# Gretl Manual

Gnu Regression, Econometrics and Time-series Library

Allin Cottrell
Department of Economics
Wake Forest University

October, 2003

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

Copyright © 2001–2003 Allin Cottrell

Permission is granted to copy, distribute and/or modify this document under the terms of the *GNU Free Documentation License*, Version 1.1 or any later version published by the Free Software Foundation (see http://www.gnu.org/licenses/fdl.html).

# Table of Contents

## List of Tables

## List of Figures

## List of Examples

# Chapter 1. Introduction

## Features at a glance

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

*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 the econometrics textbooks by Ramu Ramanathan and Jeffrey Wooldridge 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 and nonlinear least squares. It also offers (binomial) logit and probit estimation, and has a loop construct for running Monte Carlo analyses or iterative 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 D).

*Accurate*

gretl has been thoroughly tested on the NIST reference datasets. See Appendix C.

*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.

*International*

gretl will produce its output in English, Spanish or French, depending on your computer's native language setting.

## 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 grateful to William Greene, author of *Econometric Analysis*, for permission to include in the gretl distribution some of the data sets analysed in his text, and to Jeffrey

Wooldridge for helping me prepare a `gretl` version of the data sets from his *Introductory Econometrics: A Modern Approach*.

With regard to the internationalization of `gretl`, I wish to thank Ignacio Díaz-Emparanza and Michel Robitaille, who prepared the Spanish and French translations respectively.

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 Stallman 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.

Many users of `gretl` have submitted useful suggestions and bug reports. In this connection particular thanks are due to Ignacio Díaz-Emparanza and Dirk Eddelbuettel, who maintains the `gretl` package for Debian GNU/Linux.

## Installing the programs

### Linux

On the Linux[1] 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-1.1.5`).
4. The basic routine is then

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

   However, you should probably read the `INSTALL` file first, and/or 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`).

1. Terminology is a bit of a problem here, but in this manual I will use "Linux" as shorthand to refer to the GNU/Linux operating system. What is said herein about Linux mostly applies to other unix-type systems too, though some local modifications may be needed.

## 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

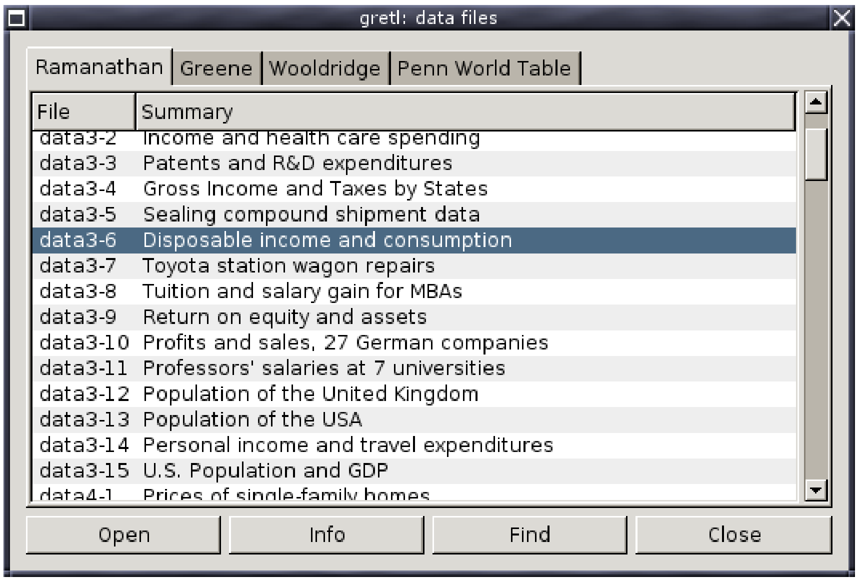
## Let's run a regression

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**



| File | Summary |
|---|---|
| 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 |
| data3-10 | Profits and sales, 27 German companies |
| data3-11 | Professors' salaries at 7 universities |
| data3-12 | Population of the United Kingdom |
| data3-13 | Population of the USA |
| data3-14 | Personal income and travel expenditures |
| data3-15 | U.S. Population and GDP |
| data4-1 | Prices of single-family homes |

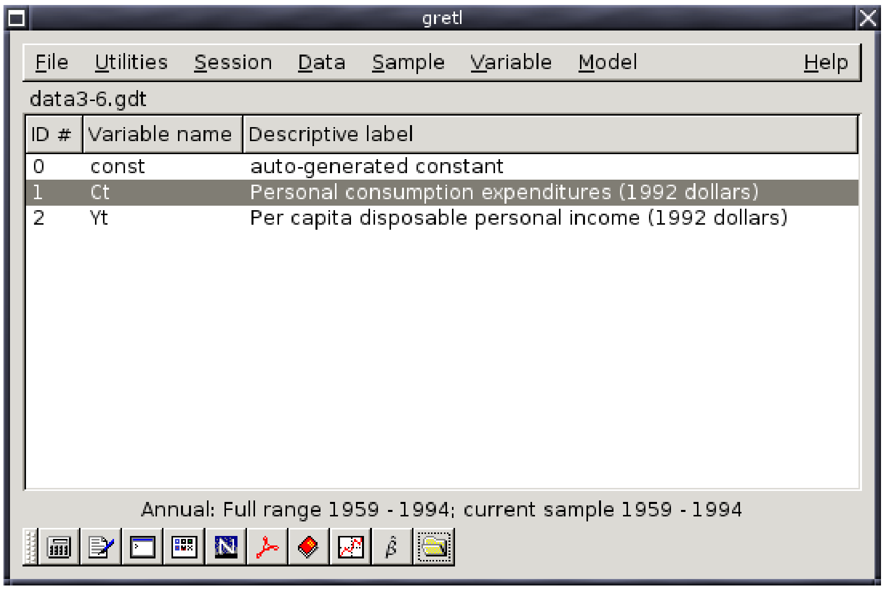
If you select a row in this window and click on "Info" this pops open the the "header file" for the data set in question, which tells you something about the source and definition of the variables. If you find a file that is of interest, you may open it by clicking on "Open", or just double-clicking 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 Linux it is called `gretl_x11` while on MS Windows it is `gretlw32.exe`. On Linux systems a wrapper script named `gretl` is also installed — see also Chapter 9.

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



data3-6.gdt

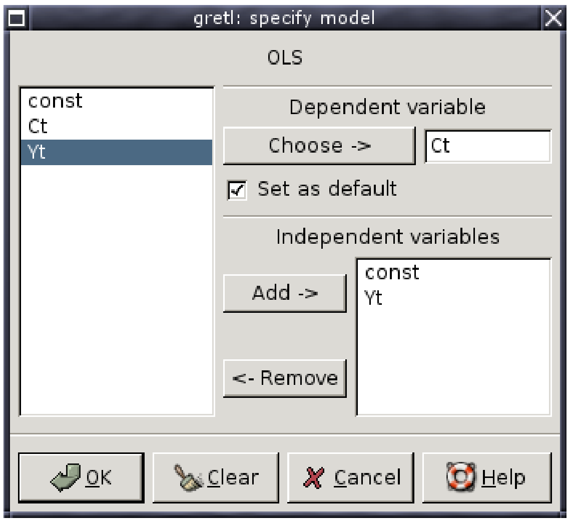
| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | auto-generated constant |
| 1 | Ct | Personal consumption expenditures (1992 dollars) |
| 2 | Yt | Per capita disposable personal income (1992 dollars) |

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 the Section called *The main window menus* 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* — see Figure 2-3.

**Figure 2-3. Model specification dialog**



To select the dependent variable, highlight the variable you want in the list on the left and click the "Choose" button that points to the Dependent variable slot. If you check the "Set as default" box this variable will be pre-selected as dependent when you next open the model dialog box. Shortcut: double-clicking on a variable on the left selects it as dependent and also sets it as the default. To select independent variables, highlight them on the left and click the "Add" button (or click the right mouse button over the highlighted variable).

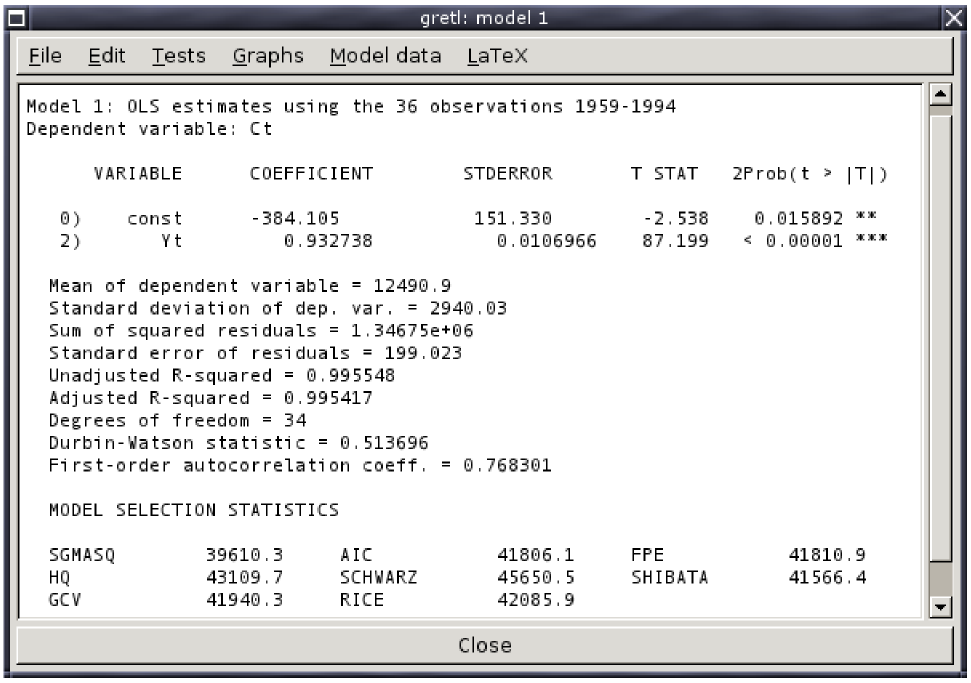
To select several variable in the list box, drag the mouse over them; to select several non-contiguous variables, hold down the `Ctrl` key and click on the variables you want.

To run a regression with consumption as the dependent variable and income as independent, click `Ct` into the Dependent slot and add `Yt` to the Independent variables list.

## 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 2-4).

**Figure 2-4. 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      -384.105          151.330       -2.538     0.015892 **
  2)       Yt         0.932738         0.0106966    87.199    < 0.00001 ***

   Mean of dependent variable = 12490.9
   Standard deviation of dep. var. = 2940.03
   Sum of squared residuals = 1.34675e+06
   Standard error of residuals = 199.023
   Unadjusted R-squared = 0.995548
   Adjusted R-squared = 0.995417
   Degrees of freedom = 34
   Durbin-Watson statistic = 0.513696
   First-order autocorrelation coeff. = 0.768301

   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.

For most models there is also an option to reprint the regression output in LaTeX format. 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.

To import `gretl` output into a word processor, you may copy and paste from an output window, using its Edit menu (or Copy button, in some contexts) to the target program. Many (not all) `gretl` windows offer the option of copying in RTF (Microsoft's "Rich Text Format") or as LaTeX. If you are pasting into a word processor, RTF may be a good option because the tabular formatting of the output is preserved.[2] Alternatively, you can save the output to a (plain text) 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.

2. Note that when you copy as RTF under MS Windows, Windows will only allow you to paste the material into applications that "understand" RTF. Thus you will be able to paste into MS Word, but not into notepad. Note also that there appears to be a bug in some versions of Windows, whereby the paste will not work properly unless the "target" application (e.g. MS Word) is running prior to copying the material in question.

Note that on the gnome desktop and under MS Windows, the File menu includes a command to send the output directly to a printer.

When pasting or importing plain text 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.

## 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 4.

—Append data: Add data to the current working data set, from a comma-separated or spreadsheet file.

—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. See Chapter 4.

—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 4 and also Appendix D.

—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 4).

—Browse databases: See the Section called *Binary databases* in Chapter 4 and the Section called *Creating a data file from scratch* in Chapter 4.

—Create data set: Initialize the built-in spreadsheet for entering data manually. See the Section called *Creating a data file from scratch* in Chapter 4.

—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. See Chapter 9 for details.

—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 D.

## § 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 3.

—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.

—Sort variables: Rearrange the listing of variables in the main window, either by ID number or alphabetically by name.

—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. 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 info: "Read info" just displays the header file information for the current data file; "Edit info" 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 may be useful with panel data; see Chapter 5.
- Compact data: For time-series data of higher than annual frequency, gives you the option of compacting the data to a lower frequency, using one of four compaction methods (average, sum, start of period or end of period).
- 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 4).
- 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.
- Set missing value code: Set a numerical value that will be interpreted as "missing" or "not available".
- 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 4.
- Interpret as time series: Opens a dialog box which enables you to set a time-series interpretation for data that were read in as undated.
- Interpret as panel: Opens 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 5).
- Restructure panel: Allows the conversion of a panel data set in stacked cross-section form into stacked time series. (Unlike the previous item, this one changes the organization of the data.)

§ 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 under "Edit attributes". 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
```

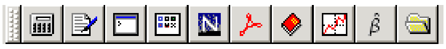
will create a new variable `foo` as the product of the existing variables `x1` and `x2`. In these formulae, variables must be referenced by name, not number.

§ Model menu 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". Also see Chapter 7 regarding the estimation of nonlinear models.

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

## 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, "Programs" tab.
2. Start a new script. Opens an editor window in which you can type a series of commands to be sent to the program as a batch.
3. Open the gretl console. A shortcut to the "Gretl console" menu item (the Section called *The main window menus* 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. Open the dialog box for estimating a model using ordinary least squares.
10. Open a window listing the datasets associated with Ramanathan's *Introductory Econometrics* (and also the datasets from Jeffrey Wooldridge's text, if these are installed — see Chapter 9).

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 3. Modes of working

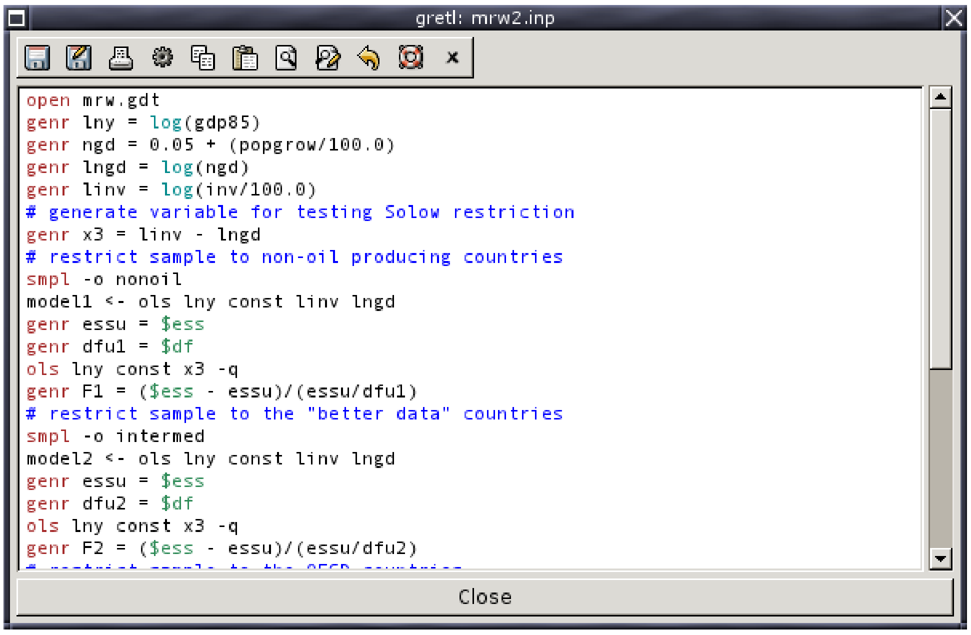
## Command scripts

As you execute commands in `gretl`, using the GUI and filling in dialog entries, those commands are recorded in the form of a "script" or batch file. 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 "View 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 or the "new script" toolbar button. In either case a script window will open (see Figure 3-1).

**Figure 3-1. Script window, editing a command file**



```
open mrw.gdt
genr lny = log(gdp85)
genr ngd = 0.05 + (popgrow/100.0)
genr lngd = log(ngd)
genr linv = log(inv/100.0)
# generate variable for testing Solow restriction
genr x3 = linv - lngd
# restrict sample to non-oil producing countries
smpl -o nonoil
model1 <- ols lny const linv lngd
genr essu = $ess
genr dful = $df
ols lny const x3 -q
genr F1 = ($ess - essu)/(essu/dfu1)
# restrict sample to the "better data" countries
smpl -o intermed
model2 <- ols lny const linv lngd
genr essu = $ess
genr dfu2 = $df
ols lny const x3 -q
genr F2 = ($ess - essu)/(essu/dfu2)
```

The toolbar at the top of the script window offers the following functions (left to right): (1) Save the file; (2) Save the file under a specified name; (3) Print the file (under Windows or the gnome desktop only); (4) Execute the commands in the file; (5) Copy selected text; (6) Paste the selected text; (7) Find and replace text; (8) Undo the last Paste or Replace action; (9) Help (if you place the cursor in a command word and press the question mark you will get help on that command); (10) Close the 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 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.

When you estimate a model using point-and-click, the model results are displayed in a separate window, offering menus which let you perform tests, draw graphs, save data from the model, and so on. Ordinarily, when you estimate a model using a script you just get a non-interactive printout of the results. You can, however, arrange for models estimated in a script to be "captured", so that you can examine them interactively when the script is finished. Here is an example of the syntax for achieving this effect:

```
Model1 <- ols Ct 0 Yt
```

That is, you type a name for the model to be saved under, then a back-pointing "assignment arrow", then the model command. You may use names that have embedded spaces if you like, but such names must always be wrapped in double quotes:

```
"Model 1" <- ols Ct 0 Yt
```

Models saved in this way will appear as icons in the `gretl` session window (see the Section called *The "session" concept*) after the script is executed. In addition, you can arrange to have a named model displayed (in its own window) automatically as follows:

```
Model1.show
```

Again, if the name contains spaces it must be quoted:

```
"Model 1".show
```

The same commands can be used for graphs. For example the following will create a plot of `Ct` against `Yt`, save it under the name "CrossPlot", and have it displayed:

```
CrossPlot <- gnuplot Ct Yt
CrossPlot.show
```

## The gretl console

A further option is available for your computing convenience. Under `gretl`'s Utilities menu you will find the item "Gretl console" (there is also an "open gretl console" button on the toolbar in the main window). 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.

In the console, you have "command history"; that is, you can use the up and down arrow keys to navigate the list of command you have entered to date. You can retrieve, edit and then re-enter a previous command.

## The "session" concept

### Introduction

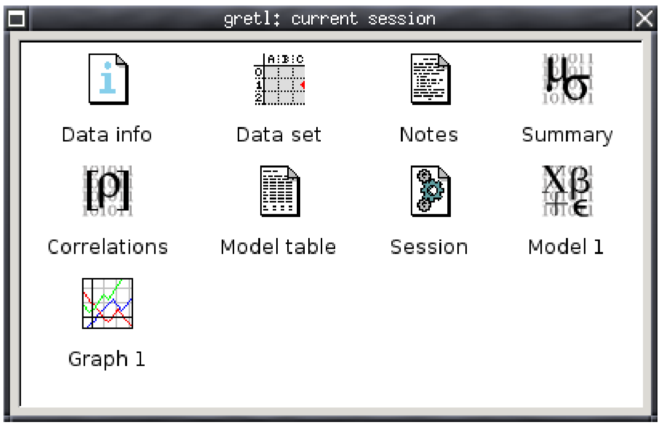
gretl offers the idea of a "session" as a way of keeping track of your work and revisiting it later. The basic idea is to provide an iconic space containing various objects pertaining to your current working session (see Figure 3-2). You can add objects (represented by icons) to this space as you go along. If you save the session, these added objects should be available again if you re-open the session later.

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.

**Figure 3-2. Icon view: one model and one graph have been added to the default icons**



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). Click on the graph window to bring up the graph menu, and select "Save to session as icon".

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.

### The model table

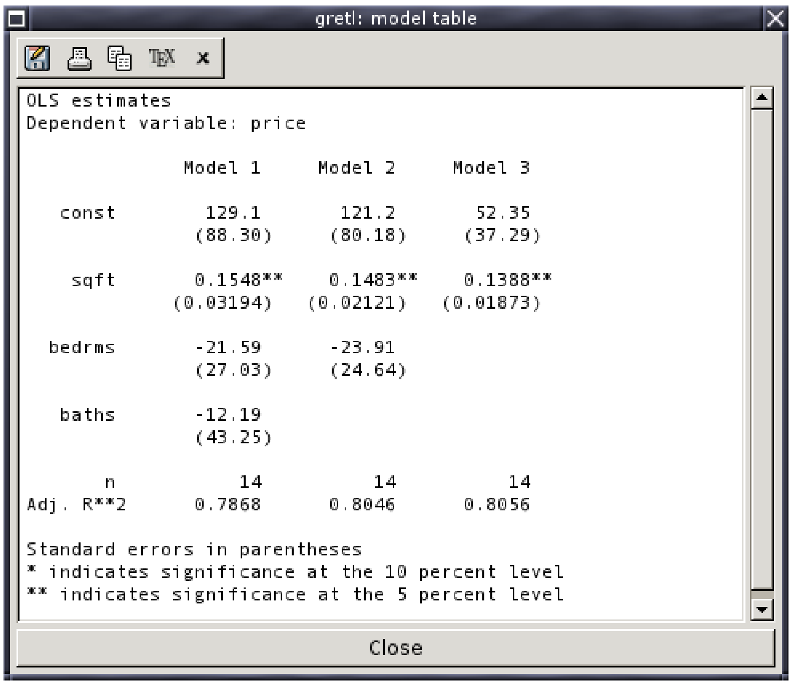
In econometric research it is common to estimate several models with a common dependent variable — the models differing in respect of which independent variables are included, or perhaps in respect of the estimator used. In this situation it is convenient to present the regression results in the form of a table, where each column contains the results (coefficient estimates and standard errors) for a given model, and each row contains the estimates for a given variable across the models.

In the Icon View window `gretl` provides a means of constructing such a table (and copying it in plain text, LaTeX or Rich Text Format). Here is how to do it:

1. Estimate a model which you wish to include in the table, and in the model display window, under the File menu, select “Save to session as icon” or “Save as icon and close”.
2. Repeat step 1 for the other models to be included in the table (up to a total of six models).
3. When you are done estimating the models, open the icon view of your gretl session, by selecting “Icon view” under the Session menu in the main gretl window, or by clicking the “session icon view” icon on the gretl toolbar.
4. In session icon view, there is an icon labeled “Model table”. Decide which model you wish to appear in the left-most column of the model table and add it to the table, either by dragging its icon onto the Model table icon, or by right-clicking on the model icon and selecting “Add to model table” from the pop-up menu.
5. Repeat step 4 for the other models you wish to include in the table. The second model selected will appear in the second column from the left, and so on.
6. When you are finished composing the model table, display it by double-clicking on its icon. Under the Edit menu in the window which appears, you have the option of copying the table to the clipboard in various formats.
7. If the ordering of the models in the table is not what you wanted, right-click on the model table icon and select “Clear table”. Then go back to step 4 above and try again.

A simple instance of `gretl`’s model table is shown in Figure 3-3.

**Figure 3-3. Example of model table**



```
OLS estimates
Dependent variable: price

                 Model 1      Model 2      Model 3

   const          129.1        121.2        52.35
                 (88.30)      (80.18)      (37.29)

    sqft         0.1548**     0.1483**     0.1388**
               (0.03194)    (0.02121)    (0.01873)

  bedrms         -21.59       -23.91
                 (27.03)      (24.64)

   baths         -12.19
                 (43.25)

       n             14           14           14
Adj. R**2        0.7868       0.8046       0.8056

Standard errors in parentheses
* indicates significance at the 10 percent level
** indicates significance at the 5 percent level
```

### Saving and re-opening sessions

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.

# Chapter 4. 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.

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

§ Comma-Separated Values (CSV) files. These can be imported using `gretl`'s "File, Open Data, Import CSV. . . " menu item, or the `import` script command. See also 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 ASCII, 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 D). 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 five ways to do this: (1) Find, or create using a text editor, a plain text data file and open it with `gretl`'s "Import ASCII" option. (2) Use your favorite spreadsheet to establish the data file, save it in Comma Separated Values format if necessary (this should not be necessary if the spreadsheet program is MS Excel or Gnumeric), then use one of `gretl`'s "Import" options (CSV, Excel or Gnumeric, as the case may be). (3) Use `gretl`'s built-in spreadsheet. (4) Select data series from a suitable database. (5) Use your favorite text editor or other software tools to a create data file in `gretl` format independently.

Here are a few comments and details on these methods.

### Common points on imported data

Options (1) and (2) involve using `gretl`'s "import" mechanism. For `gretl` to read such data successfully, certain general conditions must be satisfied:

§ The first row must contain valid variable names. A valid variable name is of 8 characters maximum; starts with a letter; and contains nothing but letters, numbers and the underscore character, _. (Longer variable names will be truncated to 8 characters.) Qualifications to the above: First, in the case of an ASCII or CSV import, if the file contains no row with variable names the program will automatically add names, `v1`, `v2` and so on. Second, by "the first row" is meant the first *relevant* row. In the case of ASCII and CSV imports, blank rows and rows beginning with a hash mark, `#`, are ignored. In the case of Excel and Gnumeric imports, you are presented with a dialog box where you can select an offset into the spreadsheet, so that `gretl` will ignore a specified number of rows and/or columns.

§ Data values: these should constitute a rectangular block, with one variable per column (and one observation per row). The number of variables (data columns) must match the number of variable names given. See also the Section called *Missing data values*.

§ Dates (or observation labels): Optionally, the *first* column may contain strings such as dates, or labels for cross-sectional observations. Such strings have a maximum of 8 characters (as with variable names, longer strings will be truncated). A column of this sort should be headed with the string `obs` or `date`, or the first row entry may be left blank.

For dates to be recognized as such, the date strings must adhere to one or other of a set of specific formats, as follows. For *annual* data: 4-digit years. For *quarterly* data: a 4-digit year, followed by a separator (either a period, a colon, or the letter `Q`), followed by a 1-digit quarter. Examples: `1997.1`, `2002:3`, `1947Q1`. For *monthly* data: a 4-digit year, followed by a period or a colon, followed by a two-digit month. Examples: `1997.01`, `2002:10`.

CSV files can use comma, space or tab as the column separator. When you use the "Import CSV" menu item you are prompted to specify the separator. In the case of "Import ASCII" the program attempts to auto-detect the separator that was used.

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”.

## Appending imported data

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 matches that of 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).

1. See www.estima.com

### Creating a gretl data file independently

It is possible to create a data file in one or other of `gretl`'s own formats 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 4.

### 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 as `NA`. 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 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 5. Panel data

## Panel structure

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. Under `gretl`'s Sample menu you will find an item "Restructure panel" which allows you to convert from stacked cross-section form to stacked time series.

In either case you can use the frequency field in the *observations* line of the data header file (see Chapter 4) 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 then import the data into `gretl`, the program may 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.

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.

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. "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 are not unusable: the "previous" value is not actually the first lag of `x1_1` but rather the last observation of `x1` for the previous cross-sectional unit. Such values are marked as missing by `gretl`.

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. Example 5-2 shows illustrative commands, assuming the unit data blocks each contain 30 observations and we want to drop the first row of each. You can then 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).

**Example 5-1. Lags with panel data**

```
(* 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
```

## 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 5-2 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 5-2. 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 6. Graphs and plots

## Gnuplot graphs

A separate program, `gnuplot`, is called to generate graphs. Gnuplot is a very full-featured graphing program with myriad options. It is available from www.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 a subset of gnuplot's options and it tries to choose sensible values for you; it also allows you to take complete control over graph details if you wish.

With a graph displayed, you can click on the graph window for a pop-up menu with the following options.

§ Save as postscript: Save the graph in encapsulated postscript (EPS) format.

§ Save as PNG: Save in Portable Network Graphics format.

§ Save to session as icon: The graph will appear in iconic form when you select "Icon view" from the Session menu.

§ Zoom: Lets you select an area within the graph for closer inspection (not available for all graphs).

§ Print: On the Gnome desktop only, lets you print the graph directly.

§ Copy to clipboard: MS Windows only, lets you paste the graph into Windows applications such as MS Word.[1]

§ Edit: Opens a controller for the plot which lets you adjust various aspects of its appearance.

§ Close: Closes the graph window.

If you know something about `gnuplot` and wish to get finer control over the appearance of a graph than is available via the graphical controller ("Edit" option), you have two further options.

§ Once the graph is saved as a session icon, you can right-click on its icon for a further pop-up menu. One of the otions here is "Edit plot commands", which opens an editing window with the actual gnuplot commands displayed. You can edit these commands and either save them for future processing or send them to gnuplot (with the "File/Send to gnuplot" menu item in the plot commands editing window).

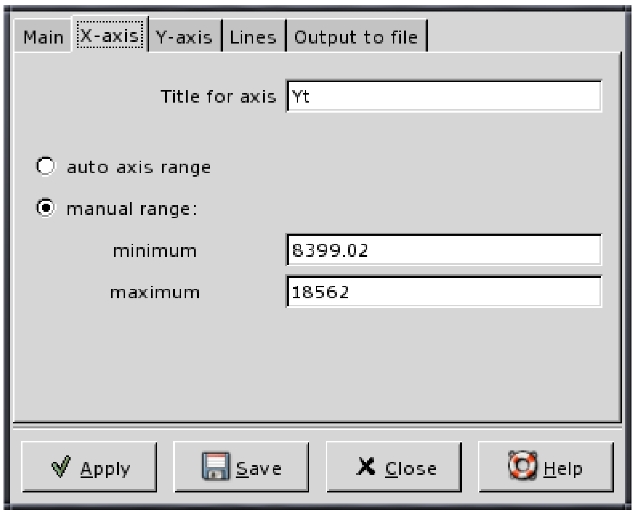
§ Another way to save the plot commands (or to save the displayed plot in formats other than EPS or PNG) is to use "Edit" item on a graph's pop-up menu to invoke the graphical controller, then click on the "Output to file" tab in the controller. You are then presented with a drop-down menu of formats in which to save the graph.

To find out more about `gnuplot` see the online manual or www.gnuplot.info.

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

1. For best results when pasting graphs into MS Office applications, choose the application's "Edit, Paste Special..." menu item, and select the option "Picture (Enhanced Metafile)".

**Figure 6-1. gretl's gnuplot controller**



Title for axis: Yt

auto axis range [✓ manual range:]

minimum: 8399.02

maximum: 18562

## 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 7. Nonlinear least squares

## Introduction and examples

As of version 1.0.9, gretl supports nonlinear least squares (NLS) using a variant of the Levenberg–Marquandt algorithm. The user must supply a specification of the regression function; prior to giving this specification the parameters to be estimated must be "declared" and given initial values. Optionally, the user may supply analytical derivatives of the regression function with respect to each of the parameters. The tolerance (criterion for terminating the iterative estimation procedure) can be set using the genr command.

The syntax for specifying the function to be estimated is the same as for the genr command. Here are two examples, with accompanying derivatives.

**Example 7-1. Consumption function from Greene**

```
nls C = alpha + beta * Y^gamma
deriv alpha = 1
deriv beta = Y^gamma
deriv gamma = beta * Y^gamma * log(Y)
end nls
```

**Example 7-2. Nonlinear function from Russell Davidson**

```
nls y = alpha + beta * x1 + (1/beta) * x2
deriv alpha = 1
deriv beta = x1 - x2/(beta*beta)
end nls
```

Note the command words nls (which introduces the regression function), deriv (which introduces the specification of a derivative), and end nls, which terminates the specification and calls for estimation. If the -o flag is appended to the last line the covariance matrix of the parameter estimates is printed.

## Initializing the parameters

The parameters of the regression function must be given initial values prior to the nls command. This can be done using the genr command (or, in the GUI program, via the menu item "Define new variable"). In some cases, where the nonlinear function is a generalization of (or a restricted form of) a linear model, it may be convenient to run an ols and initialize the parameters from the OLS coefficient estimates. In relation to the first example above, one might do:

```
ols C 0 Y
genr alpha = coeff(0)
genr beta = coeff(Y)
genr gamma = 1
```

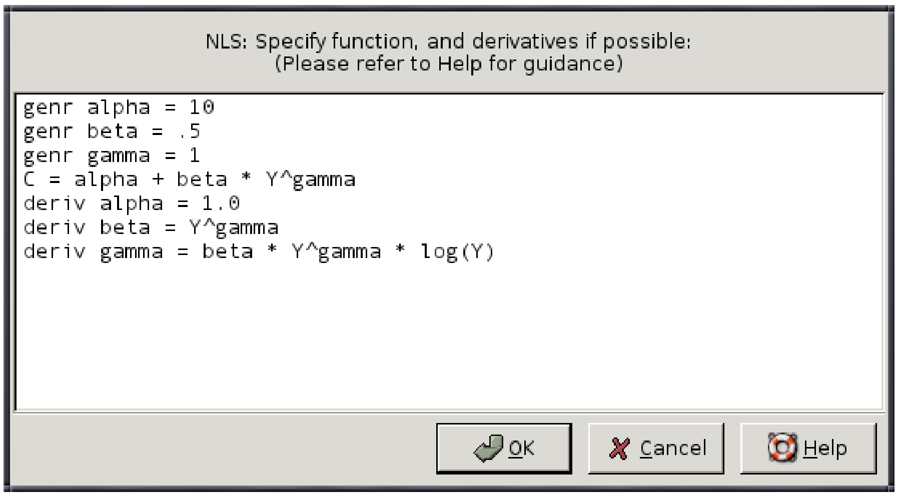
And in relation to the second example one might do:

```
ols y 0 x1 x2
genr alpha = coeff(0)
genr beta = coeff(x1)
```

## NLS dialog window

It is probably most convenient to compose the commands for NLS estimation in the form of a gretl script but you can also do so interactively, by selecting the item "Nonlinear Least Squares" under the Model menu. This opens a dialog box where you can type the function specification (possibly prefaced by genr lines to set the initial parameter values) and the derivatives, if available. An example of this is shown in Figure 7-1. Note that in this context you do not have to supply the `nls` and `end nls` tags.

**Figure 7-1. NLS dialog box**



NLS: Specify function, and derivatives if possible:
(Please refer to Help for guidance)

```
genr alpha = 10
genr beta = .5
genr gamma = 1
C = alpha + beta * Y^gamma
deriv alpha = 1.0
deriv beta = Y^gamma
deriv gamma = beta * Y^gamma * log(Y)
```

## Analytical and numerical derivatives

If you are able to figure out the derivatives of the regression function with respect to the parameters, it is advisable to supply those derivatives as shown in the examples above. If that is not possible, `gretl` will compute approximate numerical derivatives. The properties of the NLS algorithm may not be so good in this case (see the Section called *Numerical accuracy*).

If analytical derivatives are supplied, they are checked for consistency with the given nonlinear function. If the derivatives are clearly incorrect estimation is aborted with an error message. If the derivatives are "suspicious" a warning message is issued but estimation proceeds. This warning may sometimes be triggered by incorrect derivatives, but it may also be triggered by a high degree of collinearity among the derivatives.

Note that you cannot mix analytical and numerical derivatives: you should supply expressions for all of the derivatives or none.

## Controlling termination

The NLS estimation procedure is an iterative process. Iteration is terminated when a convergence criterion is met or when a set maximum number of iterations is reached, whichever comes first. The maximum number of iterations is `100*(k+1)` when analytical derivatives are given and `200*(k+1)` when numerical derivatives are used, where `k` denotes the number of parameters being estimated. The convergence criterion is that the relative error in the sum of squares, and/or the relative error between the the coefficient vector and the solution, is estimated to be no larger than some small value. This "small value" is by default the machine precision to the power 3/4, but it can be set with the `genr` command using the special variable `toler`. For example

```
genr toler = .0001
```

will relax the tolerance to 0.0001.

## Details on the code

The underlying engine for NLS estimation is based on the `minpack` suite of functions, available from netlib.org. Specifically, the following `minpack` functions are called:

| | |
|---|---|
| `lmder` | Levenberg–Marquandt algorithm with analytical derivatives |
| `chkder` | Check the supplied analytical derivatives |
| `lmdif` | Levenberg–Marquandt algorithm with numerical derivatives |
| `fdjac2` | Compute final approximate Jacobian when using numerical derivatives |
| `dpmpar` | Determine the machine precision |

On successful completion of the Levenberg–Marquandt iteration, a Gauss–Newton regression is used to calculate the covariance matrix for the parameter estimates. Since NLS results are asymptotic, there is room for debate over whether or not a correction for degrees of freedom should be applied when calculating the standard error of the regression (and the standard errors of the parameter estimates). For comparability with OLS, and in light of the reasoning given in Davidson and MacKinnon (1993), the estimates shown in `gretl` *do* use a degrees of freedom correction.

## Numerical accuracy

Table 7-1 shows the results of running the `gretl` NLS procedure on the 27 Statistical Reference Datasets made available by the U.S. National Institute of Standards and Technology (NIST) for testing nonlinear regression software.[1] For each dataset, two sets of starting values for the parameters are given in the test files, so the full test comprises 54 runs. Two full tests were performed, one using all analytical derivatives and one using all numerical approximations. In each case the default tolerance was used.[2]

Out of the 54 runs, `gretl` failed to produce a solution in 4 cases when using analytical derivatives, and in 5 cases when using numeric approximation. Of the four failures in analytical derivatives mode, two were due to non-convergence of the Levenberg–Marquandt algorithm after the maximum number of iterations (on `MGH09` and `Bennett5`, both described by NIST as of "Higher difficulty") and two were due to generation of range errors (out-of-bounds floating point values) when computing the Jacobian (on `BoxBOD` and `MGH17`, described as of "Higher difficulty" and "Average difficulty" respectively). The additional failure in numerical approximation mode was on `MGH10` ("Higher difficulty", maximum number of iterations reached).

The table gives information on several aspects of the tests: the number of outright failures, the average number of iterations taken to produce a solution and two sorts of measure of the accuracy of the estimates for both the parameters and the standard errors of the parameters.

For each of the 54 runs in each mode, if the run produced a solution the parameter estimates obtained by `gretl` were compared with the NIST certified values. We define the "minimum correct figures" for a given run as the number of significant figures to which the *least accurate* `gretl` estimate agreed with the certified value, for that run. The table shows both the average and the worst case value of this variable across all the runs that produced a solution. The same information is shown for the estimated standard errors.[3]

The second measure of accuracy shown is the percentage of cases, taking into account all parameters from all successful runs, in which the `gretl` estimate agreed with the cer-

1. For a discussion of `gretl`'s accuracy in the estimation of linear models, see Appendix C.
2. The data shown in the table were gathered from a pre-release build of `gretl` version 1.0.9, compiled with `gcc` 3.3, linked against `glibc` 2.3.2, and run under Linux on an i686 PC (IBM ThinkPad A21m).
3. For the standard errors, I excluded one outlier from the statistics shown in the table, namely `Lanczos1`. This is an odd case, using generated data with an almost-exact fit: the standard errors are 9 or 10 orders of magnitude smaller than the coefficients. In this instance `gretl` could reproduce the certified standard errors to only 3 figures (analytical derivatives) and 2 figures (numerical derivatives).

tified value to at least the 6 significant figures which are printed by default in the `gretl` regression output.

**Table 7-1. Nonlinear regression: the NIST tests**

| | Analytical derivatives | Numerical derivatives |
|---|---|---|
| Failures in 54 tests | 4 | 5 |
| Average iterations | 32 | 127 |
| Avg. of min. correct figures, parameters | 8.120 | 6.980 |
| Worst of min. correct figures, parameters | 4 | 3 |
| Avg. of min. correct figures, standard errors | 8.000 | 5.673 |
| Worst of min. correct figures, standard errors | 5 | 2 |
| Percent correct to at least 6 figures, parameters | 96.5 | 91.9 |
| Percent correct to at least 6 figures, standard errors | 97.7 | 77.3 |

Using analytical derivatives, the worst case values for both parameters and standard errors were improved to 6 correct figures on the test machine when the tolerance was tightened to 1.0e-14. Using numerical derivatives, the same tightening of the tolerance raised the worst values to 5 correct figures for the parameters and 3 figures for standard errors, at a cost of one additional failure of convergence.

Note the overall superiority of analytical derivatives: on average solutions to the test problems were obtained with substantially fewer iterations and the results were more accurate (most notably for the estimated standard errors). Note also that the six-digit results printed by `gretl` are not 100 percent reliable for difficult nonlinear problems (in particular when using numerical derivatives). Having registered this caveat, the percentage of cases where the results were good to six digits or better seems high enough to justify their printing in this form.

# 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 3 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 8 commands can be used: genr, ols, print, printf, sim, 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 = 100 * 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"
noecho
printf "alpha = %g\n", coeff(0)
printf "beta  = %g\n", coeff(x1)
printf "gamma = %g\n", coeff(x2)
```

Example 8-3 (kindly contributed by Riccardo “Jack” Lucchetti of Ancona University) shows how a loop can be used to estimate an ARMA model, exploiting the “outer product of the gradient” (OPG) regression discussed by Davidson and MacKinnon in their *Estimation and Inference in Econometrics*.

**Example 8-3. ARMA 1, 1**

```
open arma.gdt

genr c = 0
genr a = 0.1
genr m = 0.1

genr e = const * 0.0
genr de_c = e
genr de_a = e
genr de_m = e

genr crit = 1
loop while crit > 1.0e-9

   # one-step forecast errors
   genr e = y - c - a*y(-1) - m*e(-1)

   # log-likelihood
   genr loglik = -0.5 * sum(e^2)
   print loglik

   # partials of forecast errors wrt c, a, and m
   genr de_c = -1 - m * de_c(-1)
   genr de_a = -y(-1) -m * de_a(-1)
   genr de_m = -e(-1) -m * de_m(-1)

   # partials of l wrt c, a and m
   genr sc_c = -de_c * e
   genr sc_a = -de_a * e
   genr sc_m = -de_m * e

   # OPG regression
   ols const sc_c sc_a sc_m

   # Update the parameters
   genr dc = coeff(sc_c)
   genr c = c + dc
   genr da = coeff(sc_a)
   genr a = a + da
   genr dm = coeff(sc_m)
   genr m = m + dm

   printf "  constant        = %.8g (gradient = %#.6g)\n", c, dc
   printf "  ar1 coefficient = %.8g (gradient = %#.6g)\n", a, da
   printf "  ma1 coefficient = %.8g (gradient = %#.6g)\n", m, dm

   genr crit = $T - $ess
   print crit
endloop

genr se_c = stderr(sc_c)
genr se_a = stderr(sc_a)
genr se_m = stderr(sc_m)

noecho
print "
printf "constant = %.8g (se = %#.6g, t = %.4f)\n", c, se_c, c/se_c
printf "ar1 term = %.8g (se = %#.6g, t = %.4f)\n", a, se_a, a/se_a
printf "ma1 term = %.8g (se = %#.6g, t = %.4f)\n", m, se_m, m/se_m
```

## 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 8-4 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 8-4. Indexed loop example**

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

The `smpl` command in the above example illustrates the use of the variable `i` within a loop. In addition, you can use the expression `$i`: in this case the value of `i` will be substituted before the command is evaluated. This enables you to construct strings (for example, variable names) within the loop, as in Example 8-5.

**Example 8-5. Second indexed loop example**

```
open bea.dat
loop for i=1987..2001
  genr V = COMP$i
  genr TC = GOC$i - PBT$i
  genr C = TC - V
  ols PBT$i const TC V
endloop
```

The first time round this loop `V` will be set to equal `COMP1987` and the dependent variable for the `ols` will be `PBT1987`, and so on.

# 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 4 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 4) is not in the default system database directory, you must specify the full path.

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

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

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

§ The command to launch `gnuplot`.

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

§ The command with which to view TeX DVI files.

§ 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.

§ The font to be used for menus and other messages. (Note: this item is not present when `gretl` is compiled for the `gnome` desktop, since the choice of fonts is handled centrally by `gnome`.)

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. If your native language setting is not English and the local decimal point character is not the period (“`.`”), unchecking “Use locale setting for decimal point” will make `gretl` use the period regardless.

1. On Linux, 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`.

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 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. If you set the default to `.dat` then data files will be saved in the “traditional” format (see Chapter 4). Also under the “Data files” tab you can select the action for the little folder icon on the toolbar: whether it should open a listing of the data files associated with Ramanathan’s textbook, or those associated with Wooldridge’s text.

Under the “General” tab you may select the algorithm used by `gretl` for calculating least squares estimates. The default is Cholesky decomposition, which is fast, relatively economical in terms of memory requirements, and accurate enough for most purposes. The alternative is QR decomposition, which is computationally more expensive and requires more temporary storage, but which is more accurate. You are unlikely to need the extra accuracy of QR decomposition unless you are dealing with very ill-conditioned data and are concerned with coefficient or standard error values to more than 7 digits of precision.[2]

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 4). 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 *)
```

2. The option of using QR decomposition can also be activated by setting the environment variable `GRETL_USE_QR` to any non-NULL value.

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 | 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 cannot enter 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.

### coeffsum

| | |
|---|---|
| Arguments: | *`indepvars`* |
| Example: | `coeffsum xt xt_1 xr_2` |

Must follow an regression. Calculates the sum of the coefficients on the variables in the *`indepvars`* list. Prints this sum along with its standard error and the p-value for the null hypothesis that the sum is zero.

### 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.

### coint2

| | |
|---|---|
| Arguments: | *`order depvar indepvar`* |
| Examples: | `coint2 2 y x` |
| | `coint2 4 y x1 x2 -o` |

Carries out the Johansen trace test for cointegration among the listed variables for the given order. For details of this test see Hamilton (1994, Chapter 19). If the `-o` flag is given the results of the various auxiliary regressions are printed.

### 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 $t$th 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.

### critical

| | |
|---|---|
| Arguments: | *`dist param1`* [ *`param2`* ] |
| Examples: | `critical t 20` |
| | `critical X 5` |
| | `critical F 3 37` |

If *`dist`* is `t`, `X` or `F`, prints out the critical values for the student’s $t$, chi-square or $F$ distribution respectively, for the common significance levels and using the specified degrees of freedom, given as *`param1`* for t and chi-square, or *`param1`* and *`param2`* for $F$. If *`dist`* is `d`, prints the upper and lower values of the Durbin-Watson statistic at 5 percent significance, for the given number of observations, *`param1`*, and for the range of 1 to 5 explanatory 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 printed. See Chapter 7 of Greene's *Econometric Analysis* for details.

### data

Argument: *varlist*

Reads the variables in *varlist* from a database (gretl or RATS 4.0), which must have been opened previously using the `open` command. In addition, a data frequency and sample range must be established using the the `setobs` and `smpl` commands prior to using this command. Here is a full example:

```
open macrodat.rat
setobs 4 1959:1
smpl ; 1999:4
data GDP_JP GDP_UK
```

These commands open a database named `macrodat.rat`, establish a quarterly data set starting in the first quarter of 1959 and ending in the fourth quarter of 1999, and then import the series named `GDP_JP` and `GDP_UK`.

If the series to be read are of higher frequency than the working data set, you must specify a compaction method as below:

```
data (compact=average) LHUR PUNEW
```

The four available compaction methods are "average" (takes the mean of the high frequency observations), "last" (uses the last observation), "first" and "sum".

### delete

Argument: *varlist*

Removes the listed variables (given by name or number) from the dataset. *Use with caution*: no confirmation is asked, and any variables with higher ID numbers will be re-numbered.

If no *varlist* is given with this command, it deletes the last (highest numbered) variable from the dataset.

### 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)`.

### end

Ends a block of commands of some sort. For example, `end system` terminates an equation system.

### endloop

Terminates a simulation loop. See `loop`.

### equation

Arguments: *depvar indepvars*

Example: `equation y x1 x2 x3 const`

Specifies an equation within a system of equations (see the `system` command). The sytax for specifying an equation is the same as that for, e.g., the `ols` command.

### eqnprint

Options: `[ -o ] [ -f` *`filename`* `]`

Must follow the estimation of a model via OLS. Prints the estimated model in the form of a LaTeX equation. If a filename is specified using the `-f` flag output goes to that file, otherwise it goes 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. See also the `tabprint` command.

If the `-o` flag is given the LaTeX file is a complete document, ready for processing; otherwise it must be included in a document.

### 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), `>`, `<`, `=`, `>=` (greater than or equal), `<=` (less than or equal) and `!=` (not equal). 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: `max` (maximum value in a series), `min` (minimum), `mean` (arithmetic mean), `median`, `var` (variance) `sd` (standard deviation), `sst` (sum of squared deviations from the mean), `sum`, `cov` (covariance), `corr` (correlation coefficient), `pvalue`

§ Time-series functions: `diff` (first difference), `ldiff` (log-difference, or first difference of natural logs). To generate lags of a variable `x`, use the syntax `x(-N)`, where `N` represents the desired lag length; to generate leads, use `x(+N)`;

§ Miscellaneous: `cum` (cumulate), `sort`, `uniform`, `normal`, `misszero` (replace the missing observation code in a given series with zeros), `zeromiss` (the inverse operation to `misszero`), `nobs` (return the number of valid observations in a given data series).

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 (0–1) 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 5 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 *R*-squared |
| `$T` | number of observations used |
| `$df` | degrees of freedom |
| `$trsq` | *TR*-squared (sample size times *R*-squared) |
| `$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.

The internal series `uhat` and `yhat` hold, respectively, the residuals and fitted values from the last regression.

Three other "internal" variables are available: `$nobs` holds the number of observations in the current sample range (note that this may or may not equal the value of `$T`, the number of observations used in estimating the last model). The variable `t` serves as an index of the observations. Thus for instance `genr dum = (t=15)` will generate a dummy variable that has value 1 for observation 15, 0 otherwise. The variable `$pd` holds the frequency or periodicity of the data (e.g. 4 for quarterly data).

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 |

| **Command** | **Comment** |
|---|---|
| `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()` | 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 ]` |
| Addendum: | `{` *literal gnuplot commands* `}` |
| Alternate form: | `-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 "alternate form" (with `-z`) *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`.)

A further option to this command is available: following the specification of the variables to be plotted and the option flag (if any), you may add literal gnuplot commands to control the appearance of the plot (for example, setting the plot title and/or the axis ranges). These commands should be enclosed in braces, and each gnuplot command must be terminated with a semi-colon. A backslash may be used to continue a set of gnuplot commands over more than one line. Here is an example of the syntax:

```
{ set title 'My Title'; set yrange [0:1000]; }
```

## graph

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

ASCII graphics. The *yvars* (which may be given by name or number) are graphed against *xvar* using ASCII symbols. The `-o` flag will graph with 40 rows and 60 columns. Without it, the graph will be 20 by 60 (for screen output). 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) "jackknife" 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.

### if

Flow control for command execution. The syntax is

```
if condition
 commands
else
 commands
endif
```

*`condition`* must be a Boolean expression, for the syntax of which see the Section called *genr*. The `else` block is optional; `if ... endif` blocks may be nested.

### 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 4 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.)

### label

Arguments: *`varname`* `-d "`*`Descriptive string`*`" -n "`*`display name`*`"`

Sets the descriptive label for the given variable (if the `-d` flag is given, followed by a string in double quotes) and/or the "display name" for the variable (if the `-n` flag is given, followed by a quoted string). If a variable has a display name, this is used when generating graphs.

### 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.

### lad

Arguments: *`depvar indepvars`*

Calculates a regression that minimizes the sum of the absolute deviations of the observed from the fitted values of the dependent variable. Coefficient estimates are derived using the Barrodale–Roberts simplex algorithm; a warning is printed if the solution is not unique. Standard errors are derived using the bootstrap procedure with 500 drawings.

## 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` $= \ln(y_t) - \ln(y_{t-1})$ .

## leverage

Must immediately follow an `ols` command. Calculates the leverage ($h$, which must lie in the range 0 to 1) for each data point in the sample on which the previous model was estimated. Displays the residual ($u$) for each observation along with its leverage and a measure of its influence on the estimates, $uh/(1-h)$ . "Leverage points" for which the value of $h$ exceeds $2k/n$ (where $k$ is the number of parameters being estmated and $n$ is the sample size) are flagged with an asterisk. For details on the concepts of leverage and influence see Davidson and MacKinnon (1993, Chapter 2).

## 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. In the case of White's test, only the squared independent variables are used and not their cross products. In the case of the autocorrelation test, if the p-value of the LMF statistic is less than 0.05 then serial correlation-robust standard errors are calculated and displayed. For details on the calculation of these standard errors see Wooldridge (2002, Chapter 12).

## 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, Chapter 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, Chapter 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 7 commands can be used: `genr`, `ols`, `print`, `sim`, `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.)

### mpols

| | |
|---|---|
| Arguments: | *`depvar indepvars`* |
| Examples: | `mpols 1 0 2 4 6 7` |
| | `mpols y 0 x1 x2 x3` |

Computes OLS estimates with *`depvar`* as the dependent variable and *`indepvars`* as the list of independent variables, using multiple precision floating-point arithmetic. The variables may be specified by name or number; use the number zero for a constant term. This command is available only if `gretl` is compiled with support for the Gnu Multiple Precision library (GMP).

To estimate a polynomial fit, using multiple precision arithmetic to generate the required powers of the independent variable, use the form, e.g. `mpols y 0 x ; 2 3 4` This does a regression of `y` on `x`, `x` squared, `x` cubed and `x` to the fourth power. That is, the numbers (which must be positive integers) to the right of the semicolon specify the powers of `x` to be used. If more than one independent variable is specified, the last variable before the semicolon is taken to be the one that should be raised to various powers.

### 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 `incomepc` as the product of `income` and `invpop`, and `expendpc` as `expend` times `invpop`.

## nls

Performs Nonlinear Least Squares (NLS) estimation using a modified version of the Levenberg–Marquandt algorithm. The user must supply a function specification. The parameters of this function must be declared and given starting values (using the `genr` command) prior to estimation. Optionally, the user may specify the derivatives of the regression function with respect to each of the parameters; if analytical derivatives are not supplied, a numerical approximation to the Jacobian is computed.

It is easiest to show what is required by example. The following is a complete script to estimate the nonlinear consumption function set out in William Greene's *Econometric Analysis*, in chapter 11 of the 4th edition, or chapter 9 of the 5th. (The numbers to the left of the lines are for reference and are not part of the commands.)

```
1   open greene11_3.gdt
2   ols C 0 Y
3   genr alpha = coeff(0)
4   genr beta = coeff(Y)
5   genr gamma = 1.0
6   nls C = alpha + beta * Y^gamma
7   deriv alpha = 1
8   deriv beta = Y^gamma
9   deriv gamma = beta * Y^gamma * log(Y)
10  end nls
```

It is often convenient to initialize the parameters by reference to a related linear model; that is accomplished here on lines 2 to 5. The parameters alpha, beta and gamma could be set to any initial values (not necessarily based on a model estimated with OLS), although convergence of the NLS procedure is not guaranteed for an arbitrary starting point.

The actual NLS commands occupy lines 6 to 10. On line 6 the `nls` command is given: a dependent variable is specified, followed by an equals sign, followed by a function specification. The syntax for the expression on the right is the same as that for the `genr` command. The next three lines specify the derivatives of the regression function with respect to each of the parameters in turn. Each line begins with the keyword `deriv`, gives the name of a parameter, an equals sign, and an expression whereby the derivative can be calculated (again, the syntax here is the same as for `genr`). These `deriv` lines are optional, but it is recommended that you supply them if possible. Line 10, `end nls`, completes the command and calls for estimation.

For further details on NLS estimation please see Chapter 7.

## noecho

Suppresses the echoing of commands and comments (other than those inserted using the `print` command) when executing a command script.

### 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 | -q ]`

Examples:
```
ols 1 0 2 4 6 7
ols -o y 0 x1 x2 x3
ols -q y 0 x1 x2 x3
```

Computes ordinary least squares (OLS) 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 in addition to the various statistics that are printed by default. The `-q` flag suppresses the printing of output.

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).

This command can also be used to open a database (gretl or RATS 4.0) for reading, in which case it should be followed by the `data` command to extract particular series from the database.

## outfile

| | |
|---|---|
| Argument: | [ *`flag`* ] [ *`filename`* ] |
| Example: | outfile -w regress.txt |
| | ols y 0 x1 x2 |
| | outfile -c |

Divert output to *`filename`*, until further notice. Use the flag `-a` to append output to an existing file, or `-w` to start a new file (or overwrite an existing one). Only one file can be opened in this way at any given time.

The `-c` flag is used to close an output file that was previously opened as above. Output will then revert to the default stream.

In the example above, the results of one regression are written to the file `regress.txt`, which is then closed.

## pca

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

Rudimentary Principal Components Analysis. Prints the eigenvalues of the correlation matrix for the variables in *`varlist`* along with the proportion of the joint variance accounted for by each component. Also prints the corresponding eigenvectors ("component loadings").

If the `-o` flag is given, components with eigenvalues greater than 1.0 are saved to the dataset as variables, with names `PC1`, `PC2` and so on. These artificial variables are formed as the sum of (component loading) times (standardized `Xi`), where `Xi` denotes the *i*th variable in *`varlist`*.

If the `-a` flag is given, all of the components are saved as described above.

## panel

Argument: `-s` or `-c`

Request that the current data set be interpreted as a panel (pooled cross section and time series). With no flag, or with the `-s` flag, the data set is taken to be in the form of stacked time series (successive blocks of data contain time series for each cross-sectional unit). With the `-c` flag, the data set is read as stacked cross-sections (successive blocks contain cross sections for each time period). See also the Section called *setobs*.

## 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 | -t ]` or [ *`string`* ]

Examples: `print`

`print x y`

`print 1 2 3 -o`

If *`varlist`* is given, prints the values of the specified variables; if no list is given, 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. If the `-t` flag is given the data are printed by variable to 10 significant digits.

If the argument to `print` is a literal string (which must start with a double-quote, "), the string is printed as is.

## printf

Prints scalar values under the control of a format string (providing a small subset of the `printf()` statement in the C programming language). Recognized formats are `%g` and `%f`, in each case with the various modifiers available in C. Examples: the format `%.10g` prints a value to 10 significant figures; `%12.6f` prints a value to 6 decimal places, with a width of 12 characters.

The format string itself must be enclosed in double quotes. The values to be printed must follow the format string, separated by commas. These values should take the form of either (a) the names of variables in the dataset, or (b) expressions that are valid for the `genr` command. The following example prints the values of two variables plus that of a calculated expression:

```
ols 1 0 2 3
genr b = coeff(2)
genr se_b = stderr(2)
printf "b = %.8g, standard error %.8g, t = %.4f\n", b, se_b, b/se_b
```

The maximum length of a format string is 127 characters. The escape sequences `\n` (newline), `\t` (tab), `\v` (vertical tab) and `\\` (literal backslash) are recognized. To print a literal percent sign, use `%%`.

## 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.

## rename

Arguments: *varnumber newname*

Changes the name of the variable with identification number *varnumber* to *newname*. The *varnumber* must be between 1 and the number of variables in the dataset. The new name must be of 8 characters maximum, must start with a letter, and must be composed of only letters, digits, and the underscore character.

## reset

Must immediately follow the estimation of a model via OLS. Carries out Ramsey's RESET test for model specification (non-linearity) by adding the square and the cube of the fitted values to the regression and calculating the *F* statistic for the null hypothesis that the parameters on the two added terms are zero.

## 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.

## rmplot

| | |
|---|---|
| Argument: | *`varname`* |

Range–mean plot: this command creates a simple graph to help in deciding whether a time series, *y*(t), has constant variance or not. We take the full sample t=1,...,T and divide it into small subsamples of arbitrary size *k*. The first subsample is formed by *y*(1),...,*y*(k), the second is *y*(k+1), ..., *y*(2k), and so on. For each subsample we calculate the sample mean and range (= maximum minus minimum), and we construct a graph with the means on the horizontal axis and the ranges on the vertical. So each subsample is represented by a point in this plane. If the variance of the series is constant we would expect the subsample range to be independent of the subsample mean; if we see the points approximate an upward-sloping line this suggests the variance of the series is increasing in its mean; and if the points approximate a downward sloping line this suggests the variance is decreasing in the mean.

Besides the graph, gretl displays the means and ranges for each subsample, along with the slope coefficient for an OLS regression of the range on the mean and the p-value for the null hypothesis that this slope is zero. If the slope coefficient is significant at the 10 percent significance level then the fitted line from the regression of range on mean is shown on the graph.

## 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 5.

## 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.

## 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)` |
| `sim 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 current sample range, or for the range *startobs* through *endobs* if these optional arguments are given. 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 be either numerical constants or variable names previously defined; these terms may be prefixed with minus sign.

Note that as of gretl 1.2.0, `genr` (see the Section called *genr* above) works dynamically, and you will probably find it easier than `sim` for creating an autoregressive series. For example

```
genr y = 0 * const
genr y = .8*y(-1) -.2*y(-2) + normal()
```

now achieves the same object as (and with greater clarity than):

```
genr y = 0 * const
genr u = normal()
sim y u .8 -.2
```

## smpl

| | |
|---|---|
| Arguments: | `startobs endobs` |
| | `smpl +i -j` |
| | `smpl -o dummyvar` |
| | `smpl -o` |
| | `smpl -r condition` |
| | `smpl full` |

Resets the sample range. In the first form *startobs* and *endobs* must be consistent with the periodicity of the data. In the second form, the integers *i* and *j* are taken as offsets relative to the existing sample range. In the third 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 fourth form, `smpl -o`, drops all observations for which values of one or more variables are missing. The fifth form (`-r`) restricts the sample to observations that satisfy the given (Boolean) condition. The last form, `smpl full`, restores the full data range.

| | |
|---|---|
| `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 +1 -1` | advance the starting observation by one; move the ending observation back one |
| `smpl -o dum1` | draw sample of observations where `dum1=1` |
| `smpl -r income > 30000` | sample cases where `income` has a value greater than 30000. |

The intenal variable `obs` may be used in the last-noted form of `smpl` above, to exclude particular observations from the sample. For example, `smpl -r obs!=4` will drop just the fourth observation. If the data points are identified by labels, `smpl -r obs!="USA"` will drop the observation with label “USA”.

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 data series in the current data set will be stored. Note that any scalar variables will not be saved automatically: if you wish to save scalars you must explicitly list them in *varlist*.

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.

### system

Arguments: type=*systype* [ savevars=*vars* ]

Examples:
```
system type=sur
system type=sur save=resids
system type=sur save=resids,fitted
```

Starts a system of equations. At present the only type of system supported is `sur` (Seemingly Unrelated Regressions). In the optional `save=` field of the command you can specify whether to save the residuals (`resids`) and/or the fitted values (`fitted`). The system must contain at least two equations specified using the `equation` command, and it must be terminated with the line `end system`.

### tabprint

Options: `[ -o ] [ -f` *`filename`* `]`

Must follow the estimation of a model via OLS. Prints the estimated model in the form of a LaTeX tabular environment. If a filename is specified using the `-f` flag output goes to that file, otherwise it goes 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. See also the `eqnprint` command.

If the `-o` flag is given the LaTeX file is a complete document, ready for processing; otherwise it must be included in a document.

### 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 varlist`* `;` *`detlist`*

Example: `var 4 x1 x2 x3 ; const time`

Sets up and estimates (using OLS) a vector autoregression (VAR). The first argument specifies the lag order, then follows the setup for the first equation. Don't include lags among the elements of *`varlist`* — they will be added automatically. The semi-colon separates the stochastic variables, for which *`order`* lags will be included, from deterministic terms in *`detlist`*, such as the constant, a time trend, and dummy variables.

In fact, gretl is able to recognize the more common deterministic variables (constant, time trend, dummy variables with no values other than 0 and 1) as such, so these do not have to placed after the semi-colon. More complex deterministic variables (e.g. a time trend interacted with a dummy variable) must be put after the semi-colon.

A separate regression is run for each variable in varlist. Output for each equation includes *F*-tests for zero restrictions on all lags of each of the variables; an *F*-test for the significance of the maximum lag; forecast variance decompositions; and impulse response functions.

The variance decompositions and impulse responses are based on the Cholesky decomposition of the contemporaneous covariance matrix, and in this context the order in which the (stochastic) variables are given matters. The first variable in the list is assumed to be "most exogenous" within-period.

### 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 |
| Nonlinear Least Squares (`nls`) | Nonlinear models |
| Logit (`logit`) | Binary dependent variable (logistic distribution) |

| **Estimator** | **Comment** |
|---|---|
| Probit (`probit`) | Binary dependent variable (normal distribution) |
| Least Absolute Deviation (`lad`) | Alternative to Least Squares |
| Rank Correlation (`spearman`) | Correlation with ordinal data |
| Pooled OLS (`pooled`) | OLS estimation for pooled cross-section, time series data |
| Multiple precision OLS (`mpols`) | OLS estimation using multiple precision arithmetic |

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` |
| Nonlinearity (logs) | `lmtest` |
| Nonlinearity (Ramsey's RESET) | `reset` |
| Heteroskedasticity (White's test) | `lmtest` |
| Influential observations | `leverage` |
| 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` |

# Chapter 11. Troubleshooting gretl

## Bug reports

Bug reports are welcome. I believe 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?

## Auxiliary programs

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, windvi | 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

## Gretl at the console

The `gretl` package includes the command-line program `gretlcli`. On Linux it can be run from the console, or in an xterm (or similar). Under MS Windows it can be run in a console window (sometimes inaccurately called 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 Ramanathan's ESL

`gretlcli` inherits its basic command syntax from Ramu Ramanathan's `ESL`, and command scripts developed for `ESL` should be usable 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 Linux.) 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 basename 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 5 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:

```
G0M910  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, `A` for annual, `B` for business-daily (daily with five days per week) and `D` for daily (seven days per week). 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. In the case of daily data the starting and ending dates should be given in the form `YYYY/MM/DD`. 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). As of this writing there are two main variants of the GTK libraries: the 1.2 series and the 2.0 series which was launched in summer 2002. These variants are mutually incompatible. `gretl` can be built using either one — the source code package includes two sub-directories, `gui` for GTK 1.2 and `gui2` for GTK 2.0. I recommend use of GTK 2.0 if it is available, since it offers many enhancements over GTK 1.2.

`gretl` calls gnuplot for graphing. You can find gnuplot at gnuplot.info. As of this writing the most recent official release is 3.7.3 (of December, 2002). If you are comfortable compiling source code I would recommend installing gnuplot 3.8j0 or higher (source package available from sourceforge.net/projects/gnuplot. The MS Windows version of `gretl` comes with a Windows version gnuplot 3.8j0; the gretl website also offers an rpm of gnuplot 3.8j0 for x86 Linux systems.

Some features of `gretl` 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 2.0 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. The documentation of the libgretl API is by no means complete, but you can find some details by following the link "Libgretl API docs" on the `gretl` homepage.

# Appendix C. Numerical accuracy

gretl uses double-precision arithmetic throughout — except for the multiple-precision plugin invoked by the menu item "Model/High precision OLS" which represents floating-point values using a number of bits given by the environment variable GRETL_MP_BITS (default value 256). The normal equations of Least Squares are by default solved via Cholesky decomposition, which is accurate enough for most purposes (with the option of using QR decomposition instead). The program has been tested rather thoroughly on the statistical reference datasets provided by NIST (the U.S. National Institute of Standards and Technology) and a full account of the results may be found on the gretl website (follow the link "Numerical accuracy").

In October 2002 I had a useful exchange with Giovanni Baoicchi and Walter Distaso, who were writing a review of gretl for the *Journal of Applied Econonetrics*, and James MacKinnon, software review editor for the journal.[1] and I am grateful to Baoicchi and Disasto for their careful examination of the program, which prompted the following modifications.

1. The reviewers pointed out that there was a bug in gretl's "p-value finder", whereby the program printed the complement of the correct probability for negative values of $z$. This was fixed in version 0.998 of the program (released July 9, 2002).
2. They also noted that the p-value finder produced inaccurate results for extreme values of $x$ (e.g. values of around 8 to 10 in the $t$ distribution with 100 degrees of freedom). This too was fixed in gretl version 0.998, with a switch to more accurate probability distribution code.
3. The reviewers noted a flaw in the presentation of regression coefficients in gretl, whereby some coefficients could be printed to an unacceptably small number of significant figures. This was fixed in version 0.999 (released August 25, 2002): now all the statistics associated with a regression are printed to 6 significant figures.
4. It transpired from the reviewer's tests that the numerical accuracy of gretl on MS Windows was less than on Linux, where I had done my testing. For example, on the Longley data — a well-known "ill-conditioned" dataset often used for testing econometrics programs — the Windows version of gretl was getting some coefficients wrong at the 7th digit while the same coefficients were correct on Linux. This anomaly was fixed in gretl version 1.0pre3 (released October 10, 2002).

The current version of the gretl source code package contains a tests subdirectory, with a test suite based on the NIST datasets. This is invoked if you do make check in the top source directory. You are warned if the numerical accuracy falls short of standard. Please consult the file README in the tests directory for details.

The NIST test suite is also distributed with the MS Windows version of gretl. You can run the tests by invoking the program nisttest.exe.

As mentioned above, all regression statistics are printed to 6 significant figures in the current version of gretl (except when the multiple-precision plugin is used, then results are given to 12 figures). If you want to examine a particular value more closely, first save it (see the Section called *genr* in Chapter 10) then print it using print -t (see Chapter 10). This will show the value to 10 digits.

1. This review has since been published; see Baoicchi and Distaso (2003).

# Appendix D. 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 such as (binomial) logit and probit and Least Absolute Deviations. 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 E. Listing of URLs

Below is a listing of the full URLs of websites mentioned in the text.

*Census Bureau, Data Extraction Service*

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

*Estima (RATS)*

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

*Gnome desktop homepage*

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

*GNU Multiple Precision (GMP) library*

`http://swox.com/gmp/`

*GNU Octave homepage*

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

*GNU R homepage*

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

*GNU R manual*

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

*Gnuplot homepage*

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

*Gnuplot manual*

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

*Gretl data page*

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

*Gretl homepage*

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

*GTK+ homepage*

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

*GTK+ port for win32*

`http://www.gimp.org/~tml/gimp/win32/`

*Gtkextra homepage*

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

*InfoZip homepage*

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

*JRSoftware*

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

*Mingw (gcc for win32) homepage*

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

*Minpack*

`http://www.netlib.org/minpack/`

*Penn World Table*

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

*Readline homepage*

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

*Readline manual*

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

*Xmlsoft homepage*

`http://xmlsoft.org/`

# Bibliography

Baiocchi, G. and Distaso, W. (2003) "GRETL: Econometric software for the GNU generation", *Journal of Applied Econometrics*, 18, pp. 105–10.

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.

Davidson, R. and MacKinnon, J. G. (1993) *Estimation and Inference in Econometrics*, New York: Oxford University Press.

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

Hamilton, James D. (1994) *Time Series Analysis*, Princeton, NJ: Princeton University Press.

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.

Wooldridge, Jeffrey M. (2002) *Introductory Econometrics, A Modern Approach*, 2nd edition, Mason, Ohio: South-Western.