhaps easier to keep straight. If you use the second then of course you must ensure that the cross-sectional units appear in the same order in each of the period data blocks.

In either case you can use the frequency field in the *observations* line of the data header file (see Chapter 5) to make life a little easier.

§ *Grouped by unit*: Set the frequency equal to T. Suppose you have observations on 20 units in each of 5 time periods. Then this observations line is appropriate: `5 1.1 20.5` (read: frequency 5, starting with the observation for unit 1, period 1, and ending with the observation for unit 20, period 5). Then, for instance, you can refer to the observation for unit 2 in period 5 as `2.5`, and that for unit 13 in period 1 as `13.1`.

§ *Grouped by period*: Set the frequency equal to n. In this case if you have observations on 20 units in each of 5 periods, the observations line should be: `20 1.01 5.20` (read: frequency 20, starting with the observation for period 1, unit 01, and ending with the observation for period 5, unit 20). One refers to the observation for unit 2, period 5 as `5.02`.

If you decide to construct a panel data set using a spreadsheet program first, then bring the data into `gretl` as a CSV import, the program will (probably) not at first recognize the special nature of the data. You can fix this by using the command `setobs` (see Chapter 10) or the GUI menu item "Sample, Set frequency, startobs...".

Dummy variables

In a panel study you may wish to construct dummy variables of one or both of the following sorts: (a) dummies as unique identifiers for the cross-sectional units, and (b) dummies as unique identifiers of the time periods. The former may be used to allow the intercept of the regression to differ across the units, the latter to allow the intercept to differ across periods.

You can use two special functions to create such dummies. These are found under the "Data, Add variables" menu in the GUI, or under the `genr` command in script mode or `gretlcli`.

1. If you don't intend to make any conceptual or statistical distinction between cross-sectional and temporal variation in the data you can arrange the rows arbitrarily, but this is probably wasteful of information.

1. "periodic dummies" (script command `genr dummy`). The common use for this command is to create a set of periodic dummy variables up to the data frequency in a time-series study (for instance a set of quarterly dummies for use in seasonal adjustment). But it also works with panel data. Note that the interpretation of the dummies created by this command differs depending on whether the data rows are grouped by unit or by period. If the grouping is by *unit* (frequency T) the resulting variables are *period dummies* and there will be T of them. For instance `dummy_2` will have value 1 in each data row corresponding to a period 2 observation, 0 otherwise. If the grouping is by *period* (frequency n) then n *unit dummies* will be generated: `dummy_2` will have value 1 in each data row associated with cross-sectional unit 2, 0 otherwise.
2. "panel dummies" (script command `genr paneldum`). This creates all the dummies, unit and period, at a stroke. The default presumption is that the data rows are grouped by unit. The unit dummies are named `du_1`, `du_2` and so on, while the period dummies are named `dt_1`, `dt_2`, etc. The `u` (for unit) and `t` (for time) in these names will be wrong if the data rows are grouped by period: to get them right in that setting use `genr paneldum -o` (script mode only).

If a panel data set has the `YEAR` of the observation entered as one of the variables you can create a periodic dummy to pick out a particular year, e.g. `genr dum = (YEAR=1960)`. You can also create periodic dummy variables using the modulus operator, `%`. For instance, to create a dummy with value 1 for the first observation and every thirtieth observation thereafter, 0 otherwise, do

```
genr index genr dum = ((index-1)%30) = 0
```

Using lagged values with panel data

If the time periods are evenly spaced you may want to use lagged values of variables in a panel regression. In this case arranging the data rows by *unit* (stacked time-series) is definitely preferable.

Suppose you create a lag of variable`x1`, using `genr x1_1 = x1(-1)`. The values of this variable will be mostly correct, but at the boundaries of the unit data blocks they will be spurious and unusable. E.g. the value assigned to `x1_1` for observation `2.1` is not the first lag of `x1` at all, but rather the last observation of `x1` for unit 1.

If a lag of this sort is to be included in a regression you must ensure that the first observation from each unit block is dropped. One way to achieve this is to use Weighted Least Squares (`wls`) using an appropriate dummy variable as weight. This dummy (call it `lagdum`) should have value 0 for the observations to be dropped, 1 otherwise. In other words, it is complementary to a dummy variable for period 1. Thus if you have already issued the command `genr dummy` you can now do `genr lagdum = 1 - dummy_1`. If you have used `genr paneldum` you would now say `genr lagdum = 1 - dt_1`. Either way, you can now do

```
wls lagdum y const x1_1 ...
```

to get a pooled regression using the first lag of `x1`, dropping all observations from period 1.

Another option is to use the `smpl` with the `-o` flag and a suitable dummy variable. Here are illustrative commands, assuming the unit data blocks each contain 30 observations and we want to drop the first row of each:

```
(* create index variable *)
genr index
(* create dum = 0 for every 30th obs *)
genr dum = ((index-1)%30) > 0
(* sample based on this dummy *)
smpl -o dum
(* recreate the obs. structure, for 56 units *)
setobs 29 1.01 56.29
```

You can now run regressions on the restricted data set without having to use the `wls` command. If you plan to reuse the restricted data set you may wish to save it using the `store` command (see Chapter 10 below).

Pooled estimation

Having come this far, we can reveal that there is a special purpose estimation command for use with panel data, the “Pooled OLS” option under the Model menu. This command is available only if the data set is recognized as a panel. To take advantage of it, you should specify a model without any dummy variables representing cross-sectional units. The routine presents estimates for straightforward pooled OLS, which treats cross-sectional and time-series variation at par. This model may or may not be appropriate. Under the Tests menu in the model window, you will find an item “panel diagnostics”, which tests pooled OLS against the principal alternatives, the fixed effects and random effects models.

The fixed effects model adds a dummy variable for all but one of the cross-sectional units, allowing the intercept of the regression to vary across the units. An *F*-test for the joint significance of these dummies is presented: if the p-value for this test is small, that counts against the null hypothesis (that the simple pooled model is adequate) and in favor of the fixed effects model.

The random effects model, on the other hand, decomposes the residual variance into two parts, one part specific to the cross-sectional unit or “group” and the other specific to the particular observation. (This estimator can be computed only if the panel is “wide” enough, that is, if the number of cross-sectional units in the data set exceeds the number of parameters to be estimated.) The Breusch–Pagan LM statistic tests the null hypothesis (again, that the pooled OLS estimator is adequate) against the random effects alternative.

It is quite possible that the pooled OLS model is rejected against both of the alternatives, fixed effects and random effects. How, then, to assess the relative merits of the two alternative estimators? The Hausman test (also reported, provided the random effects model can be estimated) addresses this issue. Provided the unit- or group-specific error is uncorrelated with the independent variables, the random effects estimator is more efficient than the fixed effects estimator; otherwise the random effects estimator is inconsistent, in which case the fixed effects estimator is to be preferred. The null hypothesis for the Hausman test is that the group-specific error is not so correlated (and therefore the random effects model is preferable). Thus a low p-value for this tests counts against the random effects model and in favor of fixed effects.

For a rigorous discussion of this topic, see Greene (2000), chapter 14.

Illustration: the Penn World Table

The Penn World Table (homepage at pwt.econ.upenn.edu) is a rich macroeconomic panel dataset, spanning 152 countries over the years 1950–1992. The data are available in `gretl` format; please see the `gretl` data site (this is a free download, although it is not included in the main `gretl` package).

Example 6-1 below opens `pwt56_60_89.gdt`, a subset of the pwt containing data on 120 countries, 1960–89, for 20 variables, with no missing observations (the full data set, which is also supplied in the pwt package for `gretl`, has many missing observations). Total growth of real GDP, 1960–89, is calculated for each country and regressed against the 1960 level of real GDP, to see if there is evidence for “convergence” (i.e. faster growth on the part of countries starting from a low base).

Example 6-1. Use of the Penn World Table

```
open pwt56_60_89.gdt
(* for 1989 (last obs), lag 29 gives 1960, the first obs *)
genr gdp60 = RGDPL(-29)
(* find total growth of real GDP over 30 years *)
genr gdpgro = (RGDPL - gdp60)/gdp60
```

```
(* restrict the sample to a 1989 cross-section *)
smpl -r YEAR=1989
(* Convergence?  Did countries with a lower base grow faster? *)
ols gdpgro const gdp60
(* result: No! Try inverse relationship *)
genr gdp60inv = 1/gdp60
ols gdpgro const gdp60inv
(* No again.  Try dropping Africa? *)
genr afdum = (CCODE = 1)
genr afslope = afdum * gdp60
ols gdpgro const afdum gdp60 afslope
```

Chapter 7. Graphs and plots

Gnuplot graphs

A separate program, namely gnuplot, is called to generate graphs. Gnuplot is a very full-featured graphing program with myriad options. It is available from gnuplot.info (but note that a copy of gnuplot is bundled with the MS Windows version of `gretl`). `gretl` gives you direct access, via a graphical interface, to only a small subset of gnuplot's options but it tries to choose sensible values for you; it also allows you to take complete control over graph details if you wish.

Under MS Windows you can click at the top left corner of a graph window for a pull-down gnuplot menu that lets you choose various things (including copying the graph to the Windows clipboard and sending it to a printer).

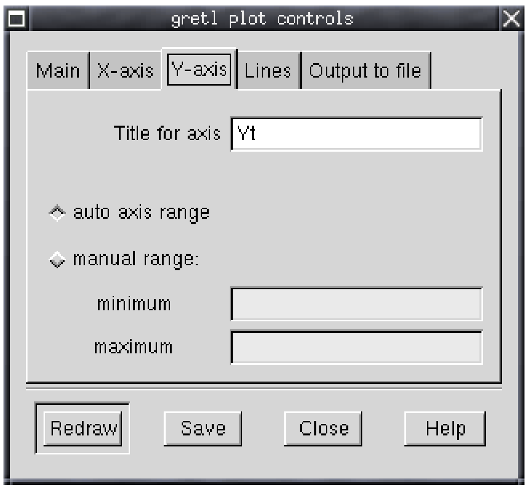
For full control over a graph, follow this procedure:

1. Close the graph window.
2. From the Session menu, choose "Add last graph".
3. In the session icon window, right-click on the new graph icon and choose either "Edit using GUI" or "Edit plot commands".

The "Edit using GUI" item pops up a graphical controller for gnuplot which lets you fine-tune various aspects of the graph. The "Edit plot commands" item opens an editor window containing the actual gnuplot command file for generating the graph: this gives you full control over graph details — if you know something about gnuplot. To find out more, see the gnuplot online manual or gnuplot.info.

See also the entry for `gnuplot` in Chapter 10 below — and the `graph` and `plot` commands for "quick and dirty" ASCII graphs.

Figure 7-1. gretl's gnuplot controller



Title for axis: Yt

auto axis range

manual range:

minimum

maximum

Boxplots

Boxplots are not generated using gnuplot, but rather by `gretl` itself.

These plots (after Tukey and Chambers) display the distribution of a variable. The central box encloses the middle 50 percent of the data, i.e. it is bounded by the first and third quartiles. The "whiskers" extend to the minimum and maximum values. A line is drawn across the box at the median.

In the case of notched boxes, the notch shows the limits of an approximate 90 percent confidence interval. This is obtained by the bootstrap method, which can take a while if the data series is very long.

Clicking the mouse in the boxplots window brings up a menu which enables you to save the plots as encapsulated postscript (EPS) or as a full-page postscript file. Under the X window system you can also save the window as an XPM file; under MS Windows you can copy it to the clipboard as a bitmap. The menu also gives you the option of opening a summary window which displays five-number summaries (minimum, first quartile, median, third quartile, maxmimum), plus a confidence interval for the median if the "notched" option was chosen.

Some details of gretl's boxplots can be controlled via a (plain text) file named `.boxplotrc` which is looked for, in turn, in the current working directory, the user's home directory (corresponding to the environment variable HOME) and the gretl user directory (which is displayed and may be changed under the "File, Preferences, General" menu). Options that can be set in this way are the font to use when producing postscript output (must be a valid generic postscript font name; the default is Helvetica), the size of the font in points (also for postscript output; default is 12), the minimum and maximum for the y-axis range, the width and height of the plot in pixels (default, 560 x 448), whether numerical values should be printed for the quartiles and median (default, don't print them), and whether outliers (points lying beyond 1.5 times the interquartile range from the central box) should be indicated separately (default, no). Here is an example:

```
font = Times-Roman
fontsize = 16
max = 4.0
min = 0
width = 400
height = 448
numbers = %3.2f
outliers = true
```

On the second to last line, the value associated with `numbers` is a "printf" format string as in the C programming language; if specified, this controls the printing of the median and quartiles next to the boxplot, if no `numbers` entry is given these values are not printed. In the example, the values will be printed to a width of 3 digits, with 2 digits of precision following the decimal point.

Not all of the options need be specified, and the order doesn't matter. Lines not matching the pattern "key = value" are ignored, as are lines that begin with the hash mark, `#`.

After each variable specified in the boxplot command, a parenthesized boolean expression may be added, to limit the sample for the variable in question. A space must be inserted between the variable name or number and the expression. Suppose you have salary figures for men and women, and you have a dummy variable `GENDER` with value 1 for men and 0 for women. In that case you could draw comparative boxplots with the following line in the boxplots dialog:

```
      salary (GENDER=1) salary (GENDER=0)
```

Chapter 8. Loop constructs

Monte Carlo simulations

gretl offers (limited) support for Monte Carlo simulations. To do such work you should either use the GUI client program in "script mode" (see the Section called *Command scripts* in Chapter 4 above), or use the command-line client. The command loop opens a special mode in which the program accepts commands to be repeated a specified number of times. Within such a loop, only six commands can be used: genr, ols, print, smpl, store and summary. With genr and ols it is possible to do quite a lot. You exit the mode of entering loop commands with endloop: at this point the stacked commands are executed. Loops cannot be nested.

The ols command gives special output in a loop context: the results from each individual regression are not printed, but rather you get a printout of (a) the mean value of each estimated coefficient across all the repetitions, (b) the standard deviation of those coefficient estimates, (c) the mean value of the estimated standard error for each coefficient, and (d) the standard deviation of the estimated standard errors. This makes sense only if there is some random input at each step.

The print command also behaves differently in the context of a loop. It prints the mean and standard deviation of the variable, across the repetitions of the loop. It is intended for use with variables that have a single value at each iteration, for example the error sum of squares from a regression.

The store command (use only one of these per loop) writes out the values of the specified variables, from each time round the loop, to the specified file. Thus it keeps a complete record of the variables. This data file can then be read into the program and analysed.

A simple example of Monte Carlo loop code is shown in Example 8-1.

Example 8-1. Simple Monte Carlo loop

```
(* create a blank data set with series length 50 *)
nulldata 50
genr x = uniform()
(* open a loop, to be repeated 100 times *)
loop 100
  genr u = normal()
  (* construct the dependent variable *)
  genr y = 10*x + 20*u
  (* run OLS regression *)
  ols y const x
  (* grab the R-squared value from the regression *)
  genr r2 = $rsq
  (* arrange for statistics on R-squared to be printed *)
  print r2
  (* save the individual coefficient estimates *)
  genr a = coeff(const)
  genr b = coeff(x)
  (* and print them to file *)
  store foo.gdt a b
endloop
```

This loop will print out summary statistics for the 'a' and 'b' estimates across the 100 repetitions, and also for the R^2 values for the 100 regressions. After running the loop, foo.gdt, which contains the individual coefficient estimates from all the runs, can be opened in gretl to examine the frequency distribution of the estimates in detail. Please note that while comment lines are permitted in a loop (as shown in the example), they cannot run over more than one line.

The command nulldata is useful for Monte Carlo work. Instead of opening a "real" data set, nulldata 50 (for instance) opens an empty data set, with only a constant, with a series length of 50. Constructed variables can then be added using the genr command.

See the `seed` command in Chapter 10 for information on generating repeatable pseudo-random series.

Iterated least squares

A second form of loop structure is designed primarily for carrying out iterated least squares. Greene (2000, ch. 11) shows how this method can be used to estimate nonlinear models.

To open this sort of loop you need to specify a *condition* rather than an unconditional number of times to iterate. This should take the form of the keyword `while` followed by an inequality: the left-hand term should be the name of a variable that is already defined; the right-hand side may be either a numerical constant or the name of another predefined variable. For example,

```
loop while essdiff > .00001
```

Execution of the commands within the loop (i.e. until `endloop` is encountered) will continue so long as the specified condition evaluates as true.

I assume that if you specify a "number of times" loop you are probably doing a Monte Carlo analysis, and hence you're not interested in the results from each individual iteration but rather the moments of certain variables over the ensemble of iterations. On the other hand, if you specify a "while" loop you're probably doing something like iterated least squares, and so you'd like to see the final result — as well, perhaps, as the value of some variable(s) (e.g. the error sum of squares from a regression) from each time round the loop. The behavior of the `print` and `ols` commands are tailored to this assumption. In a "while" loop `print` behaves as usual; thus you get a printout of the specified variable(s) from each iteration. The `ols` command prints out the results from the final estimation.

Example 8-2 uses a "while" loop to replicate the estimation of a nonlinear consumption function of the form $C = \alpha + \beta Y^{\gamma} + \epsilon$ as presented in Greene (2000, Example 11.3). This script is included in the `gretl` distribution under the name `greene11_3.inp`; you can find it in `gretl` under the menu item "File, Open command file, practice file, Greene...".

Example 8-2. Nonlinear consumption function

```
open greene11_3.gdt
(* run initial OLS *)
ols C 0 Y
genr essbak = $ess
genr essdiff = 1
genr b0 = coeff(Y)
genr gamma0 = 1
(* form the linearized variables *)
genr C0 = C + gamma0 * b0 * Y^gamma0 * log(Y)
genr x1 = Y^gamma0
genr x2 = b0 * Y^gamma0 * log(Y)
(* iterate OLS till the error sum of squares converges *)
loop while essdiff > .00001
  ols C0 0 x1 x2 -o
  genr b0 = coeff(x1)
  genr gamma0 = coeff(x2)
  genr C0 = C + gamma0 * b0 * Y^gamma0 * log(Y)
  genr x1 = Y^gamma0 genr x2 = b0 * Y^gamma0 * log(Y)
  genr ess = $ess genr
  essdiff = abs(ess - essbak)/essbak
  genr essbak = ess
endloop
(* print parameter estimates using their "proper names" *)
genr alpha = coeff(0)
genr beta = coeff(x1)
genr gamma = coeff(x2)
print alpha beta gamma
```

Indexed loop

The third form of loop construct offered in `gretl` is an indexed loop, using the internal variable `i`. You specify starting and ending values for `i`, which is incremented by one each time round the loop. The syntax looks like this: `loop i=1..20`. Example 8-3 shows one use of this construct. We have a panel data set, with observations on a number of hospitals for the years 1991 to 2000. We restrict the sample to each of these years in turn and print cross-sectional summary statistics for variables 1 through 4.

Example 8-3. Indexed loop example

```
open hospitals.gdt
loop for i=1991..2000
  smpl -r (year=i)
  summary 1 2 3 4
endloop
```

Chapter 9. Options, arguments and path-searching

gretl

`gretl` (under MS Windows, `gretlw32.exe`)[1]

— Opens the program and waits for user input.

`gretl` *`datafile`*

— Starts the program with the specified datafile in its workspace. The data file may be in native `gretl` format, CSV format, or BOX1 format (see Chapter 5 above). The program will try to detect the format of the file and treat it appropriately. See also the Section called *Path searching* below for path-searching behavior.

`gretl --help` (or `gretl -h`)

— Print a brief summary of usage and exit.

`gretl --version` (or `gretl -v`)

— Print version identification for the program and exit

`gretl --run` *`scriptfile`* (or `gretl -r` *`scriptfile`*)

— Start the program and open a window displaying the specified script file, ready to run. See the Section called *Path searching* below for path-searching behavior.

`gretl --db` *`database`* (or `gretl -d` *`database`*)

— Start the program and open a window displaying the specified database. If the database files (the `.bin` file and its accompanying `.idx` file — see the Section called *Binary databases* in Chapter 5) is not in the default system database directory, you must specify the full path.

`gretl --webdb` *`database`* (or `gretl -w` *`database`*)

— Start the program and open a window displaying the specified remote (web) database. The name of the database should be given without the `.bin` suffix.

Various things in `gretl` are configurable under the “File, Preferences” menu.

§ The base directory for `gretl`’s shared files.

§ The command to launch `gnuplot`.

§ The user’s base directory for `gretl`-related files.

§ The command to launch GNU R (see Appendix C).

§ The directory in which to start looking for native `gretl` databases.

§ The directory in which to start looking for RATS 4 databases.

§ The IP number of the `gretl` database server to access.

§ The IP number and port number of the HTTP proxy server to use when contacting the database server, if applicable (if you’re behind a firewall).

§ The calculator and editor programs to launch from the toolbar.

§ The monospaced font to be used in `gretl` screen output.

There are also some check boxes. Checking the “expert” box quells some warnings that are otherwise issued. Checking “Tell me about gretl updates” makes `gretl` attempt to query the update server at start-up. Unchecking “Show gretl toolbar” turns the icon toolbar off.

Finally, there are some binary choices: Under the “Open/Save path” tab you can set where `gretl` looks by default when you go to open or save a file — either the `gretl` user directory or the current working directory. Under the “Data files” tab you can set the default

1. On unix-type systems, a “wrapper” script named `gretl` is installed. This script checks whether the `DISPLAY` environment variable is set; if so, it launches the GUI program, `gretl_x11`, and if not it launches the command-line program, `gretlcli`.

filename suffix for data files. The standard suffix is `.gdt` but if you wish you can set this to `.dat`, which was standard in earlier versions of the program.

Settings chosen in this way are handled differently depending on the context. Under MS Windows they are stored in the Windows registry. Under the `gnome` desktop they are stored in `.gnome/gretl` in the user's home directory. Otherwise they are stored in a file named `.gretlrc` in the user's home directory.

gretlcli

`gretlcli`

— Opens the program and waits for user input.

`gretlcli` *`datafile`*

— Starts the program with the specified datafile in its workspace. The data file may be in native `gretl` format, CSV format, or BOX1 format (see Chapter 5). The program will try to detect the format of the file and treat it appropriately. See also the Section called *Path searching* for path-searching behavior.

`gretlcli --help` (or `gretlcli -h`)

— Prints a brief summary of usage.

`gretlcli --version` (or `gretlcli -v`)

— Prints version identification for the program.

`gretlcli --pvalue` (or `gretlcli -p`)

— Starts the program in a mode in which you can interactively determine p-values for various common statistics.

`gretlcli --run` *`scriptfile`* (or `gretlcli -r` *`scriptfile`*)

— Execute the commands in *`scriptfile`* then hand over input to the command line. See the Section called *Path searching* for path-searching behavior.

`gretlcli --batch` *`scriptfile`* (or `gretlcli -b` *`scriptfile`*)

— Execute the commands in *`scriptfile`* then exit. When using this option you will probably want to redirect output to a file. See the Section called *Path searching* for path-searching behavior.

When using the `--run` and `--batch` options, the script file in question must call for a data file to be opened. This can be done using the `open` command within the script. For backward compatibility with Ramanathan's original `ESL` program another mechanism is offered (`ESL` doesn't have the `open` command). A line of the form:

```
 (* ! myfile.gdt *)
```

will (a) cause `gretlcli` to load `myfile.gdt`, but will (b) be ignored as a comment by the original `ESL`. Note the specification carefully: There is exactly one space between the begin comment marker, `(*`, and the `!`; there is exactly one space between the `!` and the name of the data file.

One further kludge enables `gretl` and `gretlcli` to get datafile information from the `ESL` "practice files" included with the `gretl` package. A typical practice file begins like this:

```
(* PS4.1, using data file DATA4-1, for reproducing Table 4.2 *)
```

This algorithm is used: if an input line begins with the comment marker, search it for the string `DATA` (upper case). If this is found, extract the string from the `D` up to the next space or comma, put it into lower case, and treat it as the name of a data file to be opened.

Path searching

When the name of a data file or script file is supplied to `gretl` or `gretlcli` on the command line (see the Section called *gretl* and the Section called *gretlcli*), the file is looked for as follows:

1. “As is”. That is, in the current working directory or, if a full path is specified, at the specified location.
2. In the user’s gretl directory (see Table 9-1 for the default values).
3. In any immediate sub-directory of the user’s gretl directory.
4. In the case of a data file, search continues with the main `gretl` data directory. In the case of a script file, the search proceeds to the system script directory. See Table 9-1 for the default settings.
5. In the case of data files the search then proceeds to all immediate sub-directories of the main data directory.

Table 9-1. Default path settings

	Linux/unix	**MS Windows**
User directory	`$HOME/gretl`	`PREFIX\gretl\user`
System data directory	`PREFIX/share/gretl/data`	`PREFIX\gretl\data`
System script directory	`PREFIX/share/gretl/scripts`	`PREFIX\gretl\scripts`

Note: `PREFIX` denotes the base directory chosen at the time `gretl` is installed.

Thus it is not necessary to specify the full path for a data or script file unless you wish to override the automatic searching mechanism. (This also applies within `gretlcli`, when you supply a filename as an argument to the `open` or `run` commands.)

When a command script contains an instruction to open a data file, the search order for the data file is as stated above, except that the directory containing the script is also searched, immediately after trying to find the data file “as is”.

MS Windows

Under MS Windows configuration information for `gretl` and `gretlcli` is stored in the Windows registry. A suitable set of registry entries is created when `gretl` is first installed, and the settings can be changed under `gretl`’s “File, Preferences” menu. In case anyone needs to make manual adjustments to this information, the entries can be found (using the standard Windows program `regedit.exe`) under `Software\gretl` in `HKEY_CLASSES_ROOT` (the main `gretl` directory and the command to invoke `gnuplot`) and `HKEY_CURRENT_USER` (all other configurable variables).

Chapter 10. Command Reference

Introduction

The commands defined below may be executed in the command-line client program. They may also be placed in a "script" file for execution in the GUI, or entered using the latter's "console mode". In most cases the syntax given below also applies when you are presented with a line to type in a dialog box in the GUI (but see also `gretl`'s online help), except that you should *not* type the initial command word — it is implicit from the context. One other difference is that you should not type the `-o` flag for regression commands in GUI dialog boxes: there is a menu item for displaying the coefficient variance–covariance matrix (which is the effect of `-o` in regression commands).

The following conventions are used below:

§ A `typewriter font` is used for material that you would type directly, and also for internal names of variables.

§ Terms in *`italics`* are place-holders: you should substitute something specific, e.g. you might type `income` in place of the generic *`xvar`*.

§ `[ -o ]` means that the flag `-o` is optional: you may type it or not (but in any case don't type the brackets).

§ The phrase "estimation command" means any one of `ols`, `hilu`, `corc`, `ar`, `arch`, `hsk`, `tsls`, `wls`, `hccm`, `add`, `omit`.

Section and Chapter references below are to Ramanathan (2002).

gretl commands

add

Argument:	*`varlist`* `[ -o ]`
Examples:	`add 5 7 9`
	`add xx yy zz -o`

Must be invoked after an estimation command. The variables in *`varlist`* are added to the previous model and the new model estimated. If more than one variable is added, the F statistic for the added variables will be printed (for the OLS procedure only) along with its p-value. A p-value below 0.05 means that the coefficients are jointly significant at the 5 percent level. A number of internal variables may be retrieved using the `genr` command, provided `genr` is invoked directly after this command. The `-o` flag causes the coefficient variance–covariance matrix to be printed.

addto

Arguments:	*`modelID varlist`*
Example:	`addto 2 5 7 9`

Works like the `add` command, except that you specify a previous model (using its ID number, which is printed at the start of the model output) to take as the base for adding variables. The example above adds variables number 5, 7 and 9 to Model 2.

adf

Arguments:	*`order varname`*
Example:	`adf 2 x1`

Computes statistics for two Dickey–Fuller tests. In each case the null hypothesis is that the variable in question exhibits a unit root. The first is a *t*-test based on the model

$$(1 - L)x_t = m + gx_{t-1} + \epsilon_t$$

The null hypothesis is that $g = 0$. The second (augmented) test proceeds by estimating an unrestricted regression (with regressors a constant, a time trend, the first lag of the variable, and *order* lags of the first difference) and a restricted version (dropping the time trend and the first lag). The test statistic is

$$F_{2,T-k} = \frac{(ESS_r - ESS_u)/2}{ESS_u/(T-k)}$$

where T is the sample size, k the number of parameters in the unrestricted model, and the subscripts u and r denote the unrestricted and restricted models respectively. Note that the critical values for these statistics are not the usual ones; a p-value range is printed, when it can be determined.

ar

Arguments:	*lags* ; *depvar indepvars* [-o]
Example:	`ar 1 3 4 ; y 0 x1 x2 x3`

Computes parameter estimates using the generalized Cochrane–Orcutt iterative procedure (see Section 9.5 of Ramanathan). Iteration is terminated when successive error sums of squares do not differ by more than 0.005 percent or after 20 iterations. *lags* is a list of lags in the residuals, terminated by a semicolon. In the above example, the error term is specified as

$$u_t = \rho_1 u_{t-1} + \rho_3 u_{t-3} + \rho_4 u_{t-4} + e_t$$

depvar is the dependent variable and *indepvars* is the list of independent variables. If the `-o` flag is given, the covariance matrix of regression coefficients is printed. Residuals of the transformed regression are stored under the name `uhat`, which can be retrieved by `genr`. A number of other internal variables may be retrieved using the `genr` command, provided `genr` is invoked after this command.

arch

Arguments:	*order depvar indepvars* [-o]
Example:	`arch 4 y 0 x1 x2 x3`

Tests the model for ARCH (Autoregressive Conditional Heteroskedasticity) of the lag order specified in *order*, which must be an integer. If the LM test statistic has p-value below 0.10, then ARCH estimation is also carried out. If the predicted variance of any observation in the auxiliary regression is not positive, then the corresponding squared residual is used instead. Weighted least square estimation is then performed on the original model. The flag `-o` calls for the coefficient covariance matrix.

chow

Argument:	*obs*
Examples:	`chow 25`
	`chow 1988.1`

Must follow an OLS regression. Creates a dummy variable which equals 1 from the split point specified by *obs* to the end of the sample, 0 otherwise, and also creates interaction terms between this dummy and the original independent variables. An augmented regression is run including these terms and an F statistic is calculated, taking the augmented regression as the unrestricted and the original as restricted. This statistic is appropriate for testing the null hypothesis of no structural break at the given split point.

coint

Arguments:	*order depvar indepvar*
Examples:	`coint 2 y x`
	`coint 4 y x1 x2`

Carries out Augmented Dickey–Fuller tests on the null hypothesis that each of the variables listed has a unit root, using the given lag order. The cointegrating regression is estimated, and an ADF test is run on the residuals from this regression. The Durbin–Watson statistic for the cointegrating regression is also given. Note that none of these test statistics can be referred to the usual statistical tables.

corc

Arguments:	*depvar indepvars* [`-o`]
Examples:	`corc 1 0 2 4 6 7`
	`corc -o 1 0 2 4 6 7`
	`corc y 0 x1 x2 x3`
	`corc -o y 0 x1 x2 x3`

Computes parameter estimates using the Cochrane–Orcutt iterative procedure (see Section 9.4 of Ramanathan) with *depvar* as the dependent variable and *indepvars* as the list of independent variables. Iteration is terminated when successive estimates of the autocorrelation coefficient do not differ by more than 0.001 or after 20 iterations. If the `-o` flag is given, the covariance matrix of regression coefficients is printed. Residuals of this transformed regression are stored under the name `uhat`. Various internal variables may be retrieved using the `genr` command, provided `genr` is invoked immediately after this command.

corr

Argument:	[*varlist*]
Examples:	`corr 1 3 5`
	`corr y x1 x2 x3`

Prints the pairwise correlation coefficients for the variables in *varlist*, or for all variables in the data set if *varlist* is not given.

corrgm

Arguments:	*variable* [*maxlag*]

Prints the values of the autocorrelation function for the *variable* specified (either by name or number). See Ramanathan, Section 11.7. It is thus $\rho(u_t, u_{t-s})$ where u_t is the tth observation of the variable u and s is the number of lags.

The partial autocorrelations are also shown: these are net of the effects of intervening lags. The command also graphs the correlogram and prints the Box-Pierce Q statistic for testing the null hypothesis that the series is "white noise". This is asymptotically distributed as chi-square with degrees of freedom equal to the number of lags used.

If an (optional) integer `maxlag` value is supplied the length of the correlogram is limited to at most that number of lags, otherwise the length is determined automatically.

criteria

Arguments:	*`ess T k`*
Example:	`criteria 23.45 45 8`

Computes the model selection statistics (see Ramanathan, Section 4.3), given `ess` (error sum of squares), the number of observations (T), and the number of coefficients (k). T, k, and `ess` may be numerical values or names of previously defined variables.

cusum

Must follow the estimation of a model via OLS. Performs the CUSUM test for parameter stability. A series of (scaled) one-step ahead forecast errors is obtained by running a series of regressions: the first regression uses the first k observations and is used to generate a prediction of the dependent variable at observation at observation $k + 1$; the second uses the first $k + 1$ observations and generates a prediction for observation $k + 2$, and so on (where k is the number of parameters in the original model). The cumulated sum of the scaled forecast errors is printed and graphed. The null hypothesis of parameter stability is rejected at the 5 percent significance level if the cumulated sum strays outside of the 95 percent confidence band.

The Harvey–Collier t-statistic for testing the null hypothesis of parameter stability is also quoted. See Chapter 7 of Greene's *Econometric Analysis* for details.

delete

Removes the last (highest numbered) variable from the current data set. *Use with caution*: no confirmation is asked. Can be useful for getting rid of temporary dummy variables. There is no provision for deleting any but the last variable.

diff

Argument:	*`varlist`*

The first difference of each variable in `varlist` is obtained and the result stored in a new variable with the prefix `d_`. Thus `diff x y` creates the new variables `d_x = x(t) - x(t-1)` and `d_y = y(t) - y(t-1)`.

endloop

Terminates a simulation loop. See `loop`.

eqnprint

Must follow the estimation of a model via OLS. Prints the estimates in the form of a LaTeX equation, to a file with a name of the form `equation_N.tex`, where `N` is the number of models estimated to date in the current session. This can be incorporated in a LaTeX document. See also `tabprint`.

fcast

Argument:	[*`startobs endobs`*] *`newvarname`*
Examples:	`fcast 1997.1 1999.4 f1`

```
fcast f2
```

Must follow an estimation command. Forecasts are generated for the specified range (or the largest possible range if no *startobs* and *endobs* are given) and the values saved as *newvarname*, which can be printed, graphed, or plotted. The right-hand side variables are those in the original model. There is no provision to substitute other variables. If an autoregressive error process is specified (for `hilu`, `corc`, and `ar`) the forecast is conditional one step ahead and incorporates the error process.

fcasterr

Arguments: *startobs endobs* `[ -o ]`

After estimating an OLS model which includes a constant and at least one independent variable (these restrictions may be relaxed at some point) you can use this command to print out fitted values over the specified observation range, along with the estimated standard errors of those predictions and 95 percent confidence intervals. If the `-o` flag is given the results will also be displayed using gnuplot. The augmented regression method of Salkever (1976) is used to generate the forecast standard errors.

fit

The `fit` command (must follow an estimation command) is a shortcut for the `fcast` command. It generates fitted values, in a series called `autofit`, for the current sample, based on the last regression. In the case of time-series models, `fit` also pops up a gnuplot graph of fitted and actual values of the dependent variable against time.

freq

Argument: *var*

Prints the frequency distribution for *var* (given by name or number) along with a chi-square test for normality. In interactive mode a gnuplot graph of the distribution is displayed.

genr

Argument: *newvar* = *formula*

Creates new variables, usually through transformations of existing variables. See also `diff`, `logs`, `lags`, `ldiff`, `multiply` and `square` for shortcuts.

Supported *arithmetical operators* are, in order of precedence: `^` (exponentiation); `*`, `/` and `%` (modulus or remainder); `+` and `-`.

The available *Boolean operators* are (again, in order of precedence): `!` (negation), `&` (logical AND), `|` (logical OR), `>`, `<`, `=` and `!=` (not equal to). The Boolean operators can be used in constructing dummy variables: for instance `(x > 10)` returns 1 if `x` > 10, 0 otherwise. Supported *functions* fall into these groups:

§ Standard math functions: `abs`, `cos`, `exp`, `int` (integer part), `ln` (natural log: `log` is a synonym), `sin`, `sqrt`.

§ Statistical functions: `mean` (arithmetic mean), `median`, `var` (variance) `sd` (standard deviation), `sum`, `cov` (covariance), `corr` (correlation coefficient), `pvalue`

§ Time-series functions: `lag`, `lead`, `diff` (first difference), `ldiff` (log-difference, or first difference of natural logs).

- § Miscellaneous: `cum` (cumulate), `sort`, `uniform`, `normal`, `misszero` (replace the missing observation code in a given series with zeros), `zeromiss` (the inverse operation to `misszero`).

All of the above functions with the exception of `cov`, `corr`, `pvalue`, `uniform` and `normal` take as their single argument either the name of a variable (note that you can't refer to variables by their ID numbers in a `genr` command) or a composite expression that evaluates to a variable (e.g. `ln((x1+x2)/2)`). `cov` and `corr` both require two arguments, and return respectively the covariance and the correlation coefficient between two named variables. The `pvalue` function takes the same arguments as the `pvalue` command (see below), but in this context commas should be placed between the arguments. `uniform()` and `normal()`, which do not take arguments, return pseudo-random series drawn from the uniform (0–100) and standard normal distributions respectively (see also the `seed` command). Uniform series are generated using the C library function `rand()`; for normal series the method of Box and Muller (1958) is used.

Besides the operators and functions just noted there are some special uses of `genr`:

- § `genr time` creates a time trend variable (1,2,3,...) called `time`. `genr index` does the same thing except that the variable is called `index`.
- § `genr dummy` creates dummy variables up to the periodicity of the data. E.g. in the case of quarterly data (periodicity 4), the program creates `dummy_1` = 1 for first quarter and 0 in other quarters, `dummy_2` = 1 for the second quarter and 0 in other quarters, and so on.
- § `genr paneldum` creates a set of special dummy variables for use with a panel data set — see Chapter 6 above.
- § Various internal variables defined in the course of running a regression can be retrieved using `genr`, as follows:

`$ess`	error sum of squares
`$rsq`	unadjusted $R2$
`$nobs`	number of observations
`$df`	degrees of freedom
`$trsq`	$TR2$ (sample size times $R2$)
`$sigma`	standard error of residuals
`$lnl`	log-likelihood (logit and probit models)
`$sigma`	standard error of residuals
coeff(*var*)	estimated coefficient for variable *var*
stderr(*var*)	estimated standard error for variable *var*
rho(*i*)	*i*th order autoregressive coefficient for residuals
vcv(*var1*,*var2*)	covariance between coefficients for named variables *var1* and *var2*

Note: In the command-line program, `genr` commands that retrieve model-related data always reference the model that was estimated most recently. This is also true in the GUI program, if one uses `genr` in the "gretl console" or enters a formula using the "Define new variable" option under the Variable menu in the main window. With the GUI, however, you have the option of retrieving data from any model currently displayed in a window (whether or not it's the most recent model). You do this under the "Model data" menu in the model's window.

Table 10-1 gives several examples of uses of `genr` with explanatory notes; here are a couple of tips on dummy variables:

- § Suppose `x` is coded with values 1, 2, or 3 and you want three dummy variables, `d1` = 1 if `x` = 1, 0 otherwise, `d2` = 1 if `x` = 2, and so on. To create these, use the commands:

```
genr d1 = (x=1)
genr d2 = (x=2)
genr d3 = (x=3)
```

§ To create `z = max(x,y)` do

```
genr d = x>y
genr z = (x*d)+(y*(1-d))
```

Table 10-1. Examples of use of genr command

Command	Comment
`genr y = x1^3`	`x1` cubed
`genr y = ln((x1+x2)/x3)`	
`genr z = x>y`	sets `z(t)` to 1 if `x(t) > y(t)` else to 0
`genr y = x(-2)`	`x lagged 2 periods`
`genr y = x(2)`	`x led 2 periods`
`genr y = diff(x)`	`y(t) = x(t) - x(t-1)`
`genr y = ldiff(x)`	`y(t) = log x(t) - log x(t-1)`, the instantaneous rate of growth of `x`
`genr y = sort(x)`	sorts `x` in increasing order and stores in `y`
`genr y = -sort(-x)`	sort `x` in decreasing order
`genr y = int(x)`	truncate `x` and store its integer value as `y`
`genr y = abs(x)`	store the absolute values of `x`
`genr y = sum(x)`	sum `x` values excluding missing -999 entries
`genr y = cum(x)`	cumulation: $y_t = \sum_{\tau=1}^{t} x_\tau$
`genr aa = $ess`	set `aa` equal to the Error Sum of Squares from last regression
`genr x = coeff(sqft)`	grab the estimated coefficient on the variable `sqft` from the last regression
`genr rho4 = rho(4)`	grab the 4th-order autoregressive coefficient from the last model (presumes an `ar` model)
`genr cvx1x2 = vcv(x1, x2)`	grab the estimated coefficient covariance of vars `x1` and `x2` from the last model
`genr foo = uniform()/100`	uniform pseudo-random variable in range 0–1
`genr bar = 3 * normal()`	normal pseudo-random variable, $\mu = 0$, $\sigma = 3$

gnuplot

Arguments: *`yvars xvar`* `[ -o | -m ]`

`gnuplot -z` *`yvar xvar dummy`*

In the first case the *`yvars`* are graphed against *`xvar`*. If the flag `-o` is supplied the plot will use lines; if the flag `-m` is given the plot uses impulses (vertical lines); otherwise points will be used.

In the second case `yvar` is graphed against `xvar` with the points shown in different colors depending on whether the value of *dummy* is 1 or 0.

To make a time-series graph, do `gnuplot` *yvars* `time`. If no variable named `time` already exists, then it will be generated automatically. Special dummy variables will be created for plotting quarterly and monthly data.

In interactive mode the result is piped to gnuplot for display. In batch mode a plot file named `gpttmp01.plt` is written. (With subsequent uses of `gnuplot` similar files are created, with the number in the file name incremented.) The plots can be generated later using the command `gnuplot gpttmp.plt`. (Under MS Windows, start `wgnuplot` and open the file `gpttmp01.plt`.) To gain control over the details of the plot, edit the `.plt` file.

graph

Arguments:	*var1 var2* `[ -o ]`
	`graph` *var1 var2 var3*

ASCII graphics. In the first example, variable *var1* (which may be a name or a number) is graphed (y-axis) against *var2* (x-axis) using ASCII symbols. `-o` flag will graph with 40 rows and 60 columns. Without it, the graph will be 20 by 60 (for screen output). In the second example, both *var1* and *var2* will be graphed (on y-axis) against *var3*. This is useful to graph observed and predicted values against time. See also the `gnuplot` command.

hausman

This test is available only after estimating a model using the `pooled` command (see also `panel` and `setobs`). It tests the simple pooled model against the principal alternatives, the fixed effects and random effects models.

The fixed effects model adds a dummy variable for all but one of the cross-sectional units, allowing the intercept of the regression to vary across the units. An *F*-test for the joint significance of these dummies is presented. The random effects model decomposes the residual variance into two parts, one part specific to the cross-sectional unit and the other specific to the particular observation. (This estimator can be computed only if the number of cross-sectional units in the data set exceeds the number of parameters to be estimated.) The Breusch–Pagan LM statistic tests the null hypothesis (that the pooled OLS estimator is adequate) against the random effects alternative.

The pooled OLS model may be rejected against both of the alternatives, fixed effects and random effects. Provided the unit- or group-specific error is uncorrelated with the independent variables, the random effects estimator is more efficient than the fixed effects estimator; otherwise the random effects estimator is inconsistent and the fixed effects estimator is to be preferred. The null hypothesis for the Hausman test is that the group-specific error is not so correlated (and therefore the random effects model is preferable). A low p-value for this test counts against the random effects model and in favor of fixed effects.

hccm

Arguments:	*depvar indepvars* `[ -o ]`

Presents OLS estimates with the heteroskedasticity consistent covariance matrix estimates for the standard errors of regression coefficients using MacKinnon and White (1985) “jack-knife” estimates (see Ramanathan, Section 8.3). The coefficient covariance matrix is printed if the `-o` flag is given.

help

Gives a list of available commands. `help` *`command`* describes *`command`* (e.g. `help smpl`). You can type `man` instead of `help` if you like.

hilu

Arguments:	*`depvar indepvars`* `[ -o ]`
Examples:	`hilu 1 0 2 4 6 7`
	`hilu -o y 0 x1 x2 x3`

Computes parameter estimates using the Hildreth–Lu search procedure (fine tuned by the CORC procedure) with *`depvar`* as the dependent variable and *`indepvars`* as the list of independent variables. The error sum of squares of the transformed model is graphed against the value of rho from -0.99 to 0.99. If the `-o` flag is present, the covariance matrix of regression coefficients will be printed. Residuals of this transformed regression are stored under the name `uhat`.

hsk

Arguments:	*`depvar indepvars`* `[ -o ]`

Prints heteroskedasticity corrected estimates (see Ramanathan, ch. 8) and associated statistics. The auxiliary regression predicts the log of the square of residuals (using squares of independent variables but not their cross products) from which weighted least squares estimates are obtained. If the `-o` flag is given, the covariance matrix of regression coefficients is printed. Various internal variables may be retrieved using the `genr` command, provided `genr` is invoked immediately after this command.

import

Argument:	*`filename`* `[ -o ]`

Without the `-o` flag, brings in data from a comma-separated values (CSV) format file, such as can easily be written from a spreadsheet program. The file should have variable names on the first line and a rectangular data matrix on the remaining lines. Variables should be arranged "by observation" (one column per variable; each row represents an observation). See Chapter 5 for details.

With the `-o` flag, reads a data file in BOX1 format, as can be obtained using the Data Extraction Service of the US Bureau of the Census.

info

`info` prints out any information contained in the header file corresponding to the current datafile. (This information must be enclosed between `(*` and `*)`, these markers being placed on separate lines.)

labels

Prints out the informative labels for any variables that have been generated using `genr`, and any labels added to the data set via the GUI.

lags

Argument:	*`varlist`*

Creates new variables which are lagged values of each of the variables in varlist. The number of lagged variables equals the periodicity. For example, if the periodicity is 4 (quarterly), the command `lags x y` creates `x_1 = x(t-1)`, `x_2 = x(t-2)`, `x_3 = x(t-3)` and `x_4 x(t-4)`. Similarly for `y`. These variables must be referred to in the exact form, that is, with the underscore.

ldiff

Argument: *varlist*

The first difference of the natural log of each variable in varlist is obtained and the result stored in a new variable with the prefix `ld_`. Thus `ldiff x y` creates the new variables `ld_x` = $\ln(x_t) - \ln(x_{t-1})$ and `ld_y` = $\ln(y_t) - \ln(y_{t-1})$.

lmtest

Must immediately follow an `ols` command. Prints the Lagrange multiplier test statistics (and associated p-values) for nonlinearity and heteroskedasticity (White's test) or, if the `-o` flag is given, the LMF test statistic for serial correlation up to the periodicity (see Kiviet, 1986). The corresponding auxiliary regression coefficients are also printed out. See Ramanathan, Chapters 7, 8, and 9 for details. Only the squared independent variables are used and not their cross products. (If the internal creation of squares causes exact multicollinearity, LM test statistics cannot be obtained.)

logit

Arguments: *depvar indepvars*

Binomial logit regression. The dependent variable should be a binary variable. Maximum likelihood estimates of the coefficients on *indepvars* are obtained via the EM or Expectation–Maximization method (see Ruud, 2000, ch. 27). As the model is nonlinear the slopes depend on the values of the independent variables: the reported slopes are evaluated at the means of those variables. The chi-square statistic tests the null hypothesis that all coefficients are zero apart from the constant.

If you want to use logit for analysis of proportions (where the dependent variable is the proportion of cases having a certain characteristic, at each observation, rather than a 1 or 0 variable indicating whether the characteristic is present or not) you should not use the `logit` command, but rather construct the logit variable (e.g. `genr lgt_p = log(p/(1 - p))`) and use this as the dependent variable in an OLS regression. See Ramanathan, ch. 12.

logs

Agument: *varlist*

The natural log of each of the variables in varlist is obtained and the result stored in a new variable with the prefix `l_` which is "el" underscore. `logs x y` creates the new variables `l_x` = ln(x) and `l_y` = ln(y).

loop

Usage:	`loop` *number_of_times*
	`loop while` *condition*
	`loop for i=` *start*`..` *end*
Examples:	`loop 1000`
	`loop while essdiff > .00001`

```
loop for i=1991..2000
```

Opens a special mode in which the program accepts commands to be repeated either a specified number of times, or so long as a specified condition holds true, or for successive integer values of the (internal) index variable `i`. Within a loop, only six commands can be used: `genr`, `ols`, `print`, `smpl`, `store` and `summary` (store can't be used in a "while" loop). You exit the mode of entering loop commands with `endloop`: at this point the stacked commands are executed. Loops cannot be nested. See Chapter 8 for details.

meantest

Arguments: *`var1 var2`* `[ -o ]`

Calculates the *t* statistic for the null hypothesis that the population means are equal for the variables *`var1`* and *`var2`*, and shows its p-value. Without the `-o` flag, the statistic is computed on the assumption that the variances are equal for the two variables; with the `-o` flag the variances are assumed to be unequal. (The flag will make a difference only if there are different numbers of non-missing observations for the two variables.)

multiply

Arguments: *`x suffix varlist`*

Examples: `multiply invpop pc 3 4 5 6`

`multiply 1000 big x1 x2 x3`

The variables in *`varlist`* (referenced by name or number) are multiplied by *`x`*, which may be either a numerical value or the name of a variable already defined. The products are named with the specified *`suffix`* (maximum 3 characters). The original variable names are truncated first if need be. For instance, suppose you want to create per capita versions of certain variables, and you have the variable `pop` (population). A suitable set of commands is then: `genr invpop = 1/pop multiply invpop pc income expend` which will create `in-comepc` as the product of `income` and `invpop`, and `expendpc` as `expend` times `invpop`.

nulldata

Argument: *`series_length`*

Example: `nulldata 100`

Establishes a "blank" data set, containing only a constant, with periodicity 1 and the specified number of observations. This may be used for simulation purposes: some of the `genr` commands (e.g. `genr uniform()`, `genr normal()`, `genr time`) will generate dummy data from scratch to fill out the data set. This command may be useful in conjunction with `loop`. See also the `seed` command.

ols

Arguments: *`depvar indepvars`* `[ -o ]`

Examples: `ols 1 0 2 4 6 7`

`ols -o y 0 x1 x2 x3`

Computes ordinary least squares estimates with *`depvar`* as the dependent variable and *`indepvars`* as the list of independent variables. The `-o` flag calls for printing of the covariance matrix of regression coefficients. The variables may be specified by name or number; use the number zero for a constant term. The program also prints the p-values for *t* (two-tailed) and *F*-statistics. A p-value below 0.01 indicates significance at the 1 percent level and is denoted by ***. ** indicates significance between 1 and 5 percent and * indicates significance between 5 and 10 percent levels. Model selection statistics (described in Ramanathan, Section 4.3) are also printed. Various internal variables may be retrieved using the `genr` command, provided `genr` is invoked immediately after this command.

omit

Argument:	*`varlist`* `[ -o ]`
Example:	`omit 5 7 9`

This command must be invoked after an estimation command. The variables in *`varlist`* are omitted from the previous model and the new model estimated. If more than one variable is omitted, the Wald *F*-statistic for the omitted variables will be printed along with its p-value (for the OLS procedure only). A p-value below 0.05 means that the coefficients are jointly significant at the 5 percent level. Various internal variables may be retrieved using the `genr` command, provided `genr` is invoked immediately after this command. The coefficient covariance matrix is printed if the `-o` flag is given.

omitfrom

Arguments:	*`modelID varlist`*
Example:	`omitfrom 2 5 7 9`

Works like the `omit` command, except that you specify a previous model (using its ID number, which is printed at the start of the model output) to take as the base for omitting variables. The example above omits variables number 5, 7 and 9 from Model 2.

open

Argument:	*`datafile`*

Opens a data file. If a data file is already open, it is replaced by the newly opened one. The program will try to detect the format of the data file (native, CSV or BOX1).

pergm

Argument:	*`varname`* `[ -o ]`

Computes and displays (and if not in batch mode, graphs) the spectrum of the specified variable. Without the `-o` flag the sample periodogram is given; with the flag a Bartlett lag window of length 2 root T (where T is the sample size). is used in estimating the spectrum (see Chapter 18 of Greene's *Econometric Analysis*). When the sample periodogram is printed, a t-test for fractional integration of the series ("long memory") is also given: the null hypothesis is that the integration order is zero.

plot

Examples:	`plot x1`
	`plot x1 x2`
	`plot -o x1 x2`

Plots data values for specified variables, for the range of observations currently in effect, using ASCII symbols. Each line stands for an observation and the values are plotted horizontally. If the flag `-o` is present, `x1` and `x2` are plotted in the same scale, otherwise `x1` and `x2` are scaled appropriately. The `-o` flag should be used only if the variables have approximately the same range of values (e.g. observed and predicted dependent variable). See also `gnuplot`.

pooled

Arguments: *`depvar indepvars`* `[ -o ]`

Estimates a model via OLS (see `ols` for details on syntax), and flags it as a pooled or panel model, so that the `hausman` test item becomes available.

print

Argument: [*`varlist`*] `[ -o ]`

Examples:

```
print
print x y
print 1 2 3 -o
```

If *`varlist`* is given, prints the values of the specified variables, otherwise prints the values of all variables in the current data file. If the `-o` flag is given the data are printed by observation, otherwise they are printed by variable.

probit

Arguments: *`depvar indepvars`*

Probit regression. The dependent variable should be a binary variable. Maximum likelihood estimates of the coefficients on *`indepvars`* are obtained via iterated least squares (the EM or Expectation–Maximization method). As the model is nonlinear the slopes depend on the values of the independent variables: the reported slopes are evaluated at the means of those variables. The chi-square statistic tests the null hypothesis that all coefficients are zero apart from the constant.

Probit for analysis of proportions is not implemented in `gretl` at this point.

pvalue

Usage:

`pvalue 1` *`xvalue`* (normal distribution)

`pvalue 2` *`df xvalue`* (*t* distribution)

`pvalue 3` *`df xvalue`* (chi-square distribution)

`pvalue 4` *`dfn dfd xvalue`* (*F* distribution)

`pvalue 5` *`mean variance xvalue`* (Gamma distribution)

Computes the area to the right of *`xvalue`* in the specified distribution. *`df`* is the degrees of freedom, *`dfn`* is the d.f. for the numerator, *`dfd`* is the d.f. for the denominator. Instead of the code numbers you can use `z`, `t`, `X`, `F` and `G` for the normal, *t*, chi-square, *F*, and gamma distributions respectively.

quit

Exits from the program, giving you the option of saving the output from the session on the way out.

rhodiff

Arguments: *rholist* ; *varlist*

Examples: `rhodiff .65 ; 2 3 4`

`rhodiff r1 r2 ; x1 x2 x3`

Creates rho-differenced counterparts of the variables (given by number or by name) in *varlist* and adds them to the data set, using the suffix `#` for the new variables. Given variable `v1` in *varlist*, and entries `r1` and `r2` in *rholist*, `v1# = v1(t) - r1*v1(t-1) - r2*v1(t-2)` is created. The *rholist* entries can be given as numerical values or as the names of variables previously defined.

run

Argument: *inputfile*

Execute the commands in *inputfile* then return control to the interactive prompt.

runs

Argument: *varname*

Carries out the nonparametric "runs" test for randomness of the specified variable. If you want to test for randomness of deviations from the median, for a variable named `x1` with a non-zero median, you can do the following:

```
genr signx1 = x1 - median(x1)
runs signx1
```

scatters

Argument: *yvar* ; *xvarlist*

`scatters` *yvarlist* ; *xvar*

Examples: `scatters 1 ; 2 3 4 5`

`scatters 1 2 3 4 5 6 ; time`

Plots pairwise scatters of *yvar* against all the variables in *xvarlist*, or of all the variables in *yvarlist* against *xvar*. The first example above puts variable 1 on the *y*-axis and draws four graphs, the first having variable 2 on the *x*-axis, the second variable 3 on the *x*-axis, and so on. The second example plots each of variables 1 through 6 against time. Scanning a set of such plots can be a useful step in exploratory data analysis. The maximum number of plots is six; any extra variable in the list will be ignored.

seed

Argument: *integer*

Sets the seed for the pseudo-random number generator for the `uniform()` and `normal()` functions (see the `genr` command). By default the seed is set when the program is started, using the system time. If you want to obtain repeatable sequences of pseudo-random numbers you will need to set the seed manually.

setobs

Arguments: *periodicity startobs*

Examples: `setobs 4 1990.1`

```
setobs 12 1978.03
setobs 20 1.01
```

Force the program to interpret the current data set as time series or panel, when the data have been read in as simple undated series. *`periodicity`* must be an integer; *`startobs`* is a string representing the date or panel ID of the first observation. See also Chapter 6.

setmiss

Arguments:	*`value`* [*`varlist`*]
Examples:	`setmiss -1`
	`setmiss 100 x2`

Get the program to interpret some specific numerical data value (the first parameter to the command) as a code for "missing", in the case of imported data. If this value is the only parameter, as in the first example above, the interpretation will be applied to all series in the data set. If *`value`* is followed by a list of variables, by name or number, the interpretation is confined to the specified variable(s). Thus in the second example the data value 100 is interpreted as a code for "missing", but only for the variable `x2`.

shell

Usage:	`!` *`shellcommand`*

A `!` at the beginning of a command line is interpreted as an escape to the user's shell. Thus arbitrary shell commands can be executed from within the program (not available under MS Windows).

sim

Arguments:	*`startobs endobs y a0 a1 a2...`*	
Examples:	`sim 1979.2 1983.1 y 0 0.9`	creates `y(t) = 0.9*y(t-1)`
	`sim 15 25 y 10 0.8 x`	creates `y(t) = 10 + 0.8*y(t-1) + x(t)*y(t-2)`

Simulates values for *y* for the periods *`startobs`* through *`endobs`*. The variable *y* must have been defined earlier with appropriate initial values. The formula used is `y(t) = a0(t) + a1(t)*y(t-1) + a2(t)*y(t-2) + ...` The `ai(t)` may either be numerical constants or variable names previously defined.

smpl

Arguments:	*`startobs endobs`*
	`smpl -o` *`dummyvar`*
	`smpl -o`
	`smpl -r` *`condition`*

Resets the sample range. In the first form *`startobs`* and *`endobs`* must be consistent with the periodicity of the data. In the second form *`dummyvar`* must be an indicator variable with values 0 or 1 at each observation; the sample will be restricted to observations where the value is 1. The third form, `smpl -o`, drops all observations for which values of one or more variables are missing. The fourth form (`-r`) restricts the sample to observations that satisfy the given (Boolean) condition.

`smpl 3 10`	data with periodicity 1
`smpl 1950 1990`	annual data, periodicity 1
`smpl 1960.2 1982.4`	quarterly data
`smpl 1960.04 1985.10`	monthly data
`smpl 1960.2 ;`	keep *`endobs`* unchanged
`smpl ; 1984.3`	keep *`startobs`* unchanged
`smpl -o dum1`	draw sample of observations where `dum1=1`
`smpl -r income > 30000`	sample cases where `income` has a value greater than 30000.

One point should be noted about the `-o` and `-r` forms of `smpl`: Any "structural" information in the data header file (regarding the time series or panel nature of the data) is lost when this command is issued. You may reimpose structure with the `setobs` command.

spearman

Arguments: *`x y`* `[ -o ]`

Prints Spearman's rank correlation coefficient for the two variables *x* and *y*. The variables do not have to be ranked manually in advance; the function takes care of this. If the `-o` flag is supplied, the original data and the ranked data are printed out side by side.

The automatic ranking is from largest to smallest (i.e. the largest data value gets rank 1). If you need to invert this ranking, create a new variable which is the negative of the original first. For example:

```
genr altx = -x
spearman altx y
```

square

Argument: *`varlist`* `[ -o ]`

Generates new variables which are squares and cross products of the variables in *`varlist`* (`-o` will create the cross products). For example `square x y` will generate `sq_x` = `x` squared, `sq_y` = `y` squared and `x_y` = `x` times `y`. If a particular variable is a dummy variable it is not squared because we will get the same variable.

store

Argument: *`datafile`* [*`varlist`*] [*`flag`*]

`datafile` is the name of the file in which the values should be stored.

If *`varlist`* is absent, the values of all the variables in the current data set will be stored.

By default storage is in native `gretl` XML format. There are six valid (mutually exclusive) *`flags`*:

`-z`	The default format, but gzip compressed.
`-o`	Store the data by variables, in binary format using double precision.
`-s`	Store the data by variables, in binary format using single precision.
`-c`	Store the data in CSV (comma-separated values) format. Such data can be read directly by spreadsheet programs.
`-r`	Store the data in GNU R format.

`-m`	Store the data in GNU Octave format.
`-t`	Store the data in “traditional” ESL format, with an ascii data file and a separate informative header file.

summary

Argument: [*`varlist`*]

Print summary statistics for the variables in *`varlist`*, or for all the variables in the data set if *`varlist`* is omitted. Output consists of the mean, standard deviation (sd), coefficient of variation (= sd/mean), median, minimum, maximum, skewness coefficient, and excess kurtosis.

tabprint

Must follow the estimation of a model via OLS. Prints the estimated model in the form of a LaTeX tabular environment, to a file with a name of the form `model_N.tex`, where `N` is the number of models estimated to date in the current session. This can be incorporated in a LaTeX document. See also `eqnprint`.

testuhat

Must follow a model estimation command. Gives the frequency distribution for the residual from the model along with a chi-square test for normality.

tsls

Arguments: *`depvar varlist1`* ; *`varlist2`* [`-o`]

Example: `tsls y1 0 y2 y3 x1 x2 ; 0 x1 x2 x3 x4 x5 x6`

Computes two-stage least squares (TSLS) estimates of parameters. *`depvar`* is the dependent variable, *`varlist1`* is the list of independent variables (including right-hand side endogenous variables) in the structural equation for which TSLS estimates are needed. *`varlist2`* is the combined list of exogenous and predetermined variables in all the equations. If *`varlist2`* is not at least as long as *`varlist1`*, the model is not identified. The `-o` flag will print the covariance matrix of the coefficients. In the above example, the `y`s are the endogenous variables and the `x`s are the exogenous and predetermined variables. A number of internal variables may be retrieved using the `genr` command, provided `genr` is invoked immediately after this command.

var

Arguments: *`order depvar indepvar`*

Examples: `var 4 x1 const time x2 x3`

`var 3 1 0 2 3 4`

Sets up and estimates (via OLS) a vector autoregression. The first argument specifies the lag order, then follows the setup for the first equation, as in the `ols` command. Don’t include lags among the elements of the *`indepvar`* list — they will be added automatically. A regression will be run for each variable in the list, excluding the constant, the time trend and any dummy variables. Output for each equation includes F-tests for zero restrictions on all lags of each of the variables, and an F-test for the maximum lag.

varlist

Prints a listing of variables currently available. `list` and `ls` are synonyms.

vartest

Arguments: *`var1 var2`*

Calculates the *F* statistic for the null hypothesis that the population variances for the variables *`var1`* and *`var2`* are equal, and shows its p-value.

wls

Arguments: *`weightvar depvar indepvars`* `[ -o ]`

Weighted least squares estimates are obtained using *`weightvar`* as the weight, *`depvar`* as the dependent variable and *`indepvars`* as the list of independent variables. More specifically, an OLS regression is run on *`weightvar`* * *`depvar`* against *`weight`* * *`indepvars`*. If the *`weightvar`* is a dummy variable, this is equivalent to eliminating all observations with the number zero for *`weightvar`*. The flag `-o` will print the covariance matrix of coefficients. A number of internal variables may be retrieved using the `genr` command, provided `genr` is invoked immediately after this command.

Estimators and tests: summary

Table 10-2 shows the estimators available under the Model menu in `gretl`'s main window. The corresponding script command (if there is one available) is shown in parentheses. For details consult the command's entry in Chapter 10.

Table 10-2. Estimators

Estimator	Comment
Ordinary Least Squares (`ols`)	The workhorse estimator
Weighted Least Squares (`wls`)	Heteroskedasticity, exclusion of selected observations
HCCM (`hccm`)	Heteroskedasticity corrected covariance matrix
Heteroskedasticity corrected (`hsk`)	Weighted Least Squares based on predicted error variance
Cochrane–Orcutt (`corc`)	First-order autocorrelation
Hildreth–Lu (`hilu`)	First-order autocorrelation
Autoregressive Estimation (`ar`)	Higher-order autocorrelation (generalized Cochrane–Orcutt)
Vector Autoregression (`var`)	Systems of time-series equations
Cointegration test (`coint`)	Long-run relationships between series
Two-Stage Least Squares (`tsls`)	Simultaneous equations
Logit (`logit`)	Binary dependent variable (logistic distribution)
Probit (`probit`)	Binary dependent variable (normal distribution)
Rank Correlation (`spearman`)	Correlation with ordinal data
Pooled OLS ((`pooled`)	OLS estimation for pooled cross-section, time series data

Table 10-3 shows the tests that are available under the Tests menu in a model window, after estimation.

Table 10-3. Tests for models

Test	Corresponding command
Omit variables (*F*-test if OLS)	`omit`
Add variables (*F*-test if OLS)	`add`
Nonlinearity (squares)	`lmtest`
Heteroskedasticity (White's test)	`lmtest`
Autocorrelation up to the data frequency	`lmtest -o`
Chow (structural break)	`chow`
CUSUM (parameter stability)	`cusum`
ARCH (conditional heteroskedasticity)	`arch`
Normality of residual	`testuhat`
Panel diagnostics	`hausman`

Some additional tests are available under the Variable menu in the main window: the augmented Dickey–Fuller test (command: `adf`) and the runs test of randomness (command: `runs`).

Chapter 11. Troubleshooting gretl

As I steer `gretl` towards a "stable" release (version 1.0) I welcome any reports of bugs in the program. I think you are unlikely to find bugs in the actual calculations done by `gretl` (although this statement does not constitute any sort of warranty). You may, however, come across bugs or oddities in the behavior of the graphical interface. Please remember that the usefulness of bug reports is greatly enhanced if you can be as specific as possible: what *exactly* went wrong, under what conditions, and on what operating system? If you saw an error message, what precisely did it say? (You needn't bother, though, to quote the memory address numbers given in any crash reports from MS Windows — these won't mean anything to me.)

As mentioned above, `gretl` calls some other programs to accomplish certain tasks (gnuplot for graphing, LaTeX for high-quality typesetting of regression output, GNU R). If something goes wrong with such external links, it is not always easy to produce an informative error message window. If such a link fails when accessed from the `gretl` graphical interface, you may be able to get more information by starting `gretl` from the command prompt (e.g. from an xterm under the X window system, or from a "DOS box" under MS Windows, in which case type `gretlw32.exe`), rather than via a desktop menu entry or icon. Additional error messages may be displayed on the terminal window.

Also please note that for most external calls, `gretl` assumes that the programs in question are available in your "path" — that is, that they can be invoked simply via the name of the program, without supplying the program's full location.[1] Thus if a given program fails, try the experiment of typing the program name at the command prompt, as shown below.

System	**Graphing**	**Typsetting**	**GNU R**
X window system	gnuplot	latex, xdvi	R
MS Windows	wgnuplot.exe	latex, xdvi	RGui.exe

If the program fails to start from the prompt, it's not a `gretl` issue but rather that the program's home directory is not in your path, or the program is not installed (properly). For details on modifying your path please see the documentation or online help for your operating system or shell.

1. The exception to this rule is the invocation of gnuplot under MS Windows, where a full path to the program is given.

Chapter 12. The command line interface

The gretl package includes the command-line program gretlcli. This is essentially an updated version of Ramu Ramanathan's ESL. On unix-like systems it can be run from the console, or in an xterm (or similar). Under MS Windows it can be run in a "DOS box". gretlcli has its own help file, which may be accessed by typing "help" at the prompt. It can be run in batch mode, sending outout directly to a file (see the Section called *gretlcli* in Chapter 9 above).

If gretlcli is linked to the readline library (this is automatically the case in the MS Windows version; also see Appendix B), the command line is recallable and editable, and offers command completion. You can use the Up and Down arrow keys to cycle through previously typed commands. On a given command line, you can use the arrow keys to move around, in conjunction with Emacs editing keystokes.[1] The most common of these are:

Keystroke	**Effect**
`Ctrl-a`	go to start of line
`Ctrl-e`	go to end of line
`Ctrl-d`	delete character to right

where "`Ctrl-a`" means press the "`a`" key while the "`Ctrl`" key is also depressed. Thus if you want to change something at the beginning of a command, you *don't* have to backspace over the whole line, erasing as you go. Just hop to the start and add or delete characters.

If you type the first letters of a command name then press the Tab key, readline will attempt to complete the command name for you. If there's a unique completion it will be put in place automatically. If there's more than one completion, pressing Tab a second time brings up a list.

Changes from ESL

Command scripts developed for Ramanathan's original ESL should be usable with gretlcli with few or no changes: the only things to watch for are multi-line commands and the freq command.

§ In ESL, a semicolon is used as a terminator for many commands. I decided to remove this in gretlcli. Semicolons are simply ignored, apart from a few special cases where they have a definite meaning: as a separator for two lists in the ar and tsls commands, and as a marker for an unchanged starting or ending observation in the smpl command. In ESL semicolon termination gives the possibility of breaking long commands over more than one line; in gretlcli this is done by putting a trailing backslash \ at the end of a line that is to be continued.

§ With freq, you can't at present specify user-defined ranges as in ESL. A chi-square test for normality has been added to the output of this command.

Note also that the command-line syntax for running a batch job is simplified. For ESL you typed, e.g.

```
esl -b datafile < inputfile > outputfile
```

while for gretlcli you type:

```
gretlcli -b inputfile > outputfile
```

The inputfile is treated as a program argument; it should specify a datafile to use internally, using the syntax `open datafile` or the special comment `(* ! datafile *)`

1. Actually, the key bindings shown below are only the defaults; they can be customized. See the readline manual.

Appendix A. Data file details

Basic native format

In gretl's native data format, a data set is stored in XML (extensible mark-up language). Data files correspond to the simple DTD (document type definition) given in gretldata.dtd, which is supplied with the gretl distribution and is installed in the system data directory (e.g. /usr/share/gretl/data on a unix-type system). Data files may be plain text or gzipped. They contain the actual data values plus additional information such as the names and descriptions of variables, the frequency of the data, and so on.

Most users will probably not have need to read or write such files other than via gretl itself, but if you want to manipulate them using other software tools you should examine the DTD and also take a look at a few of the supplied practice data files: data4-1.gdt gives a simple example; data4-10.gdt is an example where observation labels are included.

Traditional ESL format

For backward compatibility, gretl can also handle data files in the "traditional" format inherited from Ramanathan's ESL program. In this format (which was the default in gretl prior to version 0.98) a data set is represented by two files. One contains the actual data and the other information on how the data should be read. To be more specific:

1. *Actual data*: A rectangular matrix of white-space separated numbers. Each column represents a variable, each row an observation on each of the variables (spreadsheet style). Data columns can be separated by spaces or tabs. The filename should have the suffix .gdt. By default the data file is ASCII (plain text). Optionally it can be gzip-compressed to save disk space. You can insert comments into a data file: if a line begins with the hash mark (#) the entire line is ignored. This is consistent with gnuplot and octave data files.
2. *Header*: The data file must be accompanied by a header file which has the same base-name as the data file plus the suffix .hdr. This file contains, in order:
   - (Optional) *comments* on the data, set off by the opening string (* and the closing string *), each of these strings to occur on lines by themselves.
   - (Required) list of white-space separated *names of the variables* in the data file. Names are limited to 8 characters, must start with a letter, and are limited to alphanumeric characters plus the underscore. The list may continue over more than one line; it is terminated with a semicolon, ;.
   - (Required) *observations* line of the form 1 1 85. The first element gives the data frequency (1 for undated or annual data, 4 for quarterly, 12 for monthly). The second and third elements give the starting and ending observations. Generally these will be 1 and the number of observations respectively, for undated data. For time-series data one can use dates of the form 1959.1 (quarterly, one digit after the point) or 1967.03 (monthly, two digits after the point). See Chapter 6 below for special use of this line in the case of panel data.
   - The keyword BYOBS.

Here is an example of a well-formed data header file.

```
(*
DATA9-6:
Data on log(money), log(income) and interest rate from US.
Source: Stock and Watson (1993) Econometrica
(unsmoothed data) Period is 1900-1989 (annual data).
Data compiled by Graham Elliott.
*)
```

```
lmoney lincome intrate ;
1 1900 1989 BYOBS
```

The corresponding data file contains three columns of data, each having 90 entries.

Three further features of the "traditional" data format may be noted.

1. If the `BYOBS` keyword is replaced by `BYVAR`, and followed by the keyword `BINARY`, this indicates that the corresponding data file is in binary format. Such data files can be written from `gretlcli` using the `store` command with the `-s` flag (single precision) or the `-o` flag (double precision).
2. If `BYOBS` is followed by the keyword `MARKERS`, `gretl` expects a data file in which the *first column* contains strings (8 characters maximum) used to identify the observations. This may be handy in the case of cross-sectional data where the units of observation are identifiable: countries, states, cities or whatever. It can also be useful for irregular time series data, such as daily stock price data where some days are not trading days — in this case the observations can be marked with a date string such as `10/01/98`. (Remember the 8-character maximum.) Note that `BINARY` and `MARKERS` are mutually exclusive flags. Also note that the "markers" are not considered to be a variable: this column does not have a corresponding entry in the list of variable names in the header file.
3. If a file with the same base name as the data file and header files, but with the suffix `.lbl`, is found, it is read to fill out the descriptive labels for the data series. The format of the label file is simple: each line contains the name of one variable (as found in the header file), followed by one or more spaces, followed by the descriptive label. Here is an example: `price New car price index, 1982 base year`

If you want to save data in traditional format, use the `-t` flag with the `store` command, either in the command-line program or in the console window of the GUI program.

Binary database details

A `gretl` database consists of two parts: an ASCII index file (with filename suffix `.idx`) containing information on the series, and a binary file (suffix `.bin`) containing the actual data. Two examples of the format for an entry in the `idx` file are shown below:

```
GOM910  Composite index of 11 leading indicators (1987=100)
M 1948.01 - 1995.11  n = 575
currbal Balance of Payments: Balance on Current Account; SA
Q 1960.1 - 1999.4 n = 160
```

The first field is the series name. The second is a description of the series (maximum 128 characters). On the second line the first field is a frequency code: `M` for monthly, `Q` for quarterly and `A` for annual. No other frequencies are accepted at present. Then comes the starting date (N.B. with two digits following the point for monthly data, one for quarterly data, none for annual), a space, a hyphen, another space, the ending date, the string "`n = `" and the integer number of observations. This format must be respected exactly.

Optionally, the first line of the index file may contain a short comment (up to 64 characters) on the source and nature of the data, following a hash mark. For example:

```
# Federal Reserve Board (interest rates)
```

The corresponding binary database file holds the data values, represented as "floats", that is, single-precision floating-point numbers, typically taking four bytes apiece. The numbers are packed "by variable", so that the first *n* numbers are the observations of variable 1, the next *m* the observations on variable 2, and so on.

Appendix B. Technical notes

`Gretl` is written in the C programming language. I have abided as far as possible by the ISO/ANSI C Standard (C89), although the graphical user interface and some other components necessarily make use of platform-specific extensions.

`gretl` is being developed under Linux. The shared library and command-line client should compile and run on any platform that (a) supports ISO/ANSI C and (b) has the zlib (compression) and libxml (XML manipulation) libraries installed. The homepage for zlib can be found at info-zip.org. Libxml is at xmlsoft.org. If the GNU readline library is found on the host system this will be used for `gretcli`, providing a much enhanced editable command line. See the readline homepage.

The graphical client program should compile and run on any system that, in addition to the above requirements, offers GTK version 1.2.3 or higher (see gtk.org).

`gretl` calls gnuplot for graphing. You can find gnuplot at gnuplot.info. As of this writing the current version is 3.7.2.

Some features of `gretl` (the built-in spreadsheet, the "session" icon window, some file selection dialogs) make use of Adrian Feguin's `gtkextra` library. You can find gtkextra at gtkextra.sourceforge.net.

A binary version of the program is available for the Microsoft Windows platform (32-bit version, i.e. Windows 95 or higher). This version was cross-compiled under Linux using mingw (the GNU C compiler, `gcc`, ported for use with win32) and linked against the Microsoft C library, `msvcrt.dll`. It uses Tor Lillqvist's port of GTK to win32. The (free, open-source) Windows installer program is courtesy of Jordan Russell (jrsoftware.org).

I'm hopeful that some users with coding skills may consider `gretl` sufficiently interesting to be worth improving and extending. To date I have not attempted to document the libgretl API (other than via the header files you'll find in the `lib/src` subdirectory of the source package). But I welcome email on this subject and if there's sufficient interest I'll put some time into documentation.

Appendix C. Advanced econometric analysis with free software

As mentioned in the main text, gretl offers a reasonably full selection of least-squares based estimators, plus a few additional estimators sych as (binomial) logit and probit. Advanced users may, however, find gretl's menu of statistical routines restrictive.

No doubt some advanced users will prefer to write their own statistical code in a fundamental computer language such as C, C++ or Fortran. Another option is to use a relatively high-level language that offers easy matrix manipulation and that already has numerous statistical routines built in, or available as add-on packages. If the latter option sounds attractive, and you are interested in using free, open source software, I would recommend taking a look at either GNU R (r-project.org) or (GNU Octave). These programs are very close to the commercial programs S and Matlab respectively.

Also as mentioned above, gretl offers the facility of exporting data in the formats of both Octave and R. In the case of Octave, the gretl data set is saved thus: the first variable listed for export is treated as the dependent variable and is saved as a vector, y, while the remaining variables are saved jointly as a matrix, X. You can pull the X matrix apart if you wish, once the data are loaded in Octave. See the Octave manual for details. As for R, the exported data file preserves any time series structure that is apparent to gretl. The series are saved as individual structures. The data should be brought into R using the source() command.

Of these two programs, R is perhaps more likely to be of immediate interest to econometricians since it offers more in the way of statistical routines (e.g. generalized linear models, maximum likelihood estimation, time series methods). I have therefore supplied gretl with a convenience function for moving data quickly into R. Under gretl's Session menu, you will find the entry "Start GNU R". This writes out an R version of the current gretl data set (Rdata.tmp, in the user's gretl directory), and sources it into a new R session. A few details on this follow.

First, the data are brought into R by writing a temporary version of .Rprofile in the current working directory. (If such a file exists it is referenced by R at startup.) In case you already have a personal .Rprofile in place, the original file is temporarily moved to .Rprofile.gretltmp, and on exit from gretl it is restored. (If anyone can suggest a cleaner way of doing this I'd be happy to hear of it.)

Second, the particular way R is invoked depends on the internal gretl variable Rcommand, whose value may be set under the File, Preferences menu. The default command is RGui.exe under MS Windows. Under X it is either R --gui=gnome if an installation of the Gnome desktop (gnome.org) was detected at compile time, or xterm -e R if Gnome was not found. Please note that at most three space-separated elements in this command string will be processed; any extra elements are ignored.

Appendix D. Listing of URLs

Below is a listing of the full URLs of websites mentioned in the text, in their order of appearance.

GNU R homepage

`http://www.r-project.org/`

GNU Octave homepage

`http://www.octave.org/`

Gretl homepage

`http://gretl.sourceforge.net/`

Census Bureau, Data Extraction Service

`http://www.census.gov/ftp/pub/DES/www/welcome.html`

Gretl data page

`http://ricardo.ecn.wfu.edu/gretl/gretl_data.html`

Estima (RATS)

`http://www.estima.com/`

Penn World Table

`http://pwt.econ.upenn.edu/`

Gnuplot homepage

`http://www.gnuplot.info/`

Gnuplot online manual

`http://ricardo.ecn.wfu.edu/gnuplot.html`

GNU Readline manual

`http://cnswww.cns.cwru.edu/~chet/readline/readline.html`

InfoZip homepage

`http://www.info-zip.org/pub/infozip/zlib/`

Xmlsoft homepage

`http://xmlsoft.org/`

GNU Readline homepage

`http://cnswww.cns.cwru.edu/~chet/readline/rltop.html`

GTK+ homepage

`http://www.gtk.org/`

Gtkextra homepage

`http://gtkextra.sourceforge.net/`

Mingw (gcc for win32) homepage

`http://www.mingw.org/`

GTK+ port for win32

`http://user.sgic.fi/~tml/gimp/win32/`

JRSoftware

`http://www.jrsoftware.org/`

Gnome desktop homepage

`http://www.gnome.org/`

GNU R manual

`http://cran.r-project.org/doc/manuals/R-intro.pdf`

Bibliography

Box, G. E. P. and Muller, M. E. (1958) “A Note on the Generation of Random Normal Deviates”, *Annals of Mathematical Statistics*, 29, pp. 610–11.

Greene, William H. (2000) *Econometric Analysis*, 4th edition, Upper Saddle River, NJ: Prentice-Hall.

Kiviet, J. F. (1986) “On the Rigour of Some Misspecification Tests for Modelling Dynamic Relationships”, *Review of Economic Studies*, 53, pp. 241–261.

MacKinnon, J. G. and White, H. (1985) “Some Heteroskedasticity-Consistent Covariance Matrix Estimators with Improved Finite Sample Properties”, *Journal of Econometrics*, 29, pp. 305–25.

R Core Development Team (2000) *An Introduction to R*, version 1.1.1,

Ramanathan, Ramu (2002) *Introductory Econometrics with Applications*, 5th edition, Fort Worth: Harcourt.

Ruud, Paul A. (2000) *An Introduction to Classical Econometric Theory*, New York and Oxford: Oxford University Press.

Salkever, D. (1976) “The Use of Dummy Variables to Compute Predictions, Prediction Errors, and Confidence Intervals”, *Journal of Econometrics*, 4, pp. 393–7.