Using **gretl** for Principles of Econometrics, 4th Edition
Version 1.0[1]

Lee C. Adkins
Professor of Economics
Oklahoma State University

August 18, 2011

[1]Visit http://www.LearnEconometrics.com/gretl.html for the latest version of this book. Also, check the errata (page 455) for changes since the last update.

License

Preface

The previous edition of this manual was about using the software package called **gretl** to do various econometric tasks required in a typical two course undergraduate or masters level econometrics sequence. This version tries to do the same, but several enhancements have been made that will interest those teaching more advanced courses. I have come to appreciate the power and usefulness of **gretl**'s powerful scripting language, now called **hansl**. **Hansl** is powerful enough to do some serious computing, but simple enough for novices to learn. In this version of the book, you will find more information about writing functions and using loops to obtain basic results. The programs have been generalized in many instances so that they could be adapted for other uses if desired. As I learn more about **hansl** specifically and programming in general, I will no doubt revise some of the code contained here. Stay tuned for further developments.

As with the last edition, the book is written specifically to be used with a particular textbook, *Principles of Econometrics, 4th edition* (*POE4*) by Hill, Griffiths, and Lim. It could be used with many other introductory texts. The data for all of the examples used herein are available as a package from my website at `http://www.learneconometrics.com/gretl.html`. If you are unfamiliar with **gretl** and are interested in using it in class, Mixon Jr. and Smith (2006) and Adkins (2011*a*) have written a brief review of **gretl** and how it can be used in an undergraduate course that you may persuade you to give it a try.

The chapters are arranged in the order that they appear in *Principles of Econometrics*. Each chapter contains a brief description of the basic models to be estimated and then gives you the specific instructions or **gretl** code to reproduce (nearly) all of the examples in the book. Where appropriate, I've added a bit of pedagogical material that complements what you'll find in the text. I've tried to keep this to a minimum since this is not supposed to serve as a substitute for your text book. The best part about this manual is that it, like **gretl**, is free. It is being distributed in Adobe's pdf format and I will make corrections to the text as I find errors.

Gretl's ability to process user written functions greatly expands the usefulness of the application. In several of the chapters functions are used to estimate models, select models, and to compute various statistics. The scripting language, continues to evolve in useful ways, becoming more transparent in use and more functional. Though not explored in this boo, the ability to give function writers access to the basic GUI and to package things into bundles is s very exciting development.

Functions can be shared and imported easily through **gretl**, especially if you are connected to the internet. If **gretl** doesn't do what you want it to now, stay tuned. It soon may. If recent activity is any indication, I am confident that the the **gretl** team will continue to improve this already very useful application. I hope that this manual is similarly useful to those using *Principles of Econometrics.*

There are some significant changes in the 4th edition of *POE* and that means there are some changes in this book from the previous edition. As in the previous edition of this e-book, I have attempted to provide **gretl** instructions for each and every example in the book. My solutions are not necessarily the most elegant. In some cases elegance gives way to simplicity of programming, especially with respect to the types of students who are likely to be using this book. I have made an effort to generalize some of the script so that it will be easier to adapt to new needs. I've also made liberal uses of loops and functions. These are powerful tools and a thorough understanding of them can take your **gretl** and econometric skills to the next level. Feel free to send suggestions.

Another change in this version of the book is that I've made some effort to generalize some of the scripts. Although that should make it easier to generalize them to a new use, it does mean that they have become a little more complicated. A heavy reliance on user written functions is evident. I invite users to take the time to work through these in order to advance your programming and econometric skills.

To make things easier to find in the book, I have added an index. In the pdf, you can click on the page number listed in the index and be taken to the relevant spot in the text. Also, the figure numbers, equation numbers, and citations are also 'hot' and can be used in this fashion as well. Since some may prefer to print the manual out rather than work from the .pdf, I opted to make the 'hot' links black in color, which disguises their functionality.

Once again, I want to thank the **gretl** team of Allin Cottrell and Riccardo "Jack" Lucchetti for putting so much effort into **gretl**. It is a wonderful program for **teaching** and **doing** econometrics. It has many capabilities beyond the ones I discuss in this book and other functions are added regularly. Also, Jack has kindly provided me with suggestions and programs that have made this much better than it would have been otherwise. Any remaining errors are mine alone.

I also want to thank my good friend and colleague Carter Hill for suggesting I write this and Oklahoma State University and our College of Business for continuing to pay me while I work on it.

Contents

List of Figures

Chapter 1

Introduction

In this chapter you will be introduced to some of the basic features of **gretl**. You'll learn how to install it, how to get around the various windows in **gretl**, and how to import data. At the end of the chapter, you'll be introduced to **gretl**'s powerful language.

1.1 What is Gretl?

Gretl is an acronym for Gnu Regression, Econometrics and Time-series Library. It is a software package for doing econometrics that is easy to use and powerful. It features a very user-friendly interface that makes it snap to use in classroom. Its flexibility, extensibility, and accuracy make it well-suited for research as well. **Gretl** is distributed as free software that can be downloaded from `http://gretl.sourceforge.net` and installed on your personal computer. Unlike software sold by commercial vendors (SAS, Eviews, Shazam to name a few) you can redistribute and/or modify **gretl** under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation. That means that you are free to patch or extend **gretl** as you see fit.

Gretl comes with many sample data files and its internet capabilities give you access to several very useful databases served by Wake Forest University. From the **gretl** web site, you can download and install sample data sets from many of the leading textbooks in econometrics, including the one that this book is based on, *Principles of Econometrics* by Hill et al. (2011).

Gretl offers a full range of least-squares based estimators, either for single equations and for systems, including vector autoregressions and vector error correction models. Several specific maximum likelihood estimators (e.g. probit, ARIMA, GARCH) are also provided natively; more advanced estimation methods can be implemented by the user via generic maximum likelihood or nonlinear GMM. **Gretl** uses a separate Gnu program called ***gnuplot*** to generate graphs and is capable of generating output in LaTeX format. **Gretl** is under constant development so you can probably expect some bugs, but in my experience it is quite stable to use with my Windows and Ubuntu Linux systems. The main developers, Allin Cottrell and Jack Lucchetti, participate daily in discussions on the **gretl** forums and quickly sort out any bugs that are reported.

Which brings me to the final plug for **gretl**, which is inspired by its openness. As seen with a lot of the better quality open source software, a community of developers and users are woven together via active user and developer forums. The input from their many participants helps to make **gretl** quite dynamic. If **gretl** will not estimate what you want today, tune-in tomorrow and someone may have written the code to estimate your econometric problem.

Furthermore, **gretl** is enhancing its scripting language to facilitate sophisticated add-ons to its basic functionality. In short, **gretl** is quickly becoming software worth getting to know for research as well as for pedagogical uses.

1.1.1 Installing Gretl

To install **gretl** on your system, you will need to download the appropriate executable file for the computer platform you are using. For Microsoft Windows users the appropriate site is `http://gretl.sourceforge.net/win32/`. One of the nice things about **gretl** is that Mac OS X and Linux versions are also available. If you are using some other computer system, you can download the source code and compile it on whatever platform you'd like. This is not something you can do with any commercial software package.

Gretl depends on some other (free) programs to perform some of its magic. If you install **gretl** on your Mac or Windows based machine using the appropriate executable file provided on **gretl**'s download page then everything you need to make **gretl** work should be installed as part of the package. If, on the other hand, you are going to build your own **gretl** using the source files, you may need to install some of the supporting packages yourself. I assume that if you are savvy enough to compile your own version of **gretl** then you probably know what to do. For most, just install the self-extracting executable, `gretl_install.exe`, available at the download site. **Gretl** comes with an Adobe pdf manual that will guide you through installation and introduce you to the interface. I suggest that you start with it, paying particular attention to the first 3 chapters, which discuss installation in more detail and some basics on how to use the interface.

Since this manual is based on the examples from *Principles of Econometrics, 4th edition* (*POE4*) by Hill et al. (2011), you should also download and install the accompanying data files that go with this book. The file is available at

`http://www.learneconometrics.com/gretl/POE4data.exe`.

This is a self-extracting windows file that will install the *POE4* data sets onto the `c:\Program Files (x86)\gretl\data` directory of your computer's harddrive.[1] If you have installed **gretl**

[1]My system is 64-bit. If your copy of Windows is 32-bit then your directory structure is likely to be different from

in any place other than `c:\Program Files (x86)\gretl` then you are given the opportunity to specify a new location in which to install the program during setup.

1.1.2 Gretl Basics

There are several different ways to work in **gretl**. Until you learn to use **gretl**'s rather simple and intuitive language syntax, the easiest way to use the program is through its built-in graphical user interface (GUI). The graphical interface should be familiar to most of you. Basically, you use your computer's mouse to open dialog boxes. Fill in the desired options and execute the commands by clicking on the **OK** button. **Gretl** is using your input from the dialogs, delivered by mouse clicks and a few keystrokes, to generate computer code that is executed in the background.

Of course, you can generate your own programs directly, either by using a command line version or by using the GUI via the **gretl console** or through **scripts**.

Gretl's command line version is a separate executable that gives you access to **gretl** commands directly from your computer's command prompt. This bypasses the GUI altogether.

To open the command line version of **gretl** in Windows, open a command window and type `gretlcli`. In Windows 7 choose `Start>Run` to open the dialog shown in figure 1.1. In the box, use

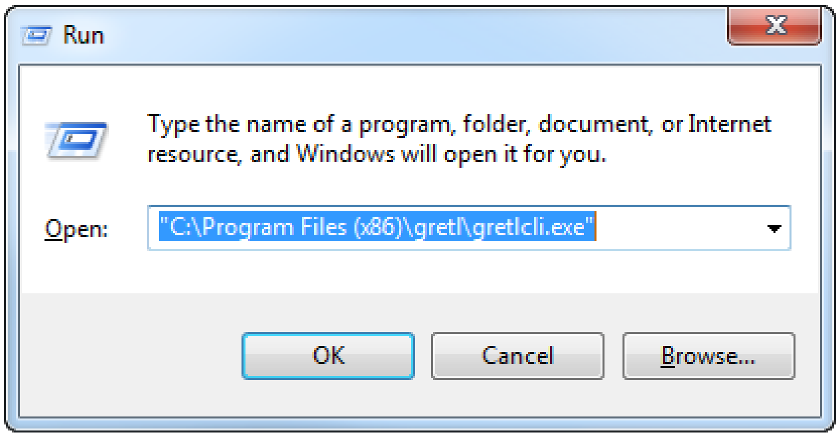
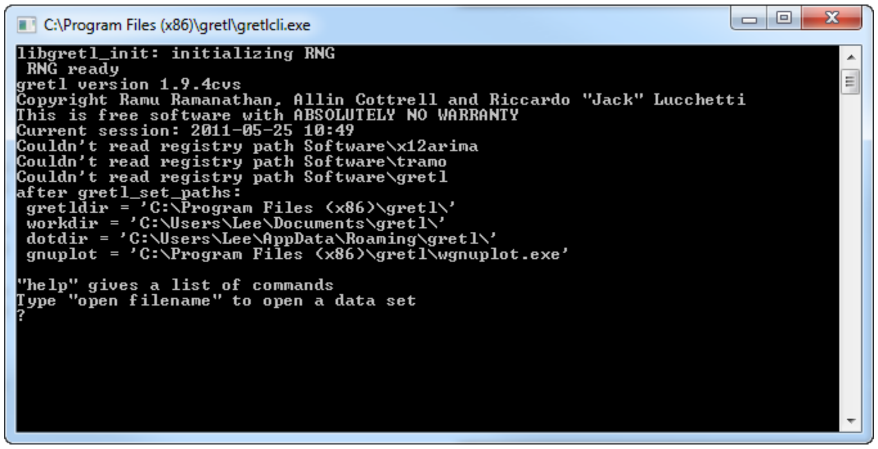

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
"C:\Program Files (x86)\gretl\gretlcli.exe"
OK
Cancel
Browse...

Figure 1.1: Opening the command line interface version of **gretl** using `Start>Run`

`Browse` button to locate the directory in which **gretl** is installed. On my machine it is installed on the `"C:\Program Files (x86)\gretl\gretlcli.exe"` drive. Click **OK** and the command line version shown in figure 1.2 opens. There are a couple of messages that certain entries could not be found in the Windows registry, which in this case means that these programs are not installed or registered on my particular machine. If you receive these, don't be alarmed. **Gretl** will still operate. The question mark (?) is the command prompt. To open one of the data sets that installs with **gretl**, type `open engel` at the prompt. The **gretl** data set *engel.gdt* opens and some

mins.



```
libgretl_init: initializing RNG
 RNG ready
gretl version 1.9.4cvs
Copyright Ramu Ramanathan, Allin Cottrell and Riccardo "Jack" Lucchetti
This is free software with ABSOLUTELY NO WARRANTY
Current session: 2011-05-25 10:49
Couldn't read registry path Software\x12arima
Couldn't read registry path Software\tramo
Couldn't read registry path Software\gretl
after gretl_set_paths:
 gretldir = 'C:\Program Files (x86)\gretl\'
 workdir = 'C:\Users\Lee\Documents\gretl\'
 dotdir = 'C:\Users\Lee\AppData\Roaming\gretl\'
 gnuplot = 'C:\Program Files (x86)\gretl\wgnuplot.exe'

"help" gives a list of commands
Type "open filename" to open a data set
?
```

Figure 1.2: The command line version of **gretl**

information about how much data and which variables it contains are printed to the screen. From here one can issue **gretl** commands or run scripts. To close the window, type `exit`.

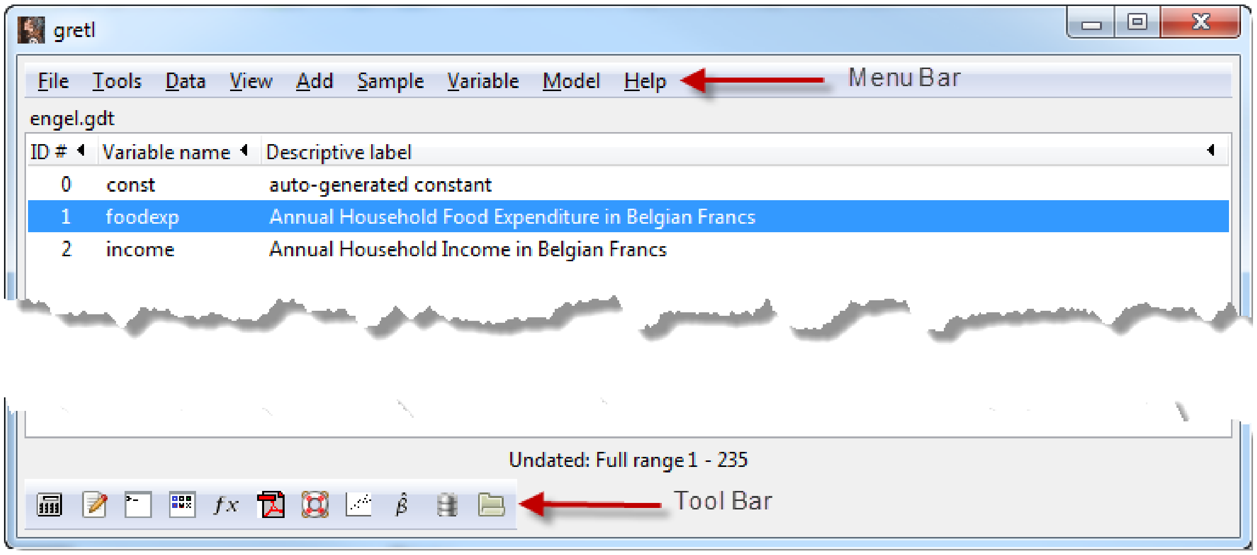
If you are in fact using the Microsoft Windows operating system, then you probably won't be using **gretl** from the command line very often anyway. This version of the program is probably the most useful for Linux users wishing to run **gretl** from a terminal window. If you are using a machine that is resource constrained, the command line interface is one way to free resources that would otherwise be used to operate the graphical interface. We won't be using the command line version in this manual.

A better way to execute single **gretl** commands is through the **gretl console**. In normal practice, the console is a lot easier to use than the `gretlcli.exe`. It offers some editing features and immediate access to other ways of using **gretl** that aren't available in the straight command line version of the program. The console and its use is discussed in section 1.3.1.

If you want to execute a series of commands, you do this using **scripts**. One of the great things about **gretl** is that it accumulates commands executed singly from the console into a **command log** that can be run in its entirety at another time. This topic can be found in section 1.3.2. So, if you have completed an analysis that involves many sequential steps, the steps can be saved to a script file which can later be opened and run in one step to get the result.

You can use the script environment to conduct Monte Carlo studies in econometrics. Monte Carlo studies use computer simulation (sometimes referred to as experiments) to study the properties of a particular technique. This is especially useful when the mathematical properties of your technique are particularly difficult to ascertain. In the exercises below, you will learn a little about doing these kinds of experiments in econometrics. Also, you can consult a separate paper of mine Adkins (2011*b*) which can be found at `http://www.learneconometrics.com/pdf/MCgretl/index.htm`.

In Figure 1.3 you will find the main window in **gretl**.


gretl
File Tools Data View Add Sample Variable Model Help
Menu Bar
engel.gdt
Undated: Full range 1 - 235
Tool Bar

ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	foodexp	Annual Household Food Expenditure in Belgian Francs
2	income	Annual Household Income in Belgian Francs

Figure 1.3: The main window for **gretl**'s GUI

Across the top of the window you find the **menu bar**. From here you import and manipulate data, analyze data, and manage output. At the bottom of the window is the **gretl** toolbar. This contains a number of useful utilities that can be launched from within **gretl**. Among other things, you can get to the **gretl** web site from here, open the pdf version of the manual, or open the MS Windows calculator (very handy!). More will be said about these functions later.

1.1.3 Common Conventions

In the beginning, I will illustrate the examples using a number of figures (an excessive number to be sure). These figures are screen captures of **gretl**'s windows as they appear when summoned from the pull-down menus. As you become familiar with **gretl** the frequency of these figures will diminish and I will direct you to the proper commands that can be executed in the console or as a script using words only. More complex series of commands may require you to use the **gretl** script facilities which basically allow you to write simple programs in their entirety, store them in a file, and then execute all of the commands in a single batch. The convention used will be to refer to menu items as `A>B>C` which indicates that you are to click on option A on the menu bar, then select B from the pull-down menu and further select option C from B's pull-down menu. All of this is fairly standard practice, but if you don't know what this means, ask your instructor now.

There are a few tricks used in this manual to make scripts work on various platforms without much modification. **Gretl** contains special macros for the location of commonly used files. There is a working directory that **gretl** reads and writes to. This location can be defined by the user using the file menu. To refer to this location generically, use the `@workdir` macro. The **gretl** installation director is referenced by `@gretldir`, and temporary storage can be accessed via `@dotdir`. If any of these directories have spaces in their names, then be sure to enclose the command in double quotes.

For example, on my Windows 7 system, **gretl** is installed in the `"C:\Program\;Files(x86)\gretl"` directory. The data sets for *POE4* are in `"@gretldir\data\poe\"`. To refer to this location I can simply use `"@gretldir\data\poe"`.

1.2 Importing Data

Obtaining data in econometrics and getting it into a format that can be used by your software can be challenging. There are dozens of different pieces of software and many use proprietary data formats that make transferring data between applications difficult. You'll notice that the authors of your book have provided data in several formats for your convenience. In this chapter, we will explore some of the data handling features of **gretl** and show you (1) how to access the data sets that accompany your textbook (2) how to bring one of those data sets into **gretl** (3) how to list the variables in the data set and (4) how to modify and save your data. **Gretl** offers great functionality in this regard. Through **gretl** you have access to a very large number of high quality data sets from other textbooks as well as from sources in industry and government. Furthermore, once opened in **gretl** these data sets can be exported to a number of other software formats.

First, we will load the food expenditure data used in chapter 2 of *POE4*. The data set contains two variables named x and y. The variable y is weekly expenditures on food in a household and x is weekly income measured in \$100 increments.

Open the main **gretl** window and click on `File>Open data>Sample file` as shown in Figure 1.4.

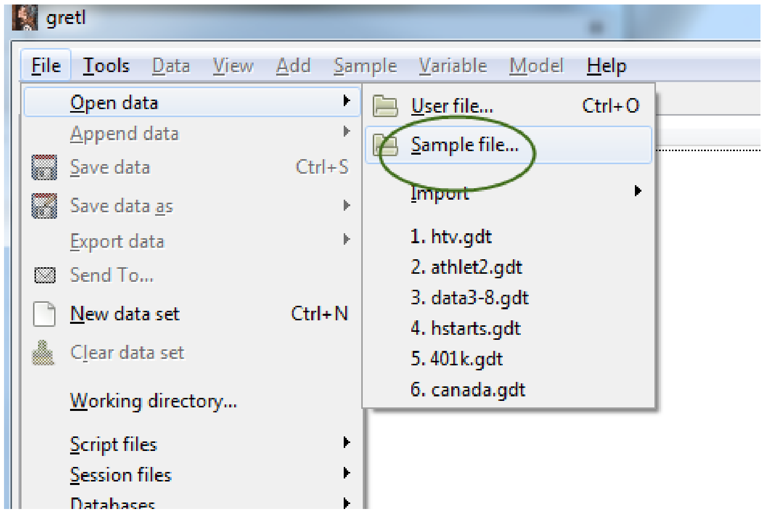
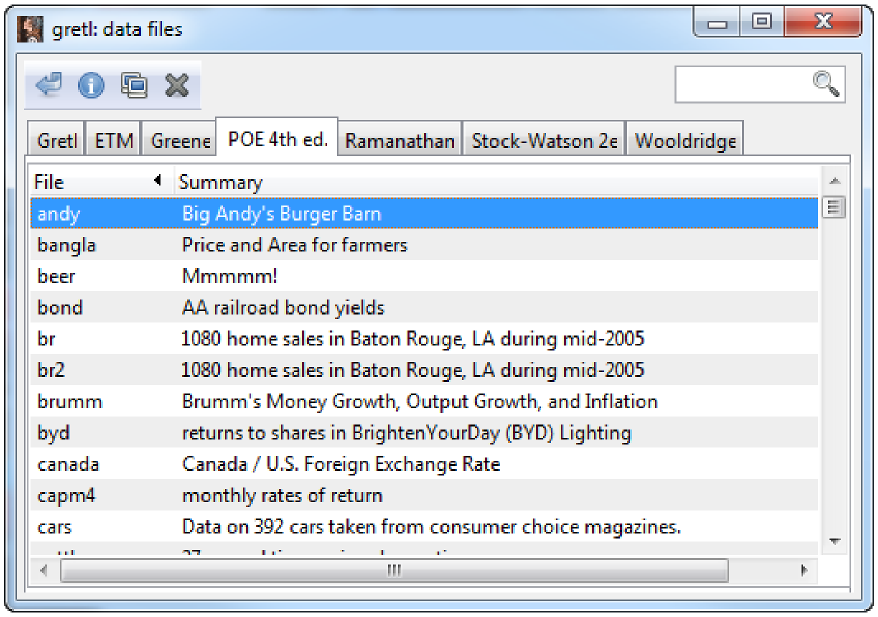


Figure 1.4: Opening sample data files from **gretl**'s main window

Alternately, you could click on the open dataset button on the toolbar. The button looks like a folder and is on the far right-hand side of the toolbar. This will open another window (Figure 1.5) that contains tabs for each of the data compilations that you have installed in the `gretl/data` directory of your program. If you installed the data sets that accompany this book using the self extracting windows program then a tab will appear like the one shown in Figure 1.5.



File	Summary
andy	Big Andy's Burger Barn
bangla	Price and Area for farmers
beer	Mmmmm!
bond	AA railroad bond yields
br	1080 home sales in Baton Rouge, LA during mid-2005
br2	1080 home sales in Baton Rouge, LA during mid-2005
brumm	Brumm's Money Growth, Output Growth, and Inflation
byd	returns to shares in BrightenYourDay (BYD) Lighting
canada	Canada / U.S. Foreign Exchange Rate
capm4	monthly rates of return
cars	Data on 392 cars taken from consumer choice magazines.

Figure 1.5: This is **gretl**'s data files window. Notice that in addition to POE, data sets from Ramanathan (2002), Greene (2003), Stock and Watson (2006), and others are installed on my system.

Click on the **POE 4th ed.** tab and scroll down to find the data set called 'food', highlight it using the cursor, and open it using the 'open' button at the top of the window. This will bring the variables of the food expenditure data set into **gretl**. At this point, select `Data` on the menu bar and then `Display values` as shown in Figure 1.6.

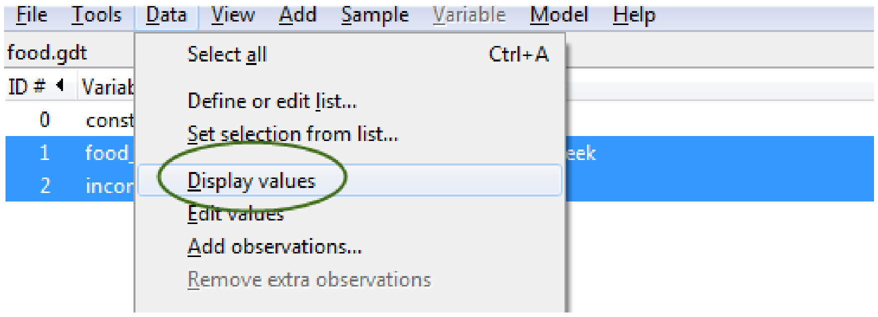


Figure 1.6: Use the cursor to highlight all of the variables. Then click `Data>Display values` to list the data set.

From the this pull-down menu a lot can be accomplished. You can edit, add observations, and impose a **structure** of your dataset. The structure of your dataset is important. You can choose between time-series, cross sections, or panel data structures. The options **Gretl** gives you depend on this structure. For instance, if your data are structured as a time-series, **gretl** will allow you to take lags and differences of the variables. Certain procedures that can be used for time-series analysis will only be available to you if your dataset has been structured structured for it. If a **gretl** command is not available from the defined dataset structure, then it will be greyed out in the pull-down menus.

Notice in Figure 1.4 that **gretl** gives you the opportunity to **import** data. Expanding this (`File>Open data>Import`) gives you access to several other formats, including Stata, Excel, Eviews, SPSS, and SAS (if installed). For instance, simply dragging a Stata dataset onto the main **gretl** window will bring the data into **gretl**.

Also, from the `File` pull-down menu you can export a data set to another format. The export feature is particularly useful for getting data into **R**.

If you click on `File>Databases>On database server` (Figure 1.4) you will be taken to a web site (provided your computer is connected to the internet) that contains a number of high quality data sets. You can pull any of these data sets into **gretl** in the same manner as that described above for the *POE4* data sets. If you are required to write a term paper in one of your classes, these data sets may provide you with all the data that you need. The database server is discussed in more detail below.

1.3 Using the gretl Language

The **gretl** GUI is certainly easy to use. However, you can get results even faster by using **gretl**'s language. The language can be used from the **console** or by collecting several lines of programming code into a file and executing them all at once in a **script**. **Gretl** now has a name for its scripting language, **hansl**. **Hansl** is a recursive acronym for hansl's a neat scripting language (or handy scripting language), and it is certainly that. There are many things you can do using this powerful tool. **Hansl**'s syntax is particularly easy to use, in my opinion, and I strongly recommend that you learn to use it.

An important fact to keep in mind when using **gretl** is that its language is **case sensitive**. This means that lower case and capital letters have different meanings in **gretl**. The practical implication of this is that you need to be very careful when using the language. Since **gretl** considers x to be different from X, it is easy to make programming errors. If **gretl** gives you a programming error statement that you can't quite decipher, make sure that the variable or command you are using is in the proper case.

1.3.1 Console

Gretl's console provides you a way to execute programs interactively. A console window opens and from the prompt (?) you can execute **gretl** commands one line at a time. You can open the

gretl console from the `Tools` pull-down menu or by a left mouse click on the "**Gretl** console" button on the toolbar. This button is the third one on the left side of the toolbar in Figure 1.3. From the console you execute commands, one by one by typing **gretl** code after the command prompt. Each command that you type in is held in memory so that you can accumulate what amounts to a "command history." To reuse a command, simply use the up arrow key to scroll through the commands you've typed in until you get to the one you want. You can edit the command to fix any syntax errors or to make any changes you desire before hitting the enter key to execute the statement.

From the command prompt, '?' you can type in commands from the **gretl** language. For instance, to estimate the food expenditure model in section 2.4 using least squares type

```
? ols y const x
```

The results will be output to the console window. You can use the window's scroll bar on the right hand side to scroll up if you need to.

Remember, (almost) anything that can be done with the pull-down menus can also be done through the console. Of course, using the console requires you to use the correct language syntax, which can be found in the **gretl** command reference. The command reference can be accessed from the toolbar by clicking the button that looks like a lifesaver. It's the fourth one from the right hand side of the toolbar.

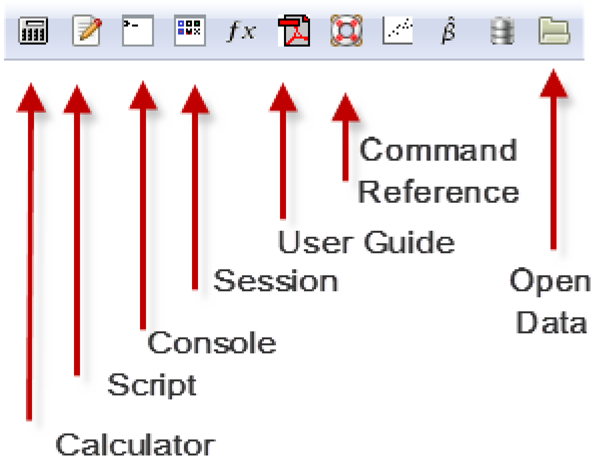
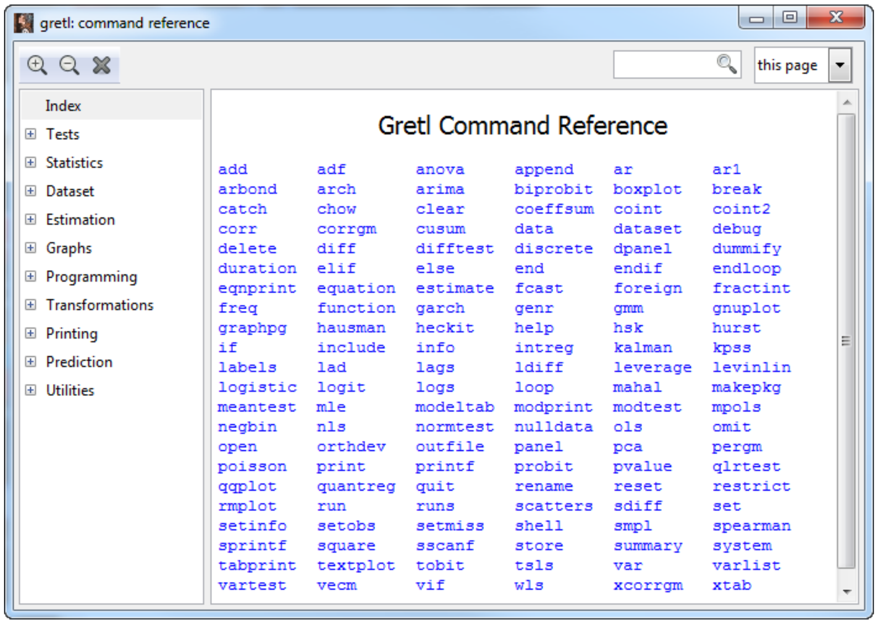


Figure 1.7: The toolbar appears at the bottom of the main menu.

The **Command Reference** is also accessible from the menu bar through **Help**. The option marked `plain text F1` actually brings up all of the commands in a hypertext format. Clicking on anything in blue will take you to the desired information for that command. Obviously, the keyboard shortcut F1 will also bring up the command reference (Figure 1.8). You'll also notice that .pdf versions of the Command and Function references can also be retrieved from the **Help** drop-down menu.



Gretl Command Reference

add	adf	anova	append	ar	ar1
arbond	arch	arima	biprobit	boxplot	break
catch	chow	clear	coeffsum	coint	coint2
corr	corrgm	cusum	data	dataset	debug
delete	diff	difftest	discrete	dpanel	dummify
duration	elif	else	end	endif	endloop
eqnprint	equation	estimate	fcast	foreign	fractint
freq	function	garch	genr	gmm	gnuplot
graphpg	hausman	heckit	help	hsk	hurst
if	include	info	intreg	kalman	kpss
labels	lad	lags	ldiff	leverage	levinlin
logistic	logit	logs	loop	mahal	makepkg
meantest	mle	modeltab	modprint	modtest	mpols
negbin	nls	normtest	nulldata	ols	omit
open	orthdev	outfile	panel	pca	pergm
poisson	print	printf	probit	pvalue	qlrtest
qqplot	quantreg	quit	rename	reset	restrict
rmplot	run	runs	scatters	sdiff	set
setinfo	setobs	setmiss	shell	smpl	spearman
sprintf	square	sscanf	store	summary	system
tabprint	textplot	tobit	tsls	var	varlist
vartest	vecm	vif	wls	xcorrgm	xtab

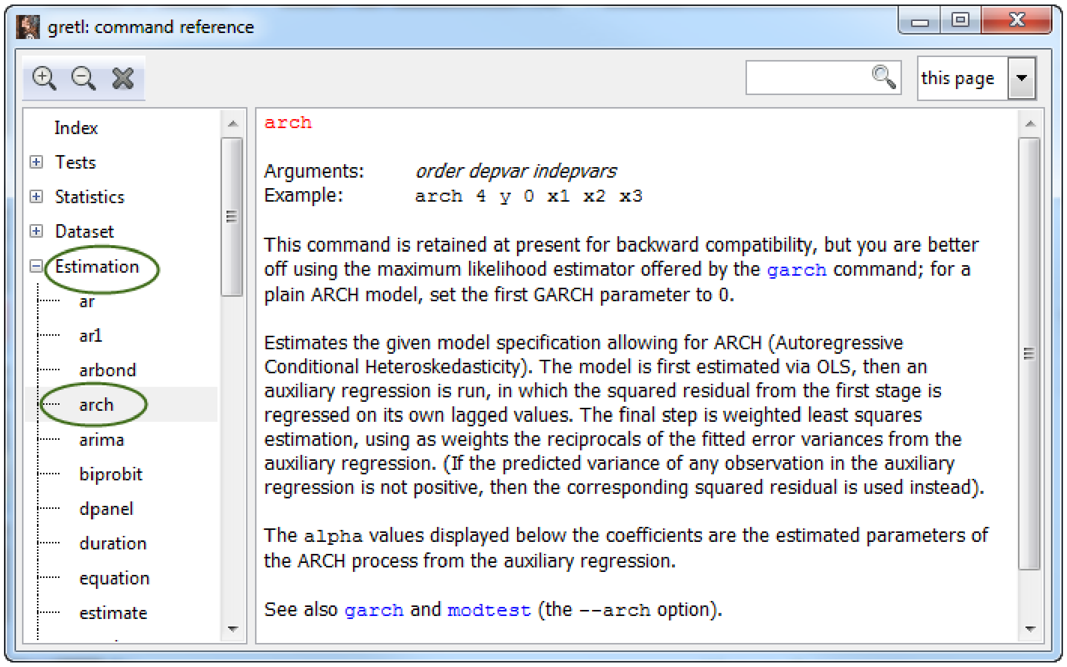
Figure 1.8: The command reference can be accessed in a number of ways: The 'life-saver' icon on the toolbar, `Help>Command reference` from the pull-down menu, or keyboard shortcut F1.

Notice that you can also search for commands by topic from the command syntax window. On the left side is a panel headed as `Index` (see Figure 1.9). Choose the desired category from the list and then select the command you want help with (e.g., `Estimation>arch`). The words indicated in blue text are links to related commands. For instance, clicking on garch will take you to the reference entry for garch modeling.

The **function reference** is a relatively new addition to **gretl** that will help you to locate the names **gretl** uses to temporarily store results (called **accessors**), to transform variables, and to write your own programs. To access the function reference, click `Help>Function reference` from the pull-down menu as shown in Figure 1.10.

1.3.2 Scripts

Gretl commands can be collected and put into a file that can be executed at once and saved to be used again. This is accomplished by opening a new **command script** from the file menu. The command `File>Script files>New script` from the pull-down menu opens the command script editor shown in Figure 1.11. Type the commands you want to execute in the box using one line for each command. Notice that in the first line of the script, `"I:\Program Files\gretl\data\poe\food.gdt"`, the complete file and path name are enclosed in double quotes,



arch

Arguments: *order depvar indepvars*
Example: `arch 4 y 0 x1 x2 x3`

This command is retained at present for backward compatibility, but you are better off using the maximum likelihood estimator offered by the `garch` command; for a plain ARCH model, set the first GARCH parameter to 0.

Estimates the given model specification allowing for ARCH (Autoregressive Conditional Heteroskedasticity). The model is first estimated via OLS, then an auxiliary regression is run, in which the squared residual from the first stage is regressed on its own lagged values. The final step is weighted least squares estimation, using as weights the reciprocals of the fitted error variances from the auxiliary regression. (If the predicted variance of any observation in the auxiliary regression is not positive, then the corresponding squared residual is used instead).

The `alpha` values displayed below the coefficients are the estimated parameters of the ARCH process from the auxiliary regression.

See also `garch` and `modtest` (the `--arch` option).

Figure 1.9: Finding help on the **`arch`** command using the Command Reference

`" "`. This is necessary because the `Program Files` directory in the pathname includes a space. If you have **gretl** installed in a location that does not include a space, then these can be omitted.

If you have a very long command that exceeds one line, use the backslash (\) as a **continuation command**. Then, to save the file, use the "save" button at the top of the box (first one from the left). If this is a new file, you'll be prompted to provide a name for it.

To run the program, click your mouse on the "gear" button. In the figure shown, the *food.gdt* **gretl** data file is opened. The `series` commands are used to take the logarithm of y and x, and the `ols` command discussed in section 2.4 is used to estimate a simple linear regression model that has $ln(y)$ as its dependent variable and $ln(x)$ as the independent variable. Note, the model also includes constant.

A new script file can also be opened from the toolbar by mouse clicking on the "new script" button or by using the keyboard command, Ctrl+N.[2]

One of the handy features of the command script window is how the help function operates. At the top of the window there is an icon that looks like a lifesaver . Click on the help button and the cursor changes into a question mark. Move the question mark over the command you want help with and click. Voila! You either get an error message or you are taken to the topic from the command reference. Slick!

[2]"Ctrl+N" means press the "Ctrl" key and, while holding it down, press "N".

1.3.3 Sessions

Gretl also has a "session" concept that allows you to save models, graphs, and data files into a common "iconic" space. The session window appears below in Figure 1.12. The session window is very handy. It contains icons that give you immediate access to information about the data set, that opens the edit data window, that display any scalars you have computed, summary statistics, correlations and any notes you may want to make.

Objects are represented as icons and these objects can be saved for later use. When you save your session, the objects you have added should be available again when you re-open the session. To add a model to your session, use the `File>Save to session as icon` option from the model's pull-down menu. To add a graph, right click on the graph and choose the option `save to session as icon`. If you want to save the results in your session, don't forget to do so; right click on the session window and choose `Save session` or from the main **gretl** window, select `File>Session files>Save session` as shown below in Figure 1.13.

Once a model or graph is added, its icon will appear in the **session icon view** window. Double-clicking on the icon displays the object, while right-clicking brings up a menu which lets you display or delete the object. You can browse the dataset, look at summary statistics and correlations, and save and revisit estimation results (Models) and graphs.

The model table is a way of combining several estimated models into a single table. This is very useful for model comparison. From the **gretl** manual ((Cottrell and Lucchetti, 2011, pp. 16-17)):

> In econometric research it is common to estimate several models with a common dependent variable the models contain different independent variables or are estimated using different estimators. 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 View menu in the main **gretl** window, or by clicking the `session icon view` icon on the **gretl** toolbar.
> 4. In the 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.

In section 6.4 you'll find an example that uses the model table and a Figure (6.13) that illustrates the result.

1.3.4 Generating New Variables

In this manual, we will be generating new variables, computing statistics based on **gretl** output, and performing matrix calculations using **gretl**'s scripting language. That means we will be generating series, scalars, matrices, lists, and even strings. How does **gretl** handle these?

Gretl is actually very forgiving in the generation of new results. The 'mother' command for doing this is `genr`. The `genr` command pretty much does it all. In the appropriate context, `series`, `scalar` and `matrix` are synonyms for this command.

So, to create a new scalar result, say create a constant c that is equal to 3, you could use `scalar c = 3` or `genr c = 3`. The `scalar` and `genr` commands let **gretl** know that you are calculating something and calling it `c`.

To create a new variable, you can use the `series` command or `genr`. Suppose you have a variable in your dataset called `food_exp`. You want to create a new variable as the natural logarithm of `food_exp`. This can be done using `series` or `genr` (e.g., `series l_food_exp = ln(food_exp)`). In the context of a `genr` or `series` formula, variables must be referenced by their names, not their ID numbers. The formula should be a well-formed combination of variable names, constants, operators and functions. Further details on some aspects of this command can be found in the **Gretl** Users Guide.

As you have seen, a `genr` command may yield either a series or a scalar result. For example, the formula `x2 = x * 2` naturally yields a series if the variable `x` is a series and a scalar if `x` is a scalar. The formulae `x = 0` and `mx = mean(x)` naturally return scalars. The `genr` command handles both cases seamlessly.

Under some circumstances you may want to have a scalar result expanded into a series or vector. You can do this by using `series` as an "alias" for the `genr` command. For example, `series x =`

`0` produces a series all of whose values are set to 0. You can also use `genr` as an alias for `scalar`. It is not possible to coerce a vector result into a scalar, but use of this keyword indicates that the result should be a scalar: if it is not, an error occurs.

In many cases, you can even omit the `genr`, `series`, `scalar`, or `matrix` statements and **gretl** will figure out what to compute based on what is on the right-hand side of your equation. This is dangerous though, because you may inadvertently be trying to compute objects with incompatible dimensions or of incompatible types.

In this book, I may at times use `genr` instead of the preferred `series` command to create new variables. I am told by members of the **gretl** team that it is better practice to call things what they are and so `series`, `scalar`, and `matrix` are better than the generic (but equally effective) `genr`. One of the amazing things about the **gretl** language is that omitting these commands altogether usually works anyway. Still, I think there are good reasons to get started on the right foot by adopting good programming practices.[3] There are at least three commands that demand the use of `genr`, rather than `series`. These involve creating a time index (`genr time`) and dummy variables (`genr unitdum` and `genr dummy`). These cases will be pointed out when we get to them.

One of the advantages of using descriptive prefixes to series, scalars, and matrices occurs when you write functions. **Gretl** functions are a very powerful way to extend **gretl**'s capabilities. They are finicky though. The inputs must be identified by type as does any output. Type mismatches are a common source of error. So, the more thought that goes into daily use will pay dividends later should you decide to start writing your own **gretl** functions.

1.4 GNUPLOT

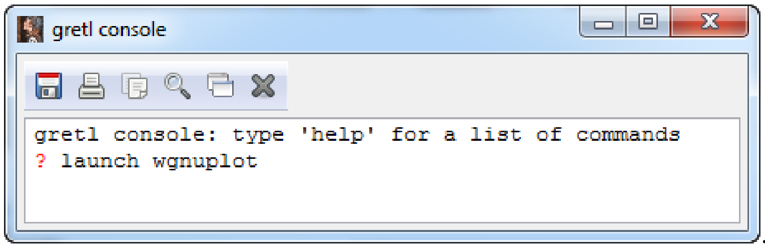
At the end of each chapter that follows you will find listings of the entire **gretl** script used to generate the results that are contained in it. When a graph is generated using **gnuplot**in a script or from the console, the output is written to a file that is placed in the working directory of **gretl**. If you are not sure where that is, click `File>Working directory` in the main **gretl** window to find or change this location. The location of the file will also be echoed to the screen so locating it should be fairly easy.

To view the graph and to edit it requires you to open the **gnuplot** program. In Windows, the easiest way to do this is to open the **gretl** console and type:

```
launch wgnuplot
```

This will look like

[3]Astute programmers will note that my own programming leaves much to be desired. Adopting better practices when learning to program would have made doing econometrics much easier.



```
gretl console: type 'help' for a list of commands
? launch wgnuplot
```

.

This opens **gnuplot** in a new window. Now, navigate to the **gnuplot** window and at the **gnuplot** command prompt type

```
load 'C:\Temp\gpttmp01.plt'
```

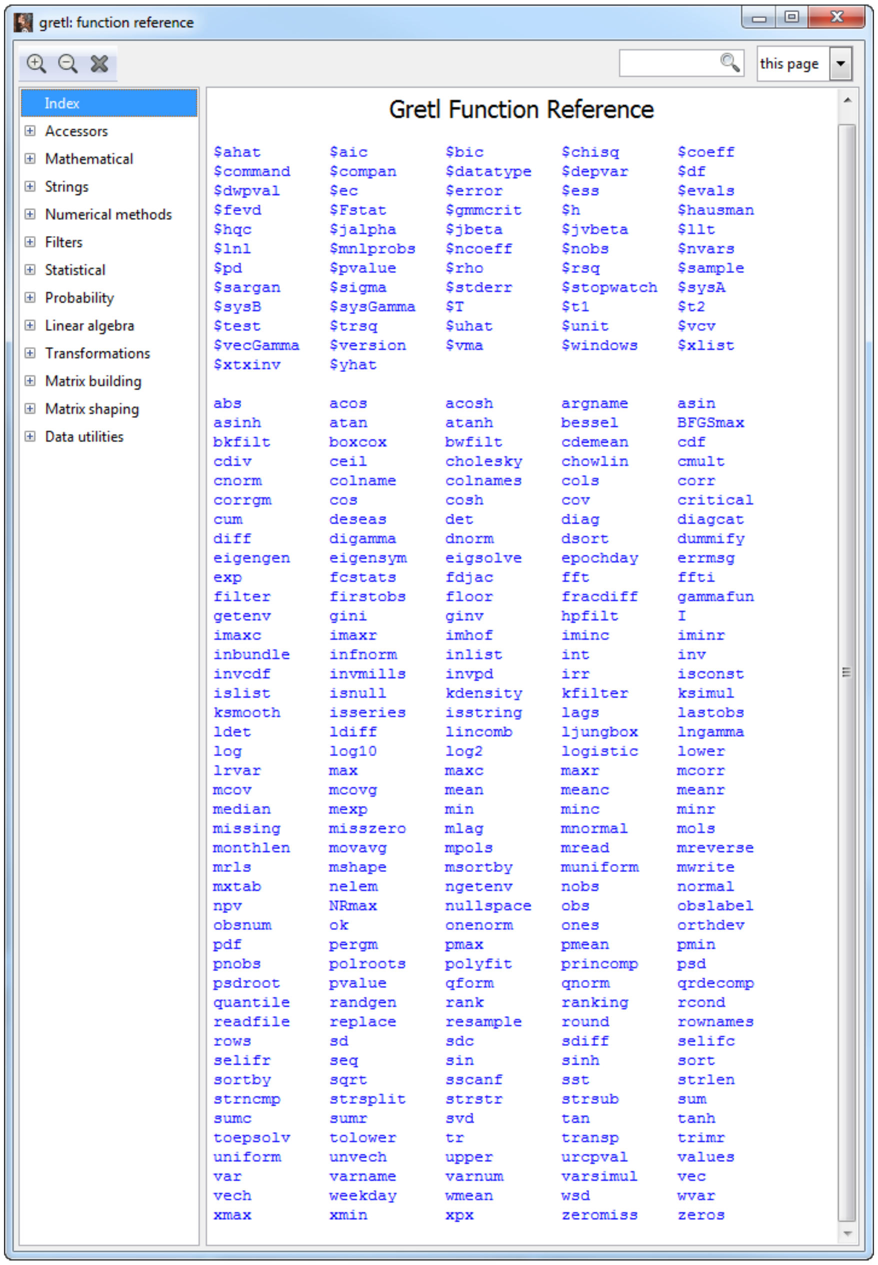
The path and filename inside the single quotes locates the file on your harddrive. **Gretl** places these plots into your working directory, which can be set using `File>Working directory` from the main **gretl** window. Figure 1.14 shows what this looks like.

Another way to do this is to open a command window (Figure 1.1) and type `"C:\Program Files (x86)\gretl\wgnuplot"` at the command prompt. The double quotes are necessary since the folder name has a space in it. This will open the **gnuplot** program shown in Figure 1.14, from which you can search for and open graphs that are written to the harddrive. This implementation is a bit clumsy and is not very well documented in the **gretl** Users Guide at this point, but as with most things **gretl** it is a work in progress. By the time you read this, the situation could be much improved.

Although scripts are given to generate graphs in this text, the best way to do it is by using the GUI or from the console. Graphs generated via GUI **or the console** open to the screen. Once the graph is generated and visible on screen, a right-click of the mouse allows you to edit the graph and to save it in a variety of useful formats. That is what I have done in a number of graphs that follow to make them easier to read from the .pdf. Using **gnuplot** manually is really only necessary if your graphs are being generated in a script as some of the ones in this text are.

You do not have to accept **gretl**'s default graph name. You can assign one yourself using the `--output=`*`filename`*, which sends your output to the specified *`filename`*.

Finally, there are a number of other types of plots you can do in **gretl**. These include boxplots, histograms, qqplots, and range/mean plots. The underlying engine that generates these is **gnuplot** , but **gretl** gives you easy access to their generation. You can also access **gnuplot** by script through `File>Script files>New script>gnuplot script` from the main menu.

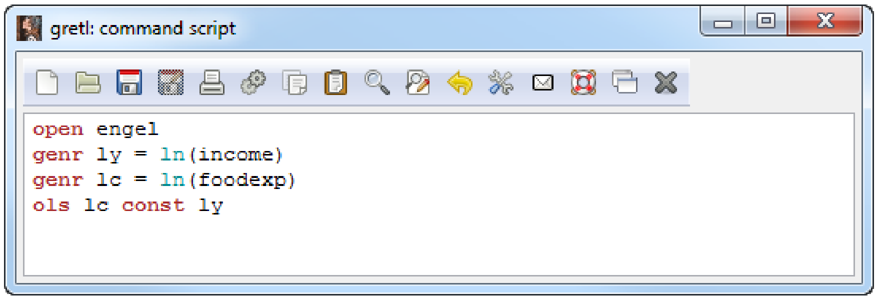


Gretl Function Reference

$ahat	$aic	$bic	$chisq	$coeff
$command	$compan	$datatype	$depvar	$df
$dwpval	$ec	$error	$ess	$evals
$fevd	$Fstat	$gmmcrit	$h	$hausman
$hqc	$jalpha	$jbeta	$jvbeta	$llt
$lnl	$mnlprobs	$ncoeff	$nobs	$nvars
$pd	$pvalue	$rho	$rsq	$sample
$sargan	$sigma	$stderr	$stopwatch	$sysA
$sysB	$sysGamma	$T	$t1	$t2
$test	$trsq	$uhat	$unit	$vcv
$vecGamma	$version	$vma	$windows	$xlist
$xtxinv	$yhat			

abs	acos	acosh	argname	asin
asinh	atan	atanh	bessel	BFGSmax
bkfilt	boxcox	bwfilt	cdemean	cdf
cdiv	ceil	cholesky	chowlin	cmult
cnorm	colname	colnames	cols	corr
corrgm	cos	cosh	cov	critical
cum	deseas	det	diag	diagcat
diff	digamma	dnorm	dsort	dummify
eigengen	eigensym	eigsolve	epochday	errmsg
exp	fcstats	fdjac	fft	ffti
filter	firstobs	floor	fracdiff	gammafun
getenv	gini	ginv	hpfilt	I
imaxc	imaxr	imhof	iminc	iminr
inbundle	infnorm	inlist	int	inv
invcdf	invmills	invpd	irr	isconst
islist	isnull	kdensity	kfilter	ksimul
ksmooth	isseries	isstring	lags	lastobs
ldet	ldiff	lincomb	ljungbox	lngamma
log	log10	log2	logistic	lower
lrvar	max	maxc	maxr	mcorr
mcov	mcovg	mean	meanc	meanr
median	mexp	min	minc	minr
missing	misszero	mlag	mnormal	mols
monthlen	movavg	mpols	mread	mreverse
mrls	mshape	msortby	muniform	mwrite
mxtab	nelem	ngetenv	nobs	normal
npv	NRmax	nullspace	obs	obslabel
obsnum	ok	onenorm	ones	orthdev
pdf	pergm	pmax	pmean	pmin
pnobs	polroots	polyfit	princomp	psd
psdroot	pvalue	qform	qnorm	qrdecomp
quantile	randgen	rank	ranking	rcond
readfile	replace	resample	round	rownames
rows	sd	sdc	sdiff	selifc
selifr	seq	sin	sinh	sort
sortby	sqrt	sscanf	sst	strlen
strncmp	strsplit	strstr	strsub	sum
sumc	sumr	svd	tan	tanh
toepsolv	tolower	tr	transp	trimr
uniform	unvech	upper	urcpval	values
var	varname	varnum	varsimul	vec
vech	weekday	wmean	wsd	wvar
xmax	xmin	xpx	zeromiss	zeros

Figure 1.10: The function reference can be accessed by `Help>Function reference` from the pull-down menu.



```
open engel
genr ly = ln(income)
genr lc = ln(foodexp)
ols lc const ly
```

Figure 1.11: The Command Script editor is used to collect a series of commands into what **gretl** calls a **script**. The script can be executed as a block, saved, and rerun at a later time.

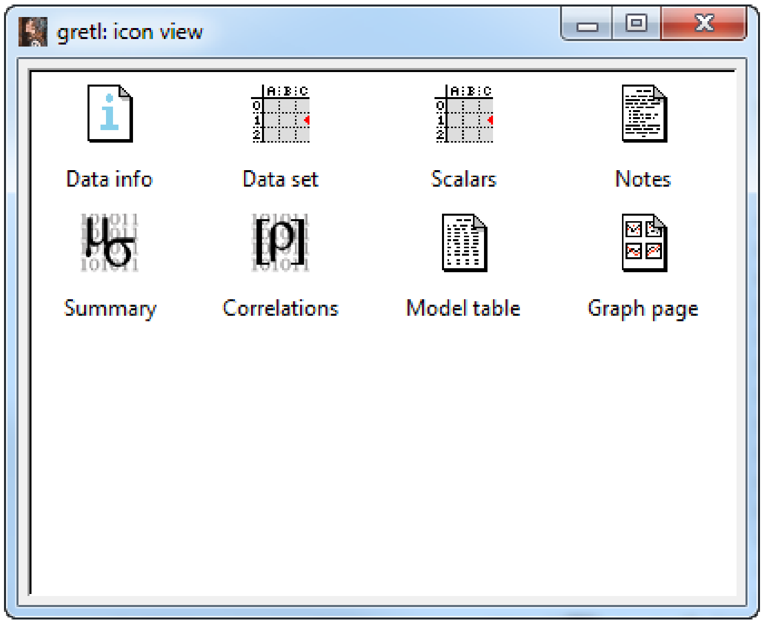


Figure 1.12: The session window

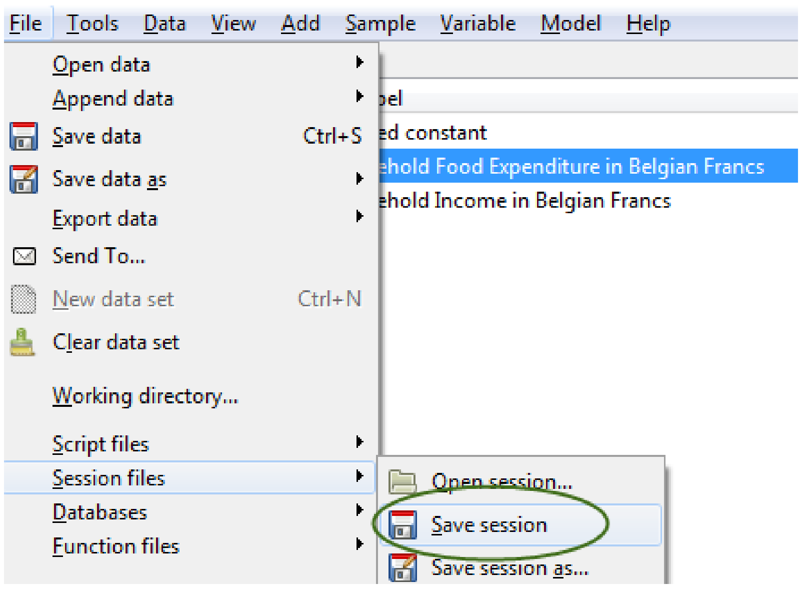
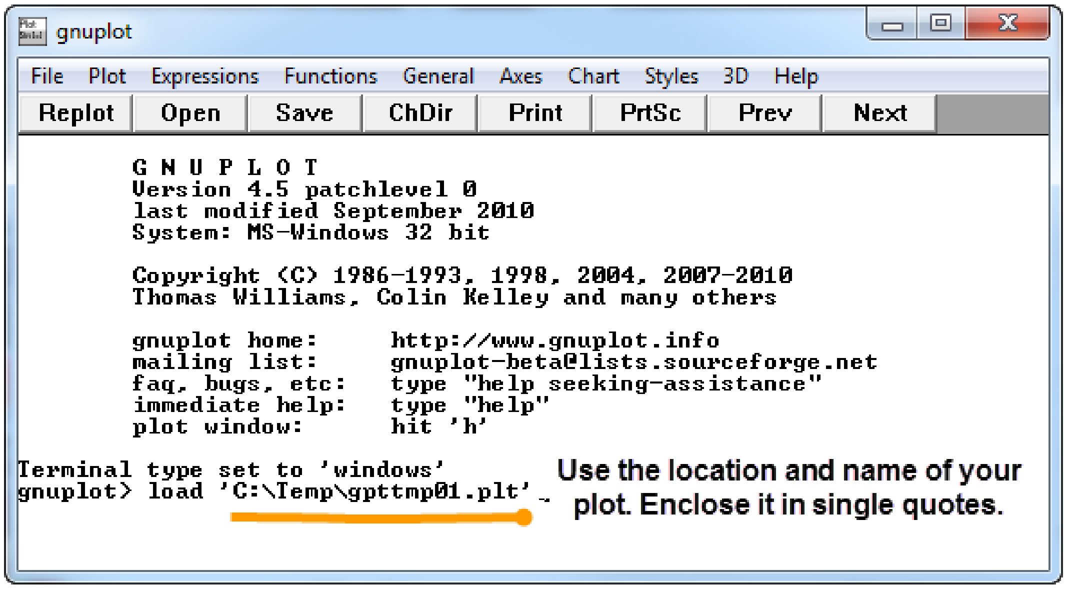


Figure 1.13: Saving a session


gnuplot
File Plot Expressions Functions General Axes Chart Styles 3D Help
Replot Open Save ChDir Print PrtSc Prev Next
Use the location and name of your plot. Enclose it in single quotes.

```
        G N U P L O T
        Version 4.5 patchlevel 0
        last modified September 2010
        System: MS-Windows 32 bit

        Copyright (C) 1986-1993, 1998, 2004, 2007-2010
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        mailing list:     gnuplot-beta@lists.sourceforge.net
        faq, bugs, etc:   type "help seeking-assistance"
        immediate help:   type "help"
        plot window:      hit 'h'

Terminal type set to 'windows'
gnuplot> load 'C:\Temp\gpttmp01.plt'
```

Figure 1.14: The GNUPLOT program window. This is opened from within **gretl** by typing `launch wgnuplot` from the console. Type `load 'filename'` to load `'filename'`, which should include the correct path. In this case the file to load is `'C:\Temp\gpttmp01.plt'`.

Chapter 2

Simple Linear Regression

In this chapter you are introduced to the simple linear regression model, which is estimated using the principle of least squares.

2.1 Simple Linear Regression Model

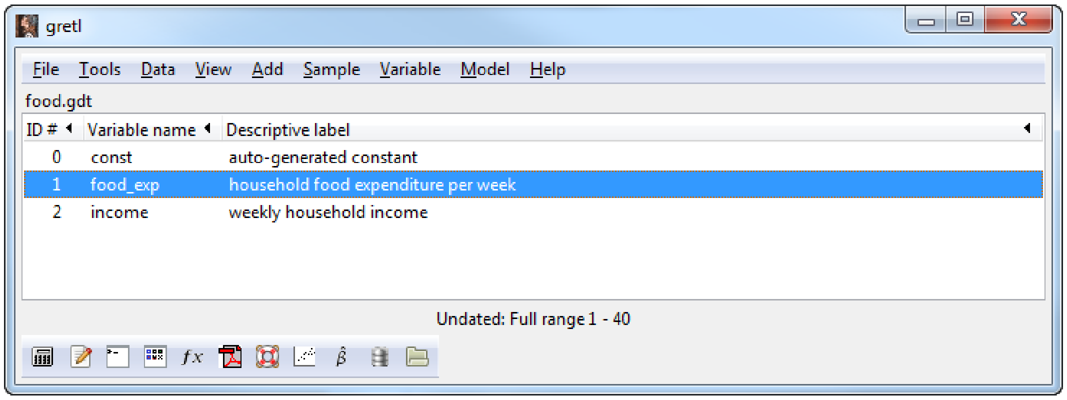
The simple linear regression model is

$$food\_exp_t = \beta_1 + \beta_2 income_t + e_t \quad t = 1, 2, \ldots, T \tag{2.1}$$

where $food\_exp_t$ is your dependent variable, $income_t$ is the independent variable, e_t is random error, and β_1 and β_2 are the parameters you want to estimate. The errors of the model, e_t, have an average value of zero for each value of $income_t$; each has the same variance, σ^2, and are uncorrelated with one another. The independent variable, $income_t$, has to take on at least two different values in your dataset. If not, you won't be able to estimate a slope! The error assumptions can be summarized as $e_t | income_t \; iid \; N(0, \sigma^2)$. The expression *iid* stands for **independently and identically distributed** and means that the errors are statistically independent from one another (and therefore uncorrelated) and that each has the same probability distribution. Taking a random sample from a single population accomplishes this.

2.2 Retrieve the Data

The first step is to load the food expenditure and income data into **gretl**. The data file is included in your **gretl** sample files–provided that you have installed the *Principles of Econometrics* data supplement that is available from our website. See section 1.1.1 for details.



food.gdt

ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	food_exp	household food expenditure per week
2	income	weekly household income

Undated: Full range 1 - 40

Figure 2.1: The main **gretl** window. The food expenditure data is loaded from *food.gdt* using `File>Open data>sample file` and choosing the food dataset from the sample files that accompany *POE4*.

Load the data from the data file *food.gdt*. Recall, this is accomplished by the commands `File>Open data>Sample file` from the menu bar.[1] Choose *food* from the list. When you bring the file containing the data into **gretl** your window will look like the one in Figure 2.1. Notice that in the `Descriptive label` column contains some information about the variables in the program's memory. For some of the datasets included with this book, it may be blank. These descriptions, when they exist, are used by the graphing program to label your output and to help you keep track of variables that are available for use. Before you graph your output or generate results for a report or paper you may want to label your variables to make the output easier to understand. This can be accomplished by editing the attributes of the variables.

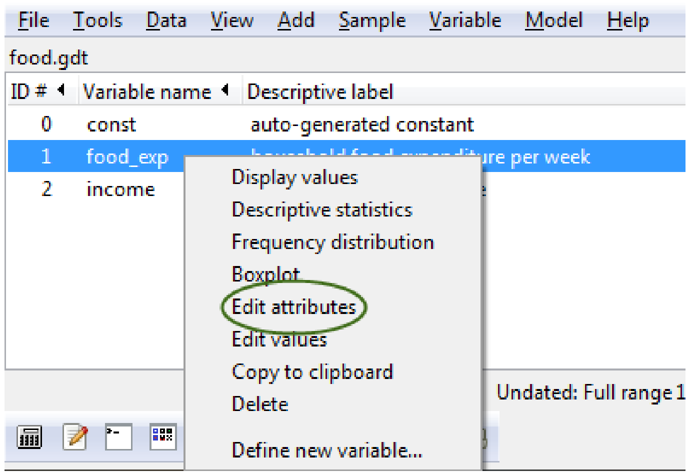
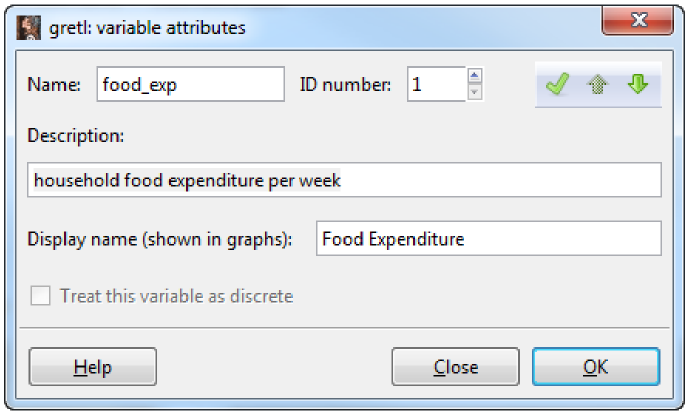


Figure 2.2: Highlight the desired variable and right-click to bring up the pull-down menu shown here. You can also use F2 or keyboard shortcut 'CTRL+e' to bring up the dialog.

[1]Alternately, you could click on the open data button on the toolbar. It's the one that looks like a folder on the far right-hand side.

To do this, first highlight the variable whose attributes you want to edit, right-click and the menu shown in (see Figure 2.2) appears. Select `Edit attributes` to open a dialog box (Figure 2.3) where you can change the variable's name, assign variable descriptions and display names. Describe and label the variable `food_exp` as 'Food Expenditure' and `income` as 'Weekly Income ($100).' The dialog can also be opened using F2 from the main **gretl** window or using the keyboard



Name: food_exp ID number: 1

Description:

household food expenditure per week

Display name (shown in graphs): Food Expenditure

☐ Treat this variable as discrete

Figure 2.3: Variable edit dialog box

shortcut, 'E.' Finally, the `setinfo` command can be used to set the description and the label used

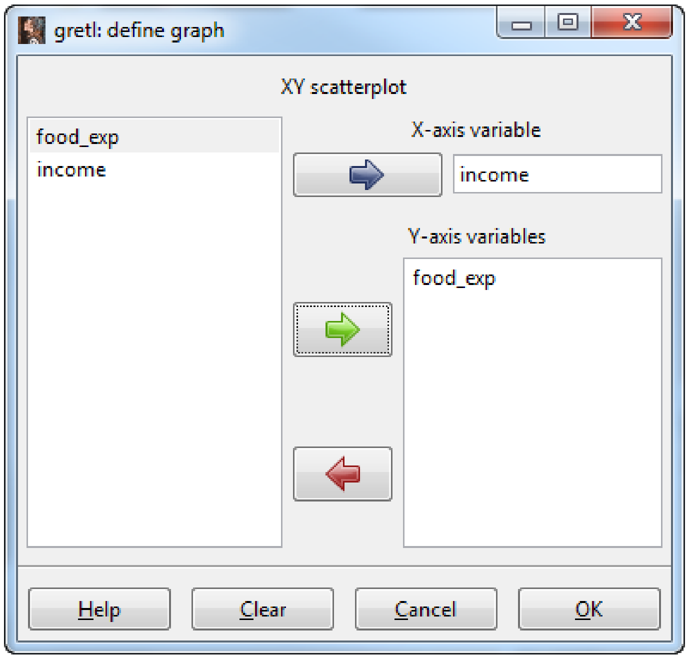


Figure 2.4: Use the dialog to plot of the food expenditure against Weekly Income

in graphs.

In the following example a script is generated that opens the *food.gdt* dataset, and adds variable descriptions, and assigns a label to be used in subsequent graphs.

```
open "@gretldir\data\poe\food.gdt"
setinfo food_exp -d "household food expenditure per week" \
    -n "Food Expenditure"
setinfo income -d "weekly household income" -n "Weekly Income"
labels
```

The `-d` flag is given followed by a string in double quotes. It is used to set the descriptive label. The `-n` flag is used similarly to set the variable's name in graphs. Notice that in the first and last uses of `setinfo` in the example that I have issued the continuation command (\) since these commands are too long to fit on a single line. If you issue the `labels` command, **gretl** will respond by printing the descriptions to the screen.

2.3 Graph the Data

To generate a graph of the food expenditure data that resembles the one in Figure 2.6 of *POE*, you can use the button on the **gretl** toolbar (third button from the right). Clicking this button brings up a dialog to plot the two variables against one another. Figure 2.4 shows this dialog where x is placed on the x-axis and y on the y-axis. The result appears in Figure 2.5. Notice that the labels applied above now appear on the axes of the graph.

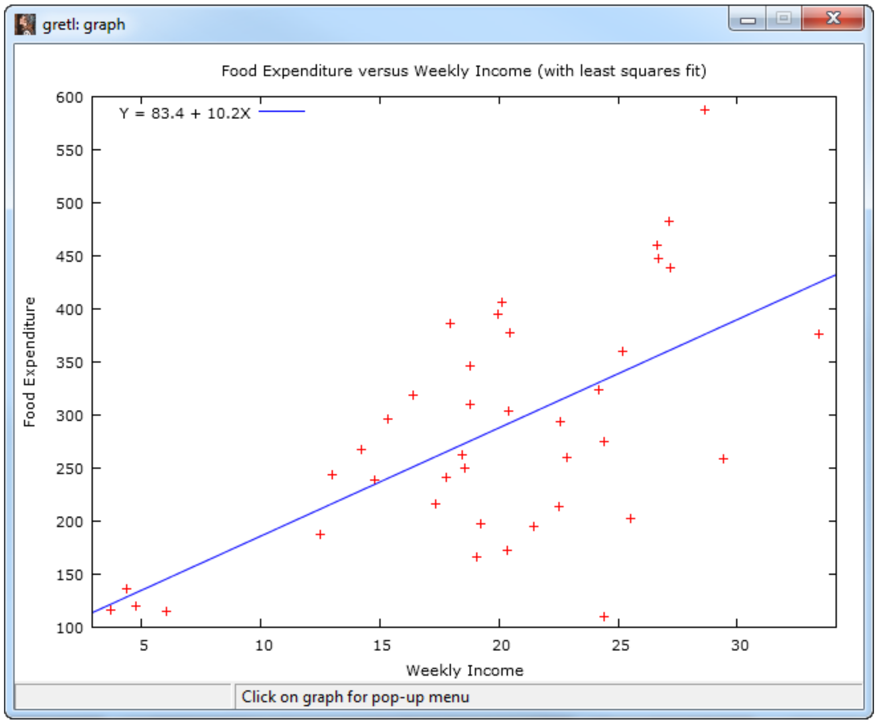
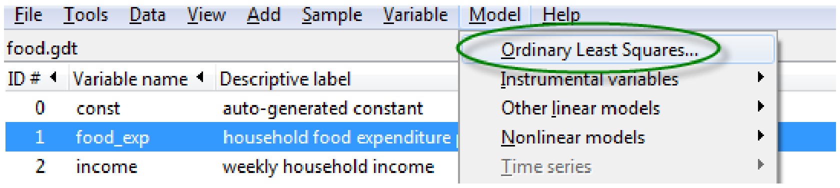


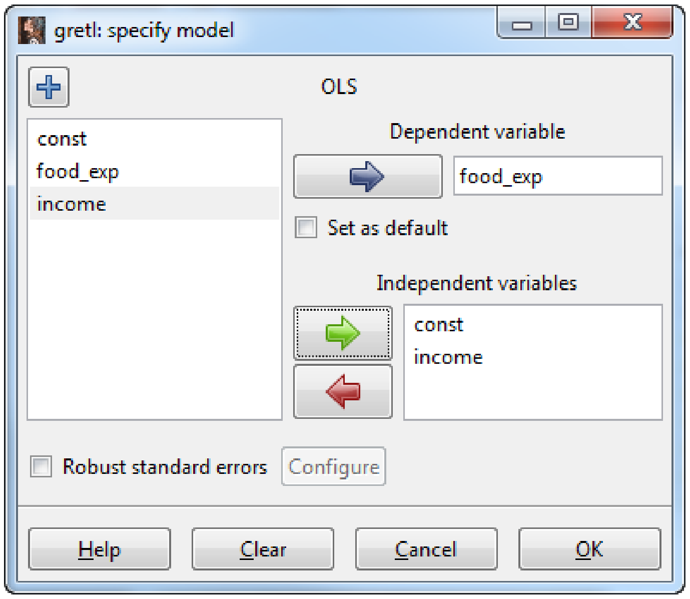
Figure 2.5: XY plot of the food expenditure data

Figure 2.5 plots food expenditures on the y axis and Weekly Income on the x. **Gretl**, by default, also plots the fitted regression line. The benefits of assigning labels to the variables becomes more obvious. Both X- and Y-axes are informatively labeled and the graph title is changed as well. More on this later.



ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	food_exp	household food expenditure
2	income	weekly household income

Figure 2.6: From the menu bar, select `Model>Ordinary Least Squares` to open the least squares dialog box



Dependent variable: food_exp

Independent variables: const, income

Figure 2.7: The Specify Model dialog box opens when you select `Model>Ordinary least squares`

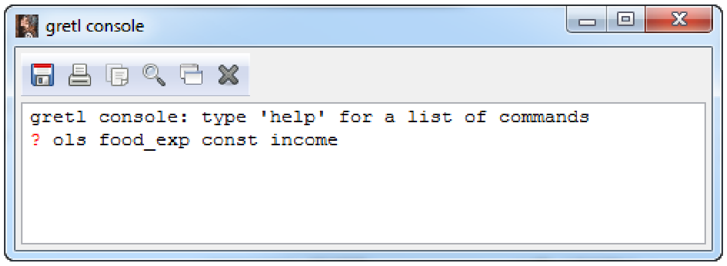
2.4 Estimate the Food Expenditure Relationship

Now you are ready to use **gretl** to estimate the parameters of the food expenditure equation.

$$food\_exp_t = \beta_1 + \beta_2 income_t + e_t \quad t = 1, 2, \ldots, T \tag{2.2}$$

From the menu bar, select `Model>Ordinary Least Squares` from the pull-down menu (see Figure 2.6) to open the dialog box shown in Figure 2.7. From this dialog you'll need to tell **gretl** which variable to use as the dependent variable and which is the independent variable. Notice that by default, **gretl** assumes that you want to estimate an intercept (β_1) and includes a constant as an independent variable by placing the variable `const` in the list by default. To include x as an independent variable, highlight it with the cursor and click the '`Add->`' button.

The **gretl** console (see section 1.3.1) provides an easy way to run a regression. The **gretl** console is opened by clicking the console button on the toolbar, ▣. The resulting console window is shown in Figure 2.8.



```
gretl console: type 'help' for a list of commands
? ols food_exp const income
```

Figure 2.8: The Gretl console window. From this window you can type in **gretl** commands directly and perform analyses very quickly–if you know the proper **gretl** commands.

At the question mark in the console simply type

```
ols y const x
```

to estimate your regression function. The syntax is very simple, `ols` tells **gretl** that you want to estimate a linear function using ordinary least squares. The first variable listed will be your dependent variable and any that follow, the independent variables. These names must match the ones used in your data set. Since ours in the food expenditure example are named, `y` and `x`, respectively, these are the names used here. Don't forget to estimate an intercept by adding a constant (`const`) to the list of regressors. Also, don't forget that **gretl** is case sensitive so that `x` and `X` are different entities.

This yields window shown in Figure 2.9 below. The results are summarized in Table 2.1.

An equivalent way to present results, especially in very small models like the simple linear regression, is to use equation form. In this format, the **gretl** results are:

$$\widehat{\text{food\_exp}} = \underset{(43.410)}{83.4160} + \underset{(2.0933)}{10.2096}\,\text{income}$$

$$T = 40 \quad \bar{R}^2 = 0.3688 \quad F(1,38) = 23.789 \quad \hat{\sigma} = 89.517$$

$$\text{(standard errors in parentheses)}$$

Finally, notice in the main **gretl** window (Figure 1.3) that the first column has a heading called

Table 2.1: OLS estimates using the 40 observations 1–40.

OLS, using observations 1–40
Dependent variable: food_exp

	Coefficient	Std. Error	t-ratio	p-value
const	83.4160	43.4102	1.9216	0.0622
income	10.2096	2.09326	4.8774	0.0000

Mean dependent var	283.5735	S.D. dependent var	112.6752
Sum squared resid	304505.2	S.E. of regression	89.51700
R^2	0.385002	Adjusted R^2	0.368818
$F(1,38)$	23.78884	P-value(F)	0.000019
Log-likelihood	−235.5088	Akaike criterion	475.0176
Schwarz criterion	478.3954	Hannan–Quinn	476.2389

ID #. An ID # is assigned to each variable in memory and you can use the ID # instead of its variable name in your programs. For instance, the following two lines yield identical results:

```
ols food_exp const income
ols 1 0 2
```

One (1) is the ID number for `food_exp` and two (2) is the ID number of `income`. The constant has ID zero (0). If you tend to use long and descriptive variable names (recommended, by the way), using the ID number can save you a lot of typing (and some mistakes).

2.4.1 Elasticity

Elasticity is an important concept in economics. It measures how responsiveness one variable is to changes in another. Mathematically, the concept of elasticity is fairly simple:

$$\varepsilon = \frac{\text{percentage change in } y}{\text{percentage change in } x} = \frac{\Delta y/y}{\Delta x/x} \tag{2.3}$$

In terms of the regression function, we are interested in the elasticity of average food expenditures with respect to changes in income:

$$\varepsilon = \frac{\Delta E(y)/E(y)}{\Delta x/x} = \beta_2 \frac{x}{E(y)}. \tag{2.4}$$

$E(y)$ and x are usually replaced by their sample means and β_2 by its estimate. The mean of `food_exp` and `income` can be obtained by using the cursor to highlight both variables, use the

`View>Summary statistics` from the menu bar as shown in Figure 2.10, and the computation can be done by hand. However, you can make this even easier by using the **gretl** language to do all of the computations–no calculator needed! Simply open up a new script and type in:

```
ols food_exp const income --quiet
scalar elast=$coeff(income)*mean(income)/mean(food_exp)
```

This yields the output shown in figure 2.11.

Following a least squares regression, **Gretl** stores the least squares estimates of the constant and the slope in variables called `$coeff(const)` and `$coeff(income)`, respectively. In addition, it uses `mean(income)` and `mean(food_exp)`to compute the mean of the variables `income` and `food_exp`. The `--quiet` option is convenient when you don't want or need the output from the regression printed to the screen. The result from this computation appears below in Figure 2.12.

2.4.2 Prediction

Similarly, **gretl** can be used to produce predictions. The predicted food expenditure of an average household having weekly income of \$2000 is:

$$\widehat{food\_exp}_t = 83.42 + 10.21 income_t = 83.42 + 10.21(20) = 287.61 \quad (2.5)$$

Remember, *income* is measured in \$100, so 20 in the above expression represents 20*\$100=\$2,000. The **gretl** script is:

```
scalar yhat = $coeff(const) + $coeff(income)*20
```

which yields the desired result.

2.4.3 Estimating Variance

In section 2.7 of *POE4*, you are given expressions for the variances of the least squares estimators of the intercept and slope as well as their covariance. These estimators require that you estimate the overall variance of the model's errors, σ^2. **Gretl** does not explicitly report the estimator, $\hat{\sigma}^2$, but rather, its square root, $\hat{\sigma}$. **Gretl** calls this "S.E. of regression" which you can see from the output is 89.517. Thus, $89.517^2 = 8013.29$. **Gretl** also reports the sum of squared residuals, equal to 304505.2, from which you can calculate the estimate. Dividing the sum of squared residuals by the estimator's degrees of freedom yields $\hat{\sigma}^2 = 304505/38 = 8013.29$.

The estimated variances and covariance of the least squares estimator can be obtained once the model is estimated by least squares by selecting the `Analysis>Coefficient covariance matrix` command from the pull-down menu of the **model** window as shown in Figure 2.13. The result is:

```
  Covariance matrix of regression coefficients:

          const        income
        1884.44      -85.9032  const
                      4.38175  income
```

So, estimated variances of the least squares estimator of the intercept and slope are 1884.44 and 4.38175, respectively. The least squares standard errors are simply the square roots of these numbers. The estimated covariance between the slope and intercept −85.9032.

You can also obtain the variance-covariance matrix by specifying the `--vcv` option when estimating a regression model. For the food expenditure example use:

```
ols food_exp const income --vcv
```

to estimate the model using least squares and to print the variance covariance matrix to the results window.

2.5 Repeated Sampling

Perhaps the best way to illustrate the sampling properties of least squares is through an experiment. In section 2.4.3 of your book you are presented with results from 10 different regressions (*POE4* Table 2.2). These were obtained using the dataset *table2-2.gdt* which is included in the **gretl** datasets that accompany this manual. To reproduce the results in this table you could estimate 10 separate regressions

```
open "@gretldir\data\poe\table2_2.gdt"
ols y1 const x
ols y2 const x
.
.
.
ols y10 const x
```

The ten regressions can be estimated more compactly using one of **gretl**'s loop constructs. The first step is to create a **list** that contains the variable names for the dependent variables as in line

1 of the script below. The statement `list ylist` is used to put data series into a collection called `ylist`; each of the series, `y1`, `y2`, ..., `y10` are included. Such named lists can be used to make your scripts less verbose and repetitious, and easier to modify. Since lists are in fact lists of series ID numbers they can be used only when a dataset is in place. The `foreach` loop in line 2 uses an index variable, `i`, to index a specified list of strings. The loop is executed once for each string in the list. The numerical value of the index starts at 1 and is incremented by 1 at each iteration. To refer to elements of the list, use the syntax `listname.$i`. Be sure to close the loop using `endloop`.

```
1 open "@gretldir\data\poe\table2_2.gdt"
2 list ylist =  y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
3 loop foreach i ylist
4     ols ylist.$i 0 1
5 endloop
```

In the **gretl** GUI, named lists can be inspected and edited under the **Data** menu in the main window, via the item Define or edit list. This dialog is shown in Figure 2.14

You can also generate your own random samples and conduct a Monte Carlo experiment using **gretl**. In this exercise you will generate 100 samples of data from the food expenditure data, estimate the slope and intercept parameters with each data set, and then study how the least squares estimator performed over those 100 different samples. What will become clear is this, the outcome from any single sample is a poor indicator of the true value of the parameters. Keep this humbling thought in mind whenever you estimate a model with what is invariably only 1 sample or instance of the true (but always unknown) data generation process.

We start with the food expenditure model:

$$food\_exp_t = \beta_1 + \beta_2 income_t + e_t \tag{2.6}$$

where $food\_exp_t$ is total food expenditure for the given time period and $income_t$ is income. Suppose further that we know how much income each of 40 households earns in a week. Additionally, we know that on average a household spends at least \$80 on food whether it has income or not and that an average household will spend ten cents of each new dollar of income on additional food. In terms of the regression this translates into parameter values of $\beta_1 = 80$ and $\beta_2 = 10$.

Our knowledge of any particular household is considerably less. We don't know how much it actually spends on food in any given week and, other than differences based on income, we don't know how its food expenditures might otherwise differ. Food expenditures are sure to vary for reasons other than differences in family income. Some families are larger than others, tastes and preferences differ, and some may travel more often or farther making food consumption more costly. For whatever reasons, it is impossible for us to know beforehand exactly how much any household will spend on food, even if we know how much income it earns. All of this uncertainty is captured by the error term in the model. For the sake of experimentation, suppose we also know that $e_t \sim N(0, 88^2)$.

With this knowledge, we can study the properties of the least squares estimator by generating samples of size 40 using the known data generation mechanism. We generate 100 samples using the known parameter values, estimate the model for each using least squares, and then use summary statistics to determine whether least squares, on average anyway, is either very accurate or precise. So in this instance, we know how much each household earns, how much the **average** household spends on food that is not related to income ($\beta_1 = 80$), and how much that expenditure rises **on average** as income rises. What we do not know is how any **particular** household's expenditures responds to income or how much is autonomous.

A single sample can be generated in the following way. The systematic component of food expenditure for the t^{th} household is $80 + 10 * income_t$. This differs from its actual food expenditure by a random amount that varies according to a normal distribution having zero mean and standard deviation equal to 88. So, we use computer generated random numbers to generate a random error, e_t, from that particular distribution. We repeat this for the remaining 39 individuals. The generates one Monte Carlo sample and it is then used to estimate the parameters of the model. The results are saved and then another Monte Carlo sample is generated and used to estimate the model and so on.

In this way, we can generate as many different samples of size 40 as we desire. Furthermore, since we know what the underlying parameters are for these samples, we can later see how close our estimators get to revealing these true values.

Now, computer generated sequences of random numbers are not actually random in the true sense of the word; they can be replicated exactly if you know the mathematical formula used to generate them and the 'key' that initiates the sequence. In most cases, these numbers *behave as if* they randomly generated by a physical process.

To conduct an experiment using least squares in **gretl** use the script found in below:

```
open "@gretldir\data\poe\food.gdt"
set seed 3213789
loop 100 --progressive --quiet
    series u = normal(0,88)
    series y1= 80+10*income+u
    ols y1 const income
endloop
```

Let's look at what each line accomplishes. The first line opens the food expenditure data set that resides in the `poe` folder of the data directory. The next line, which is actually not necessary to do the experiments, initiates the pseudo-random numbers at a specific point. This is useful, since it will allow you to get the same results each time you run the script.

In Monte Carlo experiments loops are used to estimate a model using many different samples that the experimenter generates and to collect the results. The simplest loop construct in **gretl** begins with the command `loop NMC --progressive --quiet` and ends with `endloop`. This is called a **count loop**. `NMC` in this case is the number of Monte Carlo samples you want to use and the option `--progressive` is a command that suppresses the individual output at each iteration from being printed; the `--quiet` option will suppress some printing to the screen as well.

Optionally you could add a couple of additional commands. The `print` command collects (scalar) statistics that you have computed and finds their averages and standard deviations. The `store` command allows you to store these in a **gretl** data file. These are discussed further below.

Within the loop itself, you tell **gretl** how to generate each sample and state how you want that sample to be used. The `series` command is used to generate new variables. In the first line u is generated using the **gretl** command `normal()`, which when used without arguments produces a computer generated standard normal random variable. In this case, the function contains two arguments (e.g., `series u = normal(0,88)`). The `normal` function takes an ordered pair as inputs (commonly referred to as 'arguments'), the first of which is the desired mean of the random normal and the second is the standard deviation. The next line adds this random element to the systematic portion of the model to generate a new sample for food expenditures (using the known values of income from the dataset).

Next, the model is estimated using least squares. After executing the script, **gretl** prints out some summary statistics to the screen. These appear as a result of using the `--progressive` loop option. The result appears in Figure 2.15. Note that the average value of the intercept is about 88.147. This is getting close to the truth. The average value of the slope is 9.55972, also reasonably close to the true value. If you were to repeat the experiments with larger numbers of Monte Carlo iterations, you will find that these averages get closer to the values of the parameters used to generate the data. This is what it means to be **unbiased**. Unbiasedness only has meaning within the context of repeated sampling. In your experiments, you generated many samples and averaged results over those samples to get close to finding the truth. In actual practice, you do not have this luxury; you have one sample and the proximity of your estimates to the true values of the parameters is always unknown.

In section 2.8 and in the script at the end of this chapter, you will find another example of Monte Carlo that is discussed in *POE4*. In this example, a sample of regressors is generated using a simple loop and the properties of least squares is examined using 1000 samples. The use of the `print` and `store` commands will be examined in section 2.8 as well.

2.6 Estimating Nonlinear Relationships

Since economic relationships are often not linear, we often need to allow for the possibility that the independent and dependent variable are nonlinearly related. Consider the following simple regression

$$price = \beta_1 + \beta_2 sqft + e \tag{2.7}$$

The parameter, β_2 measures the expected change in *price* given an additional square foot of living space in the home. As specified, this marginal effect is the same for homes of every size. It might make more sense to allow the size of this marginal effect to depend on the size of the house. Larger houses also tend to be more luxurious and therefore another square foot of living area might add more to the average home price. This can be modeled by using a quadratic term in the model.

$$price = \alpha_1 + \alpha_2 sqft^2 + e \tag{2.8}$$

The marginal effect of another square foot is now $\partial price/\partial sqft = 2\alpha_2 sqft$. The estimated elasticity is equal to

$$\hat{\varepsilon} = \widehat{slope} \times \frac{sqft}{price} = (2\hat{\alpha}_2) \times \frac{sqft}{price} \tag{2.9}$$

Obviously, the slope and elasticity depend on the size and price of the home. Thus, the user must select values at which these are to be evaluated. This is done in the script below where sloped for houses of size 2000, 4000, and 6000 square feet are computed. The elasticities are computed for prices of \$117461.77, \$302517.39, and \$610943.42. The scalar and series that are used are not strictly necessary in **gretl**. I've used them here to make things more clear and it is a good programming practice in general.

```
open "@gretldir\data\poe\br.gdt"
series sqft2 = sqft^2
ols price const sqft2
scalar slope_2000 = 2*$coeff(sqft2)*2000
scalar slope_4000 = 2*$coeff(sqft2)*4000
scalar slope_6000 = 2*$coeff(sqft2)*6000
scalar elast_2000 = slope_2000*2000/117461.77
scalar elast_4000 = slope_4000*4000/302517.39
scalar elast_6000 = slope_6000*6000/610943.42
```

The output from the regression is

$$\widehat{\text{price}} = \underset{(2890.4)}{55776.6} + \underset{(0.00031310)}{0.0154213}\,\text{sqft2}$$

$$T = 1080 \quad \bar{R}^2 = 0.6921 \quad F(1, 1078) = 2426.0 \quad \hat{\sigma} = 68207.$$

(standard errors in parentheses)

and the graph of home price against size is shown in Figure 2.16.

Another way to estimate a nonlinear relationship between *price* and *sqft* is to alter the functional form of the model. A log-linear model uses the logarithm of a variable as the dependent variable, and the untransformed value of regressor as the independent variable. In the simple home price model this is

$$\ln price = \gamma_1 + \gamma_2 sqft + e \tag{2.10}$$

The logarithmic transformation is often used on data that come from a heavily skewed distribution that has a long-tail to the right. Taking a look at the histograms for *price* and it natural logarithm shown in Figure 2.17 reveals just this sort of data and how the natural log can 'regularize' the series. These graphs were produced by first taking the natural log and then using the `freq` function to generate the graphs. The code is

```
series l_price = ln(price)
freq price
freq l_price
```

2.7 Regression with an Indicator Variable

An indicator variable is a variable that can be equal to one of two possible values. Commonly, this an indicator variable can be a 1 or a 0. So for instance, if a house is located in the University Town subdivision the variable is given the value of 1 and if not it is equal to 0.

$$utown = \begin{cases} 1 & \text{if house is in University Town} \\ 0 & \text{if not} \end{cases} \tag{2.11}$$

The regression model becomes

$$price = \beta_1 + \beta_2 utown + e \tag{2.12}$$

As pointed out in *POE4*, taking the expected value of a regression is very useful when it contains an indicator variable. This will reveal how to interpret its coefficient. In this model

$$E(pri\ CE) = \beta_1 + \beta_2 utown = \begin{cases} \beta_1 + \beta_2 & \text{if } utown = 1 \\ \beta_1 & \text{if } utown = 0 \end{cases} \tag{2.13}$$

So, estimating the model using the *utown.gdt* data yields

$$\widehat{\text{price}} = \underset{(1.3181)}{215.732} + \underset{(1.8296)}{61.5091}\,\text{utown}$$
$$T = 1000 \quad \bar{R}^2 = 0.5306 \quad F(1,998) = 1130.2 \quad \hat{\sigma} = 28.907$$
$$\text{(standard errors in parentheses)}$$

This implies that the average home price (in \$1000) in University Town is $215.7325 + 61.5091 = 277.2416$ and the average price elsewhere is 215.7325.

The script that produces the same result is straightforward:

```
open "@gretldir\data\poe\utown.gdt"
ols price const utown --quiet
scalar ut = $coeff(const)+$coeff(utown)
scalar other = $coeff(const)
printf "\nThe average in Utown is %.4f and the average elsewhere is %.4f\n",ut,other
```

2.8 Monte Carlo Simulation

The first step in a Monte Carlo exercise is to model the data generation process. This requires what Davidson and MacKinnon (2004) refer to as a fully specified statistical model. A **fully specified parametric model** "is one for which it is possible to simulate the dependent variable once the values of the parameters are known" (Davidson and MacKinnon, 2004, p. 19). First you'll need a regression function, for instance:

$$E(y_t|\Omega_t) = \beta_1 + \beta_2 x_t \tag{2.14}$$

where y_t is your dependent variable, x_t the dependent variable, Ω_t the current information set, and β_1 and β_2 the parameters of interest. The information set Ω_t contains x_t as well as other potential explanatory variables that determine the average of y_t. The conditional mean of y_t given the information set could represent a linear regression model or a discrete choice model. However, equation (2.14) is not complete; it requires some description of how the unobserved or excluded factors affect $y_t|\Omega_t$.

To complete the the specification we need to specify an "unambiguous recipe" for simulating the model on a computer (Davidson and MacKinnon, 2004, p. 17). This means we'll need to specify a probability distribution for the unobserved components of the model and then use a pseudo-random number generator to generate samples of the desired size.

In this example the data generation process will be as follows. We will let $N = 40$ and consider a linear model of the form

$$y_i = \beta_1 + \beta_2 x_i + e_i \quad i = 1, 2, \cdots, 40. \tag{2.15}$$

The errors of the model will *iid* $N(0, 88)$. The parameters $\beta_1 = 100$ and $\beta_2 = 10$. Finally, let $x_1, x_2, \cdots, x_{20} = 10$ and let $x_{21}, x_{22}, \cdots, x_{40} = 20$. This gives us enough information to simulate samples of y_i from the model. The **hansl** script (**hansl** is an acronym for hansl's a neat scripting language is:

```
nulldata 40
# Generate X
series x = (index>20) ? 20 : 10

# Generate systematic portion of model
series ys = 100 + 10*x

loop 1000 --progressive --quiet
    y = ys + normal(0,50)
    ols y const x
    scalar b1 = $coeff(const)
    scalar b2 = $coeff(x)
    scalar sig2 = $sigma^2
    print b1 b2 sig2
```

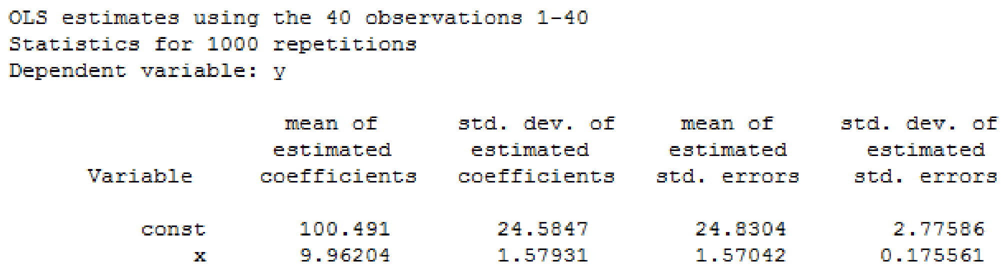
```
15     store "@workdir\coef.gdt" b1 b2 sig2
16 endloop
17
18 open "@workdir\coef.gdt"
19 summary
20 freq b2 --normal
```

The first line creates an empty dataset that has room for 40 observations. Line 3 contains a ternary **conditional assignment operator**.[2] Here is how it works. A series `x` is being created. The statement in parentheses is checked. The question mark (`?`) is the conditional assignment. If the statement in parentheses is true, then `x` is assigned the value to the left of the colon. If false it gets the value to the right. So, when `index` (a **gretl** default way of identifying the observation number) is greater than 20, `x` is set to 20, if index is less than or equal to 20 it is set to 10.

Next, the systematic portion of the model is created. For this we need `x` and the known values of the parameters (100, 10). Then we loop from 1 to 1000 in increments of 1. Normal random variates are added to the model, it is estimated by `ols`, and several statistics from that computation are retrieved, printed, and stored in a specified location.

The `normal(0,50)` statement generates normal random variables with mean of 0 and a variance of 50. The `print` statement used in this context actually tells **gretl** to accumulate the things that are listed and to print out summary statistics from their computation inside the loop. The `store` command tells **gretl** to output `b1`, `b2`, and `sig2` to an external file. The `--progressive` option to the loop command alters the `print` and `store` commands a bit, and you can consult the **Gretl** Users Guide for more information about how.

Here is the output from the Monte Carlo. First, the output from the progressive loop:

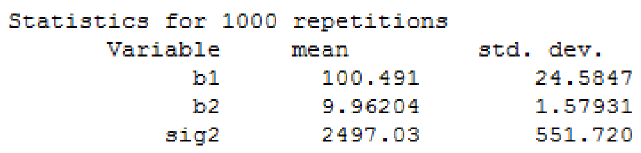


```
OLS estimates using the 40 observations 1-40
Statistics for 1000 repetitions
Dependent variable: y

                    mean of      std. dev. of     mean of     std. dev. of
                   estimated      estimated      estimated     estimated
      Variable    coefficients   coefficients   std. errors   std. errors

         const      100.491        24.5847        24.8304       2.77586
             x      9.96204        1.57931        1.57042       0.175561
```

In a progressive loop, **gretl** will print out the mean and standard deviation from the series of estimates. It works with all single equation estimators in **gretl** and is quite useful for Monte Carlo analysis. From this you can see that the average value of the constant in 1000 samples is 100.491. The average slope was 9.962. The third column gives the mean of the standard error calculation

[2]A ternary operator has three parts. In this case, the parts give us a fancy way of creating if/else statements. The first part, `a`, lies to the left of `?`, the second, `b`, falls between the question mark and the colon and the last, `c`, is to the right of the colon, e.g., `a?b:c`. If `a` is true, then `b` if not, then `c`.

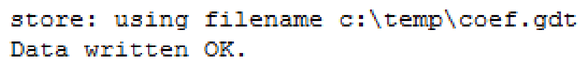
from the simulation. If the standard errors are being estimated consistently, then these should be fairly close to the standard deviation of estimated coefficients to their left. The outcome from the `print` command is:



```
Statistics for 1000 repetitions
      Variable     mean         std. dev.
            b1       100.491        24.5847
            b2       9.96204        1.57931
          sig2       2497.03        551.720
```

When the `print` command is issued, it will compute and print to the screen the 'mean' and 'std. dev.' of the estimated scalar. Notice that `b1` and `b2` match the output produced by the `--progressive` option. The `print` command is useful for studying the behavior of various statistics (like tests, confidence intervals, etc) and other estimators that cannot be handled properly within a progressive loop (e.g., `mle`, `gmm`, and `system` estimation commands).

The `store` statement works behind the scenes, but yields this informative piece of information:



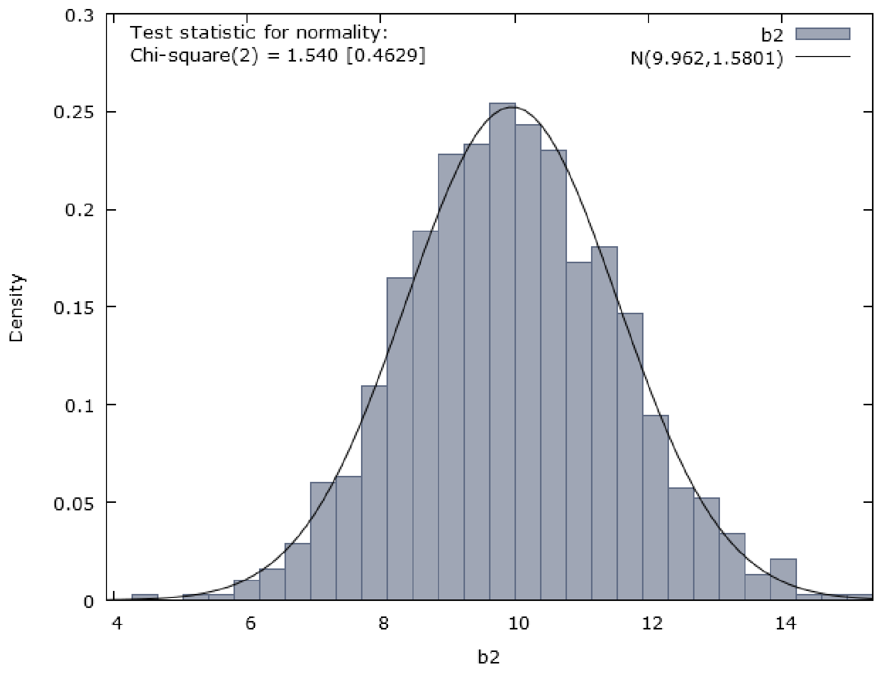
```
store: using filename c:\temp\coef.gdt
Data written OK.
```

This tells you where **gretl** wrote the dataset that contains the listed scalars, and that is was written properly. Now you are ready to open it up and perform additional analysis. In this example, we have used the `@workdir` macro. This basically tells **gretl** to go to the working directory to write the file. You could write files to **gretl**'s temporary directory using `@dotdir\coef.gdt`.

The data set is opened and the summary statistics generated (again, if needed)

```
open "@workdir\coef.gdt"
summary
freq b2 --normal
```

From here you can plot frequency distribution and test to see whether the least squares estimator of slope is normally distributed.



The histogram certainly appears to be normally distributed compared to the line plot of the normal. Also, the hypothesis test of the normality null against nonnormality cannot be rejected at any reasonable level of significance.

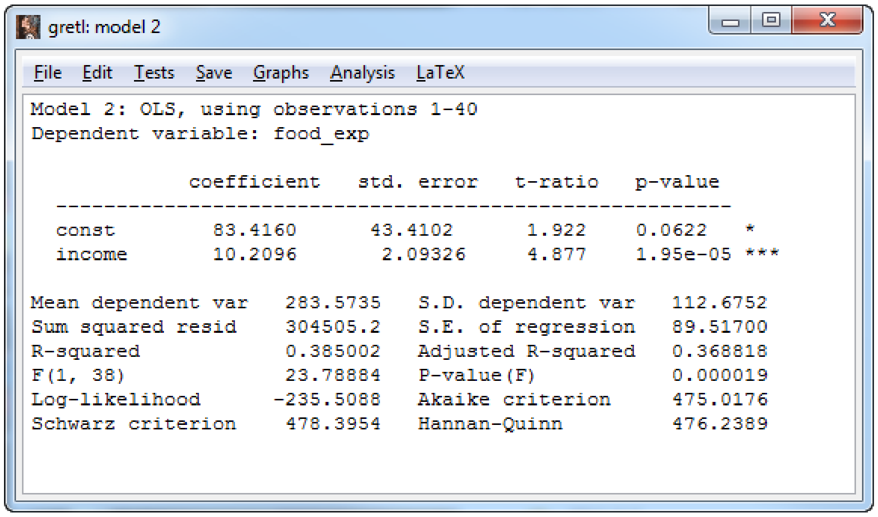
2.9 Script

The script for chapter 2 is found below. These scripts can also be found at my website http://www.learneconometrics.com/gretl.

```
set echo off
open "@gretldir\data\poe\food.gdt"
setinfo food_exp -d "household food expenditure per week" \
    -n "Food Expenditure"
setinfo income -d "weekly household income" -n "Weekly Income"
labels

#Least squares
ols food_exp const income --vcv
ols 1 0 2

#Summary Statistics
summary food_exp income

```

```
#Plot the Data
gnuplot food_exp income

#List the Data
print food_exp income --byobs

#Elasticity
ols food_exp const income --quiet
scalar elast=$coeff(income)*mean(income)/mean(food_exp)

#Prediction
scalar yhat = $coeff(const) + $coeff(income)*20

#Table 2.2
open "@gretldir\data\poe\table2_2.gdt"
list ylist =  y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
loop foreach i ylist
    ols ylist.$i const x
endloop

# slopes and elasticities at different points
open "@gretldir\data\poe\br.gdt"
series sqft2 = sqft^2
ols price const sqft2
scalar slope_2000 = 2*$coeff(sqft2)*2000
scalar slope_4000 = 2*$coeff(sqft2)*4000
scalar slope_6000 = 2*$coeff(sqft2)*6000
scalar elast_2000 = slope_2000*2000/117461.77
scalar elast_4000 = slope_4000*4000/302517.39
scalar elast_6000 = slope_6000*6000/610943.42

# histogram for price and log(price)
series l_price = ln(price)
freq price
freq l_price

# regression using indicator variables
open "@gretldir\data\poe\utown.gdt"
ols price const utown --quiet
scalar ut = $coeff(const)+$coeff(utown)
scalar other = $coeff(const)
printf "\nThe average in Utown is %.4f and the \
average elsewhere is %.4f\n",ut,other

# Monte Carlo simulation
open "@gretldir\data\poe\food.gdt"
set seed 3213789
loop 100 --progressive --quiet
    series u = normal(0,88)
    series y1= 80+10*income+u
    ols y1 const income
```

```
endloop

# Monte Carlo simulation #2
# Generate systematic portion of model
nulldata 40
# Generate X
series x = (index>20) ? 20 : 10

# Generate systematic portion of model
series ys = 100 + 10*x

loop 1000 --progressive --quiet
    series y = ys + normal(0,50)
    ols y const x
    scalar b1 = $coeff(const)
    scalar b2 = $coeff(x)
    scalar sig2 = $sigma^2
    print b1 b2 sig2
    store "@workdir\coef.gdt" b1 b2 sig2
endloop

open "@workdir\coef.gdt"
summary
freq b2 --normal
```



```
Model 2: OLS, using observations 1-40
Dependent variable: food_exp

             coefficient   std. error   t-ratio   p-value
  ---------------------------------------------------------
  const        83.4160      43.4102      1.922    0.0622    *
  income       10.2096       2.09326     4.877    1.95e-05 ***

Mean dependent var   283.5735   S.D. dependent var   112.6752
Sum squared resid    304505.2   S.E. of regression   89.51700
R-squared            0.385002   Adjusted R-squared   0.368818
F(1, 38)             23.78884   P-value(F)           0.000019
Log-likelihood      -235.5088   Akaike criterion     475.0176
Schwarz criterion    478.3954   Hannan-Quinn         476.2389
```

Figure 2.9: The model window appears with the regression results. From here you can conduct subsequent operations (graphs, tests, analysis, etc.) on the estimated model.

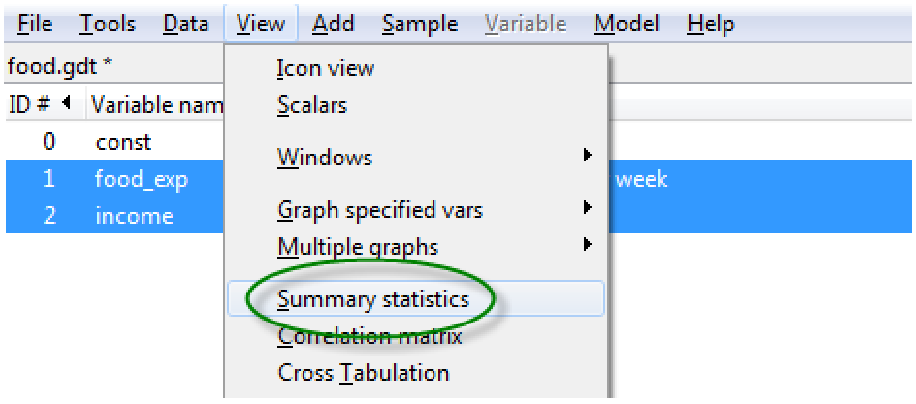
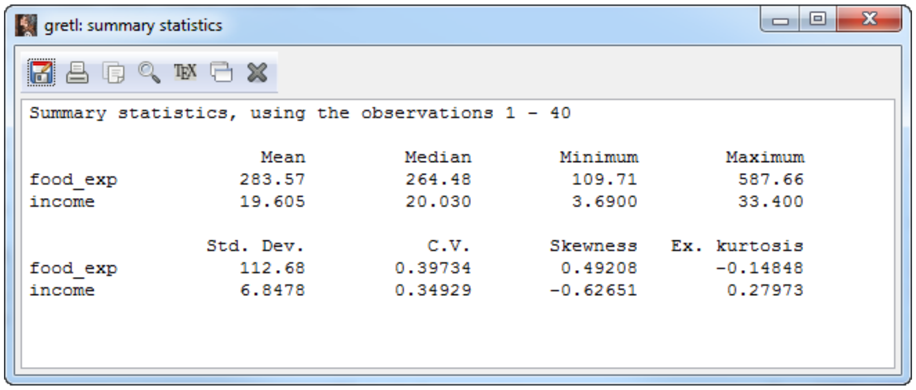
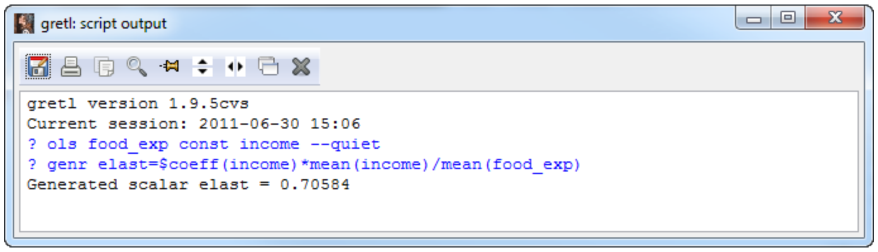


Figure 2.10: Using the pull-down menus to obtain summary statistics. Highlight the desired variables and use `View>Summary statistics` from the pull-down menu.



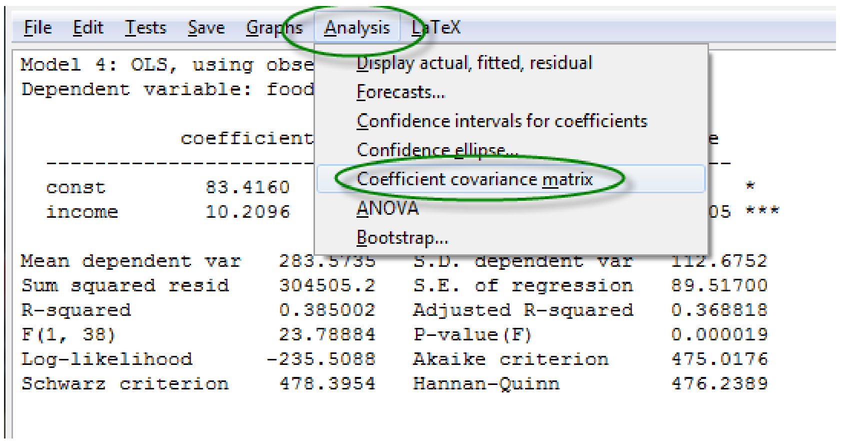
```
Summary statistics, using the observations 1 - 40

                     Mean         Median        Minimum        Maximum
food_exp           283.57         264.48         109.71         587.66
income             19.605         20.030         3.6900         33.400

                 Std. Dev.          C.V.        Skewness    Ex. kurtosis
food_exp           112.68        0.39734        0.49208       -0.14848
income             6.8478        0.34929       -0.62651        0.27973
```

Figure 2.11: Summary statistics



```
gretl version 1.9.5cvs
Current session: 2011-06-30 15:06
? ols food_exp const income --quiet
? genr elast=$coeff(income)*mean(income)/mean(food_exp)
Generated scalar elast = 0.70584
```

Figure 2.12: Results from the script to compute an elasticity based on a linear regression.


File
Edit
Tests
Save
Graphs
Analysis
LaTeX
Display actual, fitted, residual
Forecasts...
Confidence intervals for coefficients
Confidence ellipse...
Coefficient covariance matrix
ANOVA
Bootstrap...

```
Model 4: OLS, using obse
Dependent variable: food

              coefficient                                    e
  ------------------------                                  --
  const         83.4160                                       *
  income        10.2096                                   05 ***

Mean dependent var    283.5735   S.D. dependent var    112.6752
Sum squared resid     304505.2   S.E. of regression    89.51700
R-squared             0.385002   Adjusted R-squared    0.368818
F(1, 38)              23.78884   P-value(F)            0.000019
Log-likelihood       -235.5088   Akaike criterion      475.0176
Schwarz criterion     478.3954   Hannan-Quinn          476.2389
```

Figure 2.13: Obtain the matrix that contains the least squares estimates of variance and covariance from the pull-down menu of your estimated model.

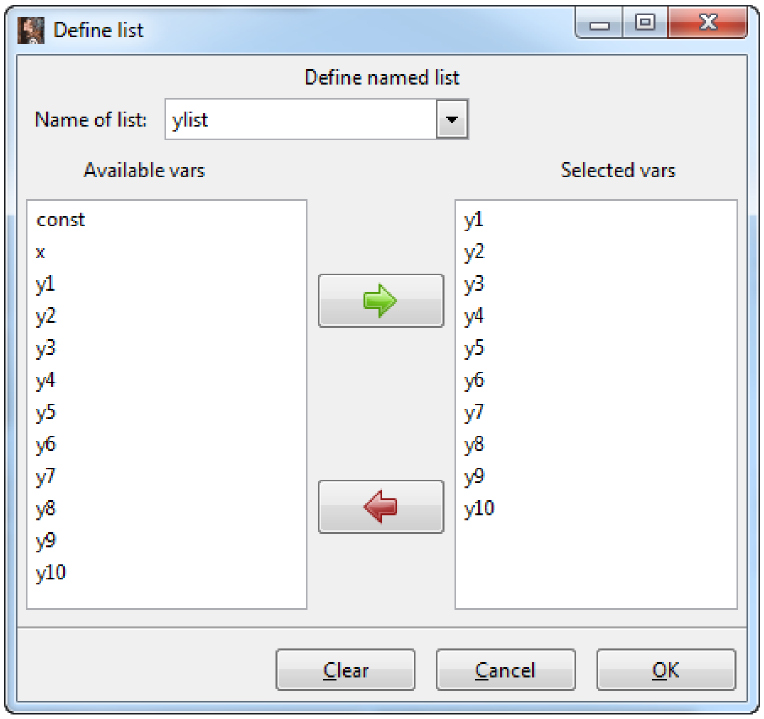


Figure 2.14: Choose `Data>Define or edit list` from the **gretl** menu bar

```
OLS estimates using the 40 observations 1-40
Statistics for 100 repetitions
Dependent variable: y1

                     mean of      std. dev. of     mean of     std. dev. of
                    estimated      estimated      estimated      estimated
      Variable    coefficients   coefficients    std. errors    std. errors

         const       88.1474        40.3705        42.1194        4.49704
        income       9.59723        2.01529        2.03102        0.216850

Statistics for 100 repetitions
       Variable      mean         std. dev.
            b1       88.1474        40.3705
            b2       9.59723        2.01529

store: using filename c:\temp\coeff.gdt
Data written OK.
```

Figure 2.15: The summary results from 100 random samples of the Monte Carlo experiment.

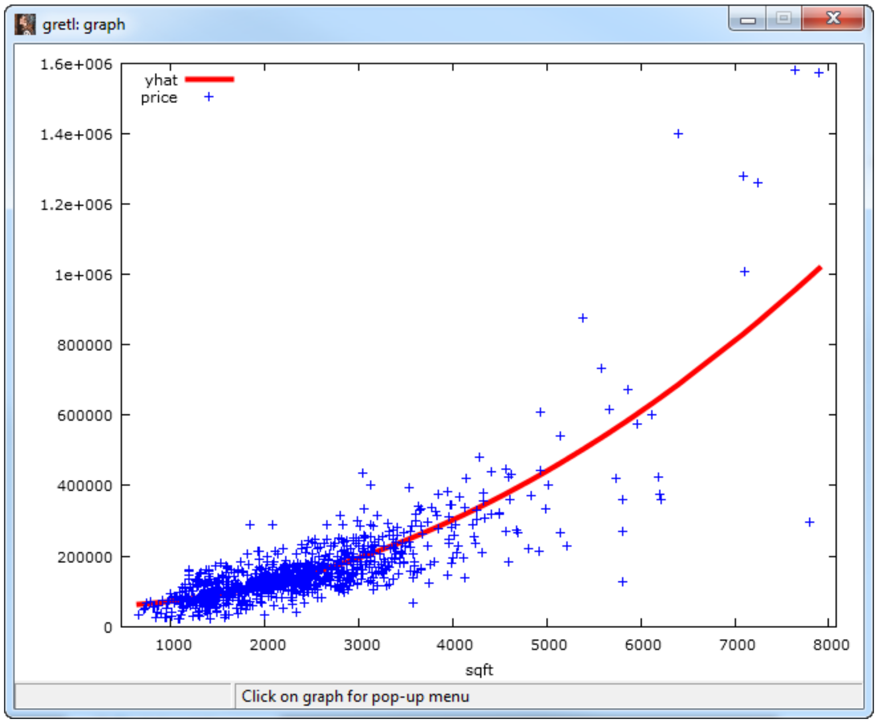


Figure 2.16: Price versus size from the quadratic model.

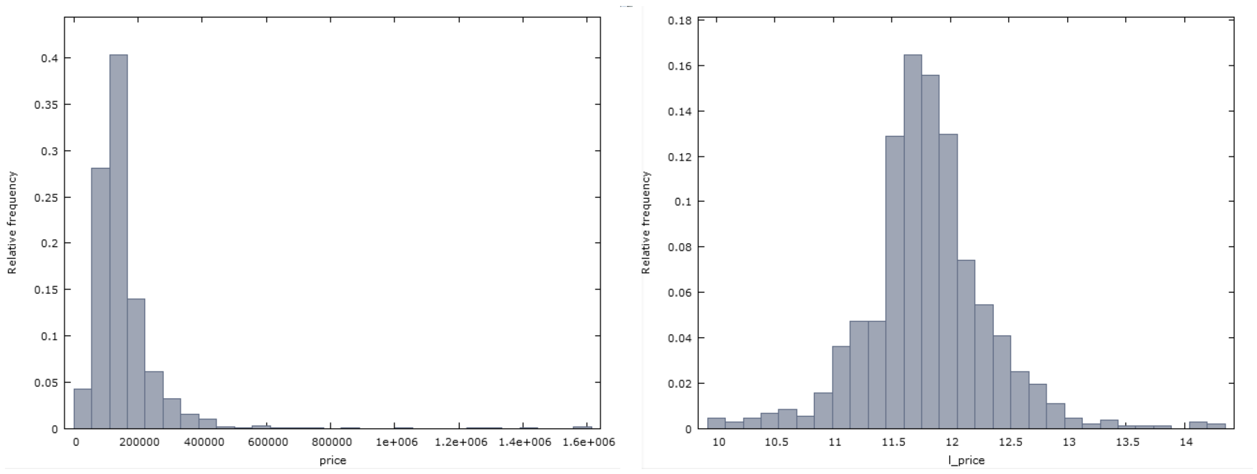


Figure 2.17: Price and its natural logarithm.

Chapter 3

Interval Estimation and Hypothesis Testing

In this chapter, I will discuss how to generate confidence intervals and test hypotheses using **gretl**. **Gretl** includes several handy utilities that will help you obtain critical values and p-values from several important probability distributions. As usual, you can use the dialog boxes or **hansl** – **gretl**'s programming language – to do this.

3.1 Confidence Intervals

It is important to know how precise your knowledge of the parameters is. One way of doing this is to look at the least squares parameter estimate along with a measure of its precision, i.e., its estimated standard error. The confidence interval serves a similar purpose, though it is much more straightforward to interpret because it gives you upper and lower bounds between which the unknown parameter will lie with a given probability.[1]

In **gretl** you can obtain confidence intervals either through a dialog or by manually building them using saved regression results. In the 'manual' method one can use the `genr` or `scalar` commands to generate upper and lower bounds based on regression results that are saved in **gretl**'s memory, letting **gretl** do the arithmetic. You can either look up the appropriate critical value from a table or use the **gretl**'s `critical` function. Both are demonstrated below.

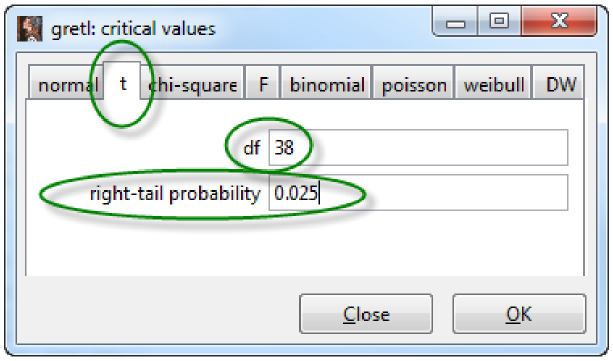
[1]This is probability in the frequency sense. Some authors fuss over the exact interpretation of a confidence interval (unnecessarily I think). You are often given stern warnings not to interpret a confidence interval as containing the unknown parameter with the given probability. However, the frequency definition of probability refers to the long run relative frequency with which some event occurs. If this is what probability is, then saying that a parameter falls within an interval with given probability means that intervals so constructed will contain the parameter that proportion of the time.

Consider the equation of a confidence interval from *POE4*

$$P[b_k - t_c se(b_k) \leq \beta_k \leq b_k + t_c se(b_k)] = 1 - \alpha \tag{3.1}$$

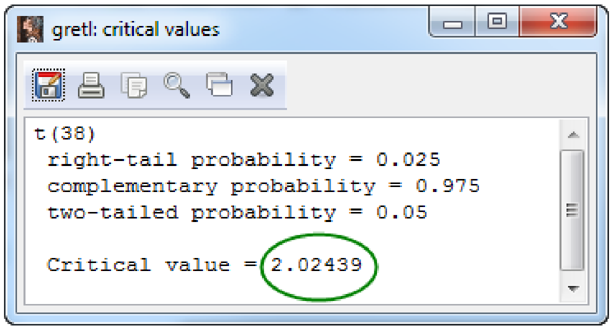
Recall that b_k is the least squares estimator of β_k, and that $se(b_k)$ is its estimated standard error. The constant t_c is the $\alpha/2$ critical value from the t-distribution and α is the total desired probability associated with the "rejection" area (the area outside of the confidence interval).

You'll need to know the critical value t_c, which can be obtained from a statistical table, the `Tools>Statistical tables` dialog contained in the program, or using the **gretl** command `critical`. First, try using the dialog box shown in Figure 3.1. Pick the tab for the t distribution and tell **gretl** how much weight to put into the right-tail of the probability distribution and how many degrees of freedom your t-statistic has, in our case, 38. Once you do, click on **OK**. You'll get the result shown in Figure 3.2. It shows that for the $t(38)$ with $\alpha/2$ right-tail probability of 0.025 and $\alpha = 0.05$, the critical value is 2.02439.[2] Then generate the lower and upper bounds (using



df 38
right-tail probability 0.025

Figure 3.1: Obtaining critical values using the `Tools>Statistical tables` dialog box.



```
t(38)
 right-tail probability = 0.025
 complementary probability = 0.975
 two-tailed probability = 0.05

 Critical value = 2.02439
```

Figure 3.2: The critical value obtained from `Tools>Statistical tables` dialog box.

the **gretl** console) with the commands:

[2] You can also get the α level critical values from the console or in a script by issuing the command `scalar c = critical(t,38,`α`)`. Here α is the desired area in the right-tail of the t-distribution.

```
1 open "@gretldir\data\poe\food.gdt"
2 ols food_exp const income
3 scalar lb = $coeff(income) - 2.024 * $stderr(income)
4 scalar ub = $coeff(income) + 2.024 * $stderr(income)
5 print lb ub
```

The first line opens the dataset. The second line (`ols`) solves for the estimates that minimize the sum of squared errors in a linear model that has `food_exp` as the dependent variable with a constant and `income` as independent variables. The next two lines generate the lower and upper bounds for the 95% confidence interval for the slope parameter β_2. The last line prints the results of the computation.

The **gretl** language syntax needs a little explanation. When **gretl** makes a computation, it will save certain results like coefficient estimates, their standard errors, sum of squared errors and so on in memory. These results can then be used to compute other statistics, provided you know the accessor's name that **gretl** uses to store and recall the computation. These so-called `accessors` carry `$` prefixes and a list of what can be accessed after estimation can be found in the function reference. Lines 3 and 4 use accessors for the coefficients (`$coeff(income)`) and standard errors (`$stderr(income)`) of the variable in parentheses. The list of accessors is actually growing quite rapidly in response to user requests, so a trip to the function reference may be worth your while to see what is available.

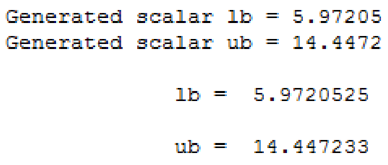
In the above example, **gretl** uses the least squares estimates and their estimated standard errors to compute confidence intervals. Following the `ols` command, least squares estimates are stored in `$coeff(`*`variable name`*`)`. Since β_2 is estimated using the variable `income`, its coefficient estimate is saved in `$coeff(income)`. The corresponding standard error is saved in `$stderr(income)`. Also, don't forget that the function reference (Figure 1.10) includes a list of **accessors**.

Equivalently, you could use **gretl**'s built-in `critical` function to obtain the desired critical value. The general syntax for the function depends on the desired probability distribution. This follows since different distributions contain different numbers of parameters (e.g., the t-distribution has a single degrees of freedom parameter while the standard normal has none!). This example uses the t-distribution and the script becomes:

```
1 open "@gretldir\data\poe\food.gdt"
2 ols food_exp const income
3 scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
4 scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
5 print lb ub
```

The syntax for the t-distribution is `critical(t,degrees of freedom,`$\alpha/2$`)`. The degrees of freedom from the preceding regression are saved as `$df` and for a $1-\alpha = 95\%$ confidence interval, $\alpha/2 = 0.025$.

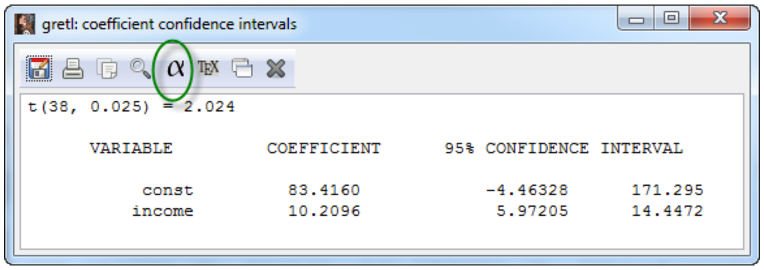
The example found in section 3.1.3 of *POE4* computes a 95% confidence interval for the income parameter in the food expenditure example. The **gretl** commands above were used to produce the output found in Figure 3.3.



```
Generated scalar lb = 5.97205
Generated scalar ub = 14.4472

            lb =  5.9720525

            ub =  14.447233
```

Figure 3.3: Obtaining the 95% confidence interval for the income coefficient in the food expenditure example.

To use the dialogs to get confidence intervals is easy as well. First estimate the model using least squares in the usual way. Choose `Model>Ordinary least squares` from the main pull-down menu, fill in the dependent and independent variables in the `ols` dialog box and click **OK**. The results appear in the **model window**. Now choose `Analysis>Confidence intervals for coefficients` from the model window's pull-down menu (seen in Figure 4.1). This generates the result shown in Figure 3.4. The circled α icon can be used to change the size of the confidence



```
t(38, 0.025) = 2.024

     VARIABLE         COEFFICIENT      95% CONFIDENCE INTERVAL

        const          83.4160        -4.46328       171.295
       income          10.2096         5.97205       14.4472
```

Figure 3.4: The 95% confidence interval for the income coefficient in the food expenditure example using the dialog.

interval, which can be set to any (integer) percentage level you desire.

3.2 Repeated Sampling

In this section, the ten samples in *table2_2.gdt* are used to produce ten sets of 95% confidence intervals. To make the program simpler, the loop construct introduced in chapter 2 is employed. The script to estimate these in the loop is:

```
open "@gretldir\data\poe\table2_2.gdt"
list ylist = y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
loop foreach i ylist --progressive --quiet
```

```
4    ols ylist.$i const x
5    scalar b1 = $coeff(const) # in gretl you can use genr or scalar
6    scalar b2 = $coeff(x)
7    scalar s1 = $stderr(const)
8    scalar s2 = $stderr(x)
9
10  # 2.024 is the .025 critical value from the t(38) distribution
11   scalar c1L = b1 - critical(t,$df,.025)*s1
12   scalar c1R = b1 + critical(t,$df,.025)*s1
13   scalar c2L = b2 - critical(t,$df,.025)*s2
14   scalar c2R = b2 + critical(t,$df,.025)*s2
15
16   scalar sigma2 = $sigma^2
17   store @workdir\coeff.gdt b1 b2 s1 s2 c1L c1R c2L c2R sigma2
18  endloop
```

As in chapter 2, the dataset is opened and a `list` is created that contains each of the ten samples of the dependent variable. The `foreach` loop is initiated in line 3 and the `--progressive` and `--quiet` options are chosen. The model is estimated using least squares and the coefficients, standard errors, lower and upper confidence limits and variance are generated and stored in the dataset *coeff.gdt*, which is placed in `c:\temp` on the harddrive.

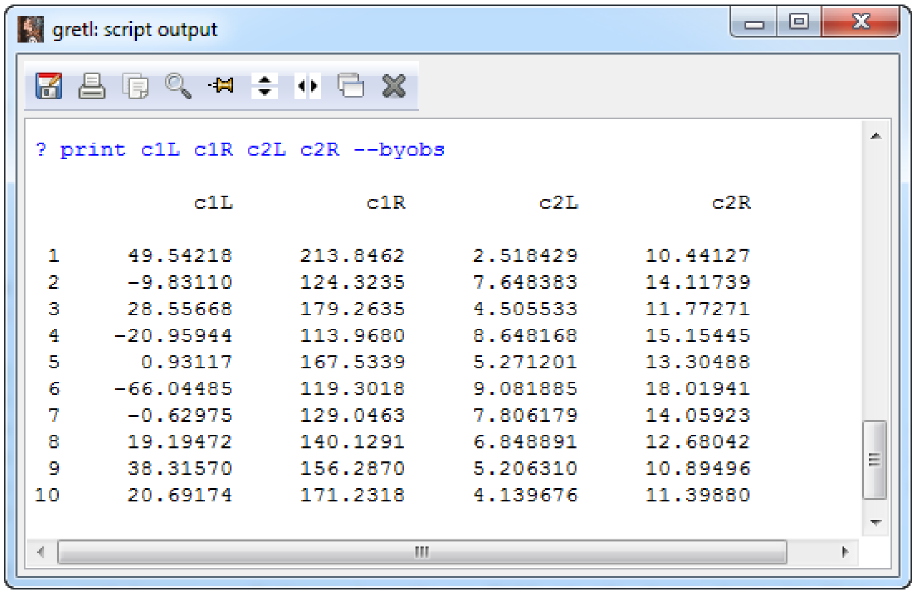
As if that is not easy enough, there is an even simpler syntax that will accomplish the same thing. It uses the fact that the dependent variables all begin with the letter 'y' and have number suffixes. In this case the `foreach` loop can be simplified by replacing lines 2-4 with:

```
loop foreach i y1..y10
   ols $i const x
```

Once this is executed, one can open *coeff.gdt* and perform further analysis. In this case, I will print the upper and lower confidence bounds as Hill et al. have done in their Table 3.2 of *POE4*.

```
open @workdir\coeff.gdt
print c1L c1R c2L c2R --byobs
```

The `--byobs` option is used with the `print` command, otherwise each of the series will be printed out separately. The result appears below in Figure 3.5. Recall that the true value of $\beta_2 = 10$ and each of the estimated intervals contains it. The actual value of the intercept is 80, and β_1 falls also falls within the estimated boundaries in each of the samples. In a large number of samples, we would expect about 5% of the intervals would not contian the true value of the parameters. This is explored in the next section.



```
? print c1L c1R c2L c2R --byobs

             c1L          c1R          c2L          c2R

 1      49.54218     213.8462     2.518429     10.44127
 2      -9.83110     124.3235     7.648383     14.11739
 3      28.55668     179.2635     4.505533     11.77271
 4     -20.95944     113.9680     8.648168     15.15445
 5       0.93117     167.5339     5.271201     13.30488
 6     -66.04485     119.3018     9.081885     18.01941
 7      -0.62975     129.0463     7.806179     14.05923
 8      19.19472     140.1291     6.848891     12.68042
 9      38.31570     156.2870     5.206310     10.89496
10      20.69174     171.2318     4.139676     11.39880
```

Figure 3.5: Confidence intervals for 10 samples.

3.3 Monte Carlo Experiment

Once again, the consequences of repeated sampling can be explored using a simple Monte Carlo study. In this case, we will generate 100 samples and count the number of times the confidence interval includes the true value of the parameter. The simulation will be based on the *food.gdt* dataset.

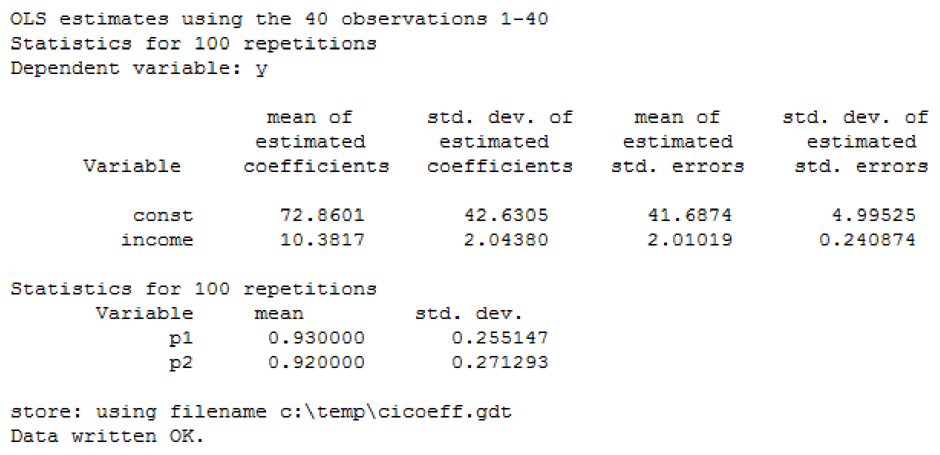
The new script looks like this:

```
open "@gretldir\data\poe\food.gdt"
set seed 3213798
loop 100 --progressive --quiet
    series u = normal(0,88)
    series y = 80 + 10*income + u
    ols y const income

    scalar c1L = $coeff(const) - critical(t,$df,.025)*$stderr(const)
    scalar c1R = $coeff(const) + critical(t,$df,.025)*$stderr(const)
    scalar c2L = $coeff(income) - critical(t,$df,.025)*$stderr(income)
    scalar c2R = $coeff(income) + critical(t,$df,.025)*$stderr(income)

    # Compute the coverage probabilities of the Confidence Intervals
    scalar p1 = (80>c1L && 80<c1R)
    scalar p2 = (10>c2L && 10<c2R)

    print p1 p2
```

```
18     store @workdir\cicoeff.gdt c1L c1R c2L c2R
19 endloop
```

The results are stored in the **gretl** data set *cicoeff.gdt*. Opening this data set (`open @workdir\cicoeff.gdt`) and examining the data will reveal interval estimates that vary much like those in Tables 3.1 and 3.2 of *POE4*. In line 4 of this script pseudo-random normals are drawn using the `normal(mean,sd)` command, and the mean has been set to 0 and the standard deviation to 88. The samples of `y` are generated linearly (`80+10*food_exp`) to which the random component is added in line 5. Then, the upper and lower bounds are computed. In lines 14 and 15 **gretl**'s "and" logical operator, `&&`, is used to determine whether the coefficient (80 or 10) falls within the computed bounds. The operator `&&` actually yields the intersection of two sets so that if 80 is greater than the lower bound and smaller than the upper `p1`, then the condition is true and `p1` is equal to 1. If the statement is false, it is equal to zero. Averaging `p1` and `p2` gives you the proportion of times in the Monte Carlo that the condition is true, which amounts to the empirical coverage rate of the computed interval.

With this seed, I get the following (Figure 3.6) result: You can see that the intercept falls within



```
OLS estimates using the 40 observations 1-40
Statistics for 100 repetitions
Dependent variable: y

                     mean of      std. dev. of     mean of     std. dev. of
                    estimated      estimated      estimated     estimated
      Variable     coefficients   coefficients    std. errors   std. errors

         const       72.8601        42.6305        41.6874       4.99525
        income       10.3817        2.04380        2.01019       0.240874

Statistics for 100 repetitions
       Variable      mean          std. dev.
             p1     0.930000       0.255147
             p2     0.920000       0.271293

store: using filename c:\temp\cicoeff.gdt
Data written OK.
```

Figure 3.6: The empirical coverage rates of nominal 95% confidence intervals from 100 random samples.

the estimated interval 93 out of 100 times and the slope within its interval 92% of the time.

3.4 Hypothesis Tests

Hypothesis testing allows us to confront any prior notions we may have about the model with what we actually observe. Thus, if before drawing a sample, I believe that autonomous weekly food expenditure is no less than $40, then once the sample is drawn I can determine via a hypothesis test whether experience is actually consistent with this belief.

In section 3.4 of your textbook the authors test several hypotheses about β_2. In 3.4.1a the null hypothesis is that $\beta_2 = 0$ against the alternative that it is positive (i.e., $\beta_2 > 0$). The test statistic is:

$$t = (b_2 - 0)/se(b_2) \sim t_{38} \tag{3.2}$$

provided that $\beta_2 = 0$ (the null hypothesis is true). Select $\alpha = 0.05$ which makes the critical value for the one sided alternative ($\beta_2 > 0$) equal to 1.686. The decision rule is to reject H_0 in favor of the alternative if the computed value of your t-statistic falls within the rejection region of your test; that is if it is larger than 1.686.

The information you need to compute t is contained in the least squares estimation results produced by **gretl**:

Model 1: OLS, using observations 1–40
Dependent variable: food_exp

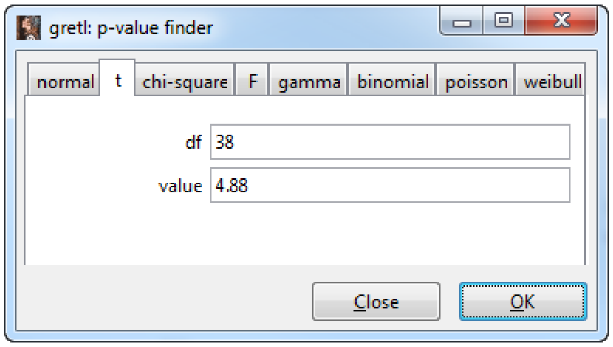
	Coefficient	Std. Error	t-ratio	p-value
const	83.4160	43.4102	1.9216	0.0622
income	10.2096	2.09326	4.8774	0.0000

Mean dependent var	283.5735	S.D. dependent var	112.6752
Sum squared resid	304505.2	S.E. of regression	89.51700
R^2	0.385002	Adjusted R^2	0.368818
$F(1, 38)$	23.78884	P-value(F)	0.000019
Log-likelihood	−235.5088	Akaike criterion	475.0176
Schwarz criterion	478.3954	Hannan–Quinn	476.2389

The computations

$$t = (b_2 - 0)/se(b_2) = (10.21 - 0)/2.09 = 4.88 \tag{3.3}$$

Since this value falls within the rejection region, then there is enough evidence at the 5% level of significance to convince us that the null hypothesis is incorrect; the null hypothesis rejected at this level of significance.


gretl: p-value finder

df 38
value 4.88

Figure 3.7: The dialog box for obtaining p-values using the built in statistical tables in **gretl**.

We can use **gretl** to get the p-value for this test using the `Tools` pull-down menu. In this dialog, you have to fill in the degrees of freedom for your t-distribution (38), the value of b_2 (10.21), its value under the null hypothesis–something **gretl** refers to as 'mean' (0), and the estimated standard error from your printout (2.09). This will yield the information:

```
t(38): area to the right of 4.88 = 9.65032e-006
(two-tailed value = 1.93006e-005; complement = 0.999981)
```

This indicates that the area in one tail is almost zero. The p-value is well below the usual level of significance, $\alpha = .05$, and the hypothesis is rejected.

Gretl also includes a programming command that will compute p-values from several distributions. The `pvalue` function works similarly to the `critical` function discussed in the preceding section. The syntax is:

```
scalar p = pvalue(distribution, parameters, xval)
```

The `pvalue` function computes the area to the right of `xval` in the specified `distribution`. Choices include z for Gaussian, t for Student's t, X for chi-square, F for F, G for gamma, B for binomial, P for Poisson, W for Weibull, or E for generalized error. The argument `parameters` refers to the distribution's **known** parameters, as in its degrees of freedom. So, for this example try

```
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
scalar t2 = ($coeff(income)-0)/$stderr(income)
scalar p2 = pvalue(t,$df,t2)
```

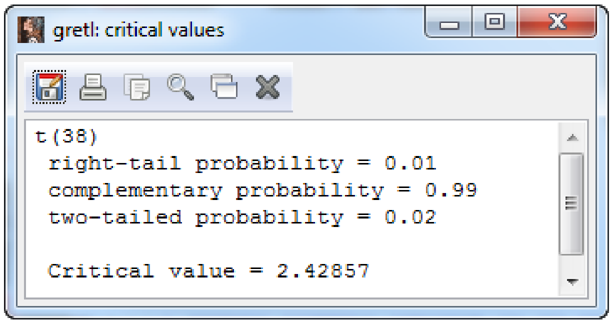
The result is 9.72931e-006, which is very close to the value produced by the dialog box. This values differ because the value in the dialog box was rounded to 4.88 whereas the computed value here has many more significant digits to use in the computation.

In the next example, the authors of *POE4* test the hypothesis that $\beta_2 = 5.5$ against the alternative that $\beta_2 > 5.5$. The computations

$$t = (b_2 - 5.5)/se(b_2) = (10.21 - 5.5)/2.09 = 2.25 \tag{3.4}$$

The significance level in this case is chosen to be 0.01 and the corresponding critical value can be found using a tool found in **gretl**. The `Tools>Statistical tables` pull-down menu bring up the dialog found in Figure 3.1.

This result is found in Figure 3.8. The 0.01 one-sided critical value is 2.42857. Since 2.25 is less



```
t(38)
 right-tail probability = 0.01
 complementary probability = 0.99
 two-tailed probability = 0.02

 Critical value = 2.42857
```

Figure 3.8: The results from the dialog box for obtaining critical values using the built in statistical tables in **gretl**.

than this, we cannot reject the null hypothesis at the 1% level of significance.

In section 3.4.2 of *POE4*, the authors conduct a one-sided test where the rejection region falls within the left tail of the t-distribution. The null hypothesis is $\beta_2 = 15$ and the alternative is $\beta_2 < 15$. The test statistic and distribution is

$$t = (b_2 - 15)/se(b_2) \sim t_{38} \tag{3.5}$$

provided that $\beta_2 = 15$. The computation is

$$t = (b_2 - 15)/se(b_2) = (10.21 - 15)/2.09 = -2.29 \tag{3.6}$$

Based on the desired level of significance, $\alpha = 0.05$, we would reject the null in favor of the one-sided alternative if $t < -1.686$. It is and therefore we conclude that the coefficient is less than 15 at this level of significance.

In section 3.4.3 of *POE4* examples of two-tailed tests are found. In the first example the economic hypothesis that households will spend \$7.50 of each additional \$100 of income on food. So, $H_0 : \beta_2 = 7.50$ and the alternative is $H_1 : \beta_2 \neq 7.50$. The statistic is $t = (b_2 - 7.5)/se(b_2) \sim t_{38}$ if H_0 is true which is computed $t = (b_2 - 7.5)/se(b_2) = (10.21 - 7.5)/2.09 = 1.29$. The two-sided, $\alpha = 0.05$ critical value is 2.024. This means that you reject H_0 if either $t < -2.024$ or if $t > 2.024$. The computed statistic is neither, and hence we do not reject the hypothesis that β_2 is \$7.50. There simply isn't enough information in the sample to convince us otherwise.

You can draw the same conclusions from using a confidence interval that you can from this two-sided t-test. The $100(1 - \alpha)\%$ confidence interval for β_2 is

$$b_2 - t_c se(b_2) \leq \beta_2 \leq b_2 + t_c se(b_2) \tag{3.7}$$

In terms of the example the compute interval is

$$10.21 - 2.024(2.09) \leq \beta_2 \leq 10.21 + 2.024(2.09) \tag{3.8}$$

which as we saw earlier in the manual is $5.97 \leq \beta_2 \leq 14.45$. From a hypothesis testing standpoint, you would not be able to reject the hypothesis that β_2 is different from 7.5 at the 5% level of significance because 7.5 falls within this interval.

In the next example a test of the overall significance of β_2 is conducted. As a matter of routine, you always want to test to see if your slope parameter is different from zero. If not, then the variable associated with it may not belong in your model. So, $H_0 : \beta_2 = 0$ and the alternative is $H_1 : \beta_2 \neq 0$. The statistic is $t = (b_2 - 0)/se(b_2) \sim t_{38}$, if H_0 is true, and this is computed $t = (b_2 - 0)/se(b_2) = (10.21 - 0)/2.09 = 4.88$. Once again, the two-sided, $\alpha = 0.05$ critical value is 2.024 and 4.88 falls squarely within the 5% rejection region of this test. These numbers should look familiar since this is the test that is conducted by default whenever you run a regression in **gretl**.

As we saw earlier, **gretl** also makes obtaining one- or two-sided p-values for the test statistics you compute very easy. Simply use p-value finder dialog box available from the **Tools** pull-down menu (see Figure 3.8) to obtain one or two sided p-values.

3.5 Script for t-values and p-values

One thing we've shown in this chapter is that many of the results obtained using the pull-down menus (often referred to as the GUI) in **gretl** can be obtained using **hansl** from the console or in a script. In fact, the **gretl**'s GUI is merely a front-end to its programming language.[3] In this chapter we used the `pvalue` and `critical` functions to get p-values or critical values of statistics. The following script accumulates what we've covered and completes the examples in the text.

```
open "@gretldir\data\poe\food.gdt"
ols food_exp const income

#One sided test (Ha:  b2 > zero)
scalar tratio1 = ($coeff(income) - 0)/ $stderr(income)
scalar c1 = critical(t,$df,.05)
scalar p1 = pvalue(t,$df,tratio1)
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio1, c1,p1

#One sided test (Ha:  b2>5.5)
scalar tratio2 = ($coeff(income) - 5.5)/ $stderr(income)
scalar c2 = critical(t,$df,.05)
scalar p2 = pvalue(t,$df,tratio2)
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio2, c2,p2

#One sided test (Ha:  b2<15)
scalar tratio3 = ($coeff(income) - 15)/ $stderr(income)
scalar c3 = -1*critical(t,$df,.05)
scalar p3 = pvalue(t,$df,abs(tratio3))
printf "The statistic = %.4f, 5%% critical value = %.4f and\
```

[3]This is true in Stata as well.

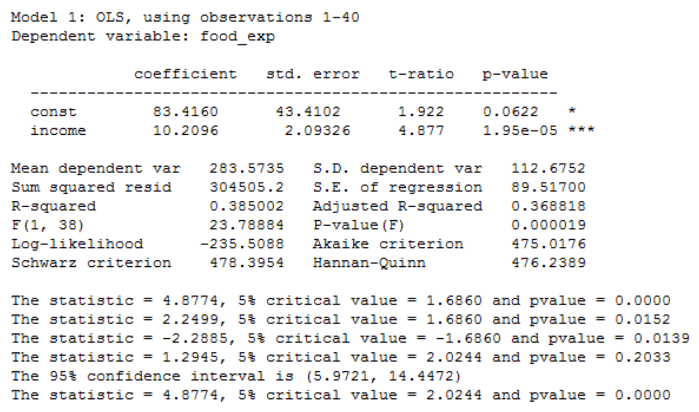
```
23 pvalue = %.4f\n",tratio3, c3,p3
24
25 #Two sided test (Ha:  b2 not equal 7.5)
26 scalar tratio4 = ($coeff(income) - 7.5)/ $stderr(income)
27 scalar c4 = critical(t,$df,.025)
28 scalar p4 = 2*pvalue(t,$df,abs(tratio4))
29 printf "The statistic = %.4f, 5%% critical value = %.4f and\
30 pvalue = %.4f\n",tratio4, c4,p4
31
32 #Confidence interval
33 scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
34 scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
35 printf "The 95%% confidence interval is (%.4f, %.4f)\n",lb,ub
36
37 #Two sided test (Ha:  b2 not equal zero)
38 scalar tratio5 = ($coeff(income) - 0)/ $stderr(income)
39 scalar c5 = critical(t,$df,.025)
40 scalar p5 = 2*pvalue(t,$df,abs(tratio5))
41 printf "The statistic = %.4f, 5%% critical value = %.4f and\
42 pvalue = %.4f\n",tratio5, c5,p5
```

The `pvalue` function in **gretl** measures the area of the probability distribution that lies to the right of the computed statistic. If the computed t-ratio is positive and your alternative is two-sided, multiply the result by 2 to measure the area to the left of its negative; this can be seen in lines 28 and 40. The other function used here is `printf`. This function is a fancy way of printing your results to the screen and its use is explained in detail in section 5.2.2. Because the lines are long, the continuation command (\) discussed in chapter 1 was also used. This tells **gretl** that the current line continues to the next.

If the t-ratio is negative, **gretl** won't compute the area (and you wouldn't want it to, anyway). This is what happened for `tratio3` in the script and I used the absolute value function, `abs( )`, in line 21 to get its positive value. The area to the right of the positive value is equivalent to the area left of the negative value. Hence, the computation is correct.

Basically, proper use of the `pvalue` in one-sided tests of a single hypothesis requires a little thought. Too much thought, in my opinion. I would avoid it unless you are comfortable with its use. In other hypothesis testing contexts (e.g., χ^2 and F-tests) p-values are much easier to use correctly. I use them freely in those cases. With t-tests or z-tests (normal distribution), it is just easier conduct a test by comparing the computed value of your statistic to the correct critical value.

The output from the script is nice and neat, thanks to the use of `printf` and the use of `set echo off`. This appears in Figure 3.9 below. The `set echo off` command used at the beginning of the chapter ending scripts reduces what is printed to the screen when the script is executed. Ordinarily, **gretl** will write (echo) each command executed back to the screen before it produces the requested output. This is useful in most cases, but when running a longer script, it is bothersome. The `set echo off` turns the default echoing of commands off. To turn it back on, use `set echo on`.



```
Model 1: OLS, using observations 1-40
Dependent variable: food_exp

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const        83.4160      43.4102      1.922    0.0622   *
  income       10.2096       2.09326     4.877    1.95e-05 ***

Mean dependent var   283.5735   S.D. dependent var   112.6752
Sum squared resid    304505.2   S.E. of regression   89.51700
R-squared            0.385002   Adjusted R-squared   0.368818
F(1, 38)             23.78884   P-value(F)           0.000019
Log-likelihood      -235.5088   Akaike criterion     475.0176
Schwarz criterion    478.3954   Hannan-Quinn         476.2389

The statistic = 4.8774, 5% critical value = 1.6860 and pvalue = 0.0000
The statistic = 2.2499, 5% critical value = 1.6860 and pvalue = 0.0152
The statistic = -2.2885, 5% critical value = -1.6860 and pvalue = 0.0139
The statistic = 1.2945, 5% critical value = 2.0244 and pvalue = 0.2033
The 95% confidence interval is (5.9721, 14.4472)
The statistic = 4.8774, 5% critical value = 2.0244 and pvalue = 0.0000
```

Figure 3.9: The results produced by the script to test hypotheses in the simple regression.

3.6 Linear Combination of Parameters

Since **gretl** stores and gives access to the estimated values of the coefficients and the variance-covariance matrix, testing hypotheses about linear combinations of parameters is very simple. Suppose you want an estimate of the average weekly food expenditure for a family earning \$2000 per week. The average for any level of income is modeled using linear regression:

$$E(food\_exp|income) = \beta_1 + \beta_2 income \tag{3.9}$$

It can easily be shown that $E(c_1X + c_2Y + c_3) = c_1E(X) + c_2E(Y) + c_3$ where c_1, c_2, and c_3 are constants. If least squares is unbiased for the intercept and slope then $E(b_1) = \beta_1$ and $E(b_2) = \beta_2$. Hence, an estimate of the food expenditure for a family earning \$2000 per week is

$$\widehat{food\_exp} = b_1 + b_2 20 = 83.416 + 10.2096 \times 20 = 287.6089 \tag{3.10}$$

The hypothesis that the average is statistically greater than \$250 can be formally tested as:

$$H_0 : \beta_1 + \beta_2 \leq 0 \quad H_1 : \beta_1 + 20\beta_2 > 250 \tag{3.11}$$

The statistic

$$t = \frac{b_1 + 20b_2 - 250}{se(b_1 + 20b_2 - 250)} \sim t_{N-2} \text{ under } H_0 \tag{3.12}$$

Taking the variance of a linear combination is only slightly more complicated than finding the mean since in the variance calculation any covariance between X and Y needs to be accounted for. In general, $var(c_1X + c_2Y + c_3) = c_1^2 var(X) + c_2^2 var(Y) + 2c_1c_2 cov(X, Y)$. Notice that adding a constant to a linear combination of random variables has no effect on its variance–only its mean. For a regression model, the elements needed to make this computation are found in the variance-covariance matrix.

The precision of least squares (and other estimators) is summarized by the **variance-covariance matrix**, which includes a measurement of the variance of the intercept and the slope, and covariance between the two. The variances of the least squares estimator fall on the diagonal of this square matrix and the covariance is on the off-diagonal.

$$cov(b_1, b_2) = \begin{bmatrix} var(b_1) & cov(b_1, b_2) \\ cov(b_1, b_2) & var(b_2) \end{bmatrix} \tag{3.13}$$

All of these elements have to be estimated from the data. To print an estimate of the variance-covariance matrix following a regression use the `--vcv` option with your regression in **gretl**:

```
ols food_exp const income --vcv
```

In terms of the hypothesis, $var(b_1 + 20b_2 - 250) = 1^2 var(b_1) + 20^2 var(b_2) + 2(1)(20) cov(b_1, b_2)$. The covariance matrix printed by this option is:

```
Covariance matrix of regression coefficients:

         const        income
       1884.44      -85.9032   const
                     4.38175   income
```

The arithmetic for variance is $var(b_1 + 20b_2 - 250) = 1884.44 + (400)(4.38175) + (40)(-85.9032) = 201.017$. The square root of this is the standard error, i.e., 14.178.

Of course, once you know the estimated standard error, you could just as well estimate an interval for the average food expenditure. The script to do just that is found below. Using **hansl** to do the arithmetic makes things a lot easier.

```
scalar vc = $vcv[1,1]+20^2*$vcv[2,2]+2*20*$vcv[2,1]
scalar se = sqrt(vc)
scalar tval = ($coeff(const)+20*$coeff(income)-250)/se
scalar p = pvalue(t,$df,tval)

scalar avg_food_20 = $coeff(const)+20*$coeff(income)
scalar lb = avg_food_20-critical(t,$df,0.025)*se
scalar ub = avg_food_20+critical(t,$df,0.025)*se

print vc se tval p avg_food_20 lb ub
```

In the first line, the accessor `$vcv` is used. In it is the variance-covariance from the previously estimated model. (The square brackets contain the row and column location of the desired element. That is, the estimated variance of b_1 is the element located in the first row and first column, hence `$vcv[1,1]`. The covariance between b_1 and b_2 can be found either in the first row, second column or the second row, first column. So, `$vcv[1,2]=$vcv[2,1]`. The script also produces the p-value associated with a 5% one sided test.

In line 6 the average food expenditure is computed at $income = 20$, which corresponds to \$2000/week (income is measured in \$100). The lower and upper 95% confidence intervals are computed in lines 7 and 8.

```
? print vc se tval p avg_food_20 lb ub

             vc =  201.01688
             se =  14.178042
           tval =  2.6526132
              p =  0.0057953880
    avg_food_20 =  287.60886
             lb =  258.90692
             ub =  316.31081
```

You can see that the manual calculation and that from **gretl** are the same. The p-value is less than 0.05 and we would reject H_0 in favor of the alternative in this case. The average food expenditure for a family earning \$2000/week is \$287. The 95% confidence interval for the average is (\$258.907, \$316.311).

3.7 Script

```
set echo off
# confidence intervals
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
scalar lb = $coeff(income) - 2.024 * $stderr(income)
scalar ub = $coeff(income) + 2.024 * $stderr(income)
print lb ub

# using the critical function to get critical values
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
print lb ub

# t-ratio
open "@gretldir\data\poe\food.gdt"
ols food_exp const income

#One sided test (Ha:  b2 > zero)
scalar tratio1 = ($coeff(income) - 0)/ $stderr(income)
scalar c1 = critical(t,$df,.05)
scalar p1 = pvalue(t,$df,tratio1)
```

```
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio1, c1,p1

#One sided test (Ha:  b2>5.5)
scalar tratio2 = ($coeff(income) - 5.5)/ $stderr(income)
scalar c2 = critical(t,$df,.05)
scalar p2 = pvalue(t,$df,tratio2)
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio2, c2,p2

#One sided test (Ha:  b2<15)
scalar tratio3 = ($coeff(income) - 15)/ $stderr(income)
scalar c3 = -1*critical(t,$df,.05)
scalar p3 = pvalue(t,$df,abs(tratio3))
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio3, c3,p3

#Two sided test (Ha:  b2 not equal 7.5)
scalar tratio4 = ($coeff(income) - 7.5)/ $stderr(income)
scalar c4 = critical(t,$df,.025)
scalar p4 = 2*pvalue(t,$df,tratio4)
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio4, c4,p4

#Confidence interval
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
printf "The 95%% confidence interval is (%.4f, %.4f)\n",lb,ub

#Two sided test (Ha:  b2 not equal zero)
scalar tratio5 = ($coeff(income) - 0)/ $stderr(income)
scalar c5 = critical(t,$df,.025)
scalar p5 = 2*pvalue(t,$df,tratio5)
printf "The statistic = %.4f, 5%% critical value = %.4f and\
pvalue = %.4f\n",tratio5, c5,p5

# linear combinations of coefficients
open "@gretldir\data\poe\food.gdt"
ols food_exp const income --vcv
scalar vc = $vcv[1,1]+20^2*$vcv[2,2]+2*20*$vcv[2,1]
scalar se = sqrt(vc)
scalar tval = ($coeff(const)+20*$coeff(income)-250)/se
scalar p = pvalue(t,$df,tval)

scalar avg_food_20 = $coeff(const)+20*$coeff(income)
scalar lb = avg_food_20-critical(t,$df,0.025)*se
scalar ub = avg_food_20+critical(t,$df,0.025)*se

print vc se tval p avg_food_20 lb ub
```

And for the repeated sampling exercise, the script is:

```
set echo off
open "@gretldir\data\poe\table2_2.gdt"
list ylist = y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
loop foreach i ylist --progressive --quiet
  ols ylist.$i const x
  scalar b1 = $coeff(const)
  scalar b2 = $coeff(x)
  scalar s1 = $stderr(const)
  scalar s2 = $stderr(x)

# 2.024 is the .025 critical value from the t(38) distribution
  scalar c1L = b1 - critical(t,$df,.025)*s1
  scalar c1R = b1 + critical(t,$df,.025)*s1
  scalar c2L = b2 - critical(t,$df,.025)*s2
  scalar c2R = b2 + critical(t,$df,.025)*s2

  scalar sigma2 = $sigma^2
  store @workdir\coeff.gdt b1 b2 s1 s2 c1L c1R c2L c2R sigma2
endloop

open @workdir\coeff.gdt
print c1L c1R c2L c2R --byobs
```

Monte Carlo to measure coverage probabilities of confidence intervals

```
set echo off
open "@gretldir\data\poe\food.gdt"
set seed 3213798
loop 100 --progressive --quiet
    series u = normal(0,88)
    series y = 80 + 10*income + u
    ols y const income
    # 2.024 is the .025 critical value from the t(38) distribution
    scalar c1L = $coeff(const) - critical(t,$df,.025)*$stderr(const)
    scalar c1R = $coeff(const) + critical(t,$df,.025)*$stderr(const)
    scalar c2L = $coeff(income) - critical(t,$df,.025)*$stderr(income)
    scalar c2R = $coeff(income) + critical(t,$df,.025)*$stderr(income)

    # Compute the coverage probabilities of the Confidence Intervals
    scalar p1 = (80>c1L && 80<c1R)
    scalar p2 = (10>c2L && 10<c2R)

    print p1 p2
    store @workdir\cicoeff.gdt c1L c1R c2L c2R
endloop
```

Chapter 4

Prediction, Goodness-of-Fit, and Modeling Issues

Several extensions of the simple linear regression model are now considered. First, conditional predictions are generated using results saved by **gretl**. Then, a commonly used measure of the quality of the linear fit provided by the regression is discussed. We then take a brief detour to discuss how **gretl** can be used to provide professional looking output that can be used in your research.

The choice of functional form for a linear regression is important and the RESET test of the adequacy of your choice is examined. Finally, the residuals are tested for normality. Normality of the model's errors is a useful property in that, when it exists, it improves the the performance of least squares and the related tests and confidence intervals we've considered when sample sizes are small (finite).

4.1 Prediction in the Food Expenditure Model

Generating predicted values of food expenditure for a person with a given income is very simple in **gretl**. After estimating the model with least squares, you can use the `genr` or `series` to get predicted values for all the observations or use `scalar` to get a prediction at a specific point. In the example, a household having $income_o = \$2000$ of weekly income is predicted to spend approximately \$287.61 on food. Recalling that income is measured in hundreds of dollars in the data, the **gretl** commands to compute this from the console are:

```
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
```

```
scalar yhat0 = $coeff(const) + $coeff(income)*20
```

This yields $\widehat{food\_exp}_0 = 287.609$. We could have used `genr` rather than `scalar` (or nothing at all before `yhat0`) and the correct result would be computed. Using `scalar` makes it clear to someone else reading the program that you intend this to compute a single number, not a series.

Obtaining the 95% confidence interval is slightly harder in that there are no internal commands in **gretl** that will do this. The information needed is readily available, however. The formula is:

$$\widehat{var}(f) = \hat{\sigma}^2 + \frac{\hat{\sigma}^2}{T} + (income_o - \overline{income})^2 \widehat{var}(b_2) \tag{4.1}$$

In section 2.4 we estimated $\hat{\sigma}^2 = 8013.29$ and $\widehat{var}(b_2) = 4.3818$. The mean value of income is found by highlighting the variable *income* in the main **gretl** window and the selecting `View>Summary Statistics` from the pull-down menu. This yields $\overline{income} = 19.6047$.[1] The t_{38} 5% critical value is 2.0244 and the computation[2]

$$\widehat{var}(f) = 8013.2941 + \frac{8013.2941}{40} + (20 - 19.6047)^2 * 4.3818 = 8214.31 \tag{4.2}$$

Then, the confidence interval is:

$$\widehat{food\_exp}_0 \pm t_c se(f) = 287.6069 \pm 2.0244\sqrt{8214.31} = [104.132, 471.086] \tag{4.3}$$

The complete script to produce the computed results in **gretl** is:

```
ols food_exp const income
scalar yhat0 = $coeff(const) + $coeff(income)*20
scalar f=8013.2941+(8013.2941/40)+4.3818*(20-19.6047)^2
scalar ub=yhat0+2.0244*sqrt(f)
scalar lb=yhat0-2.0244*sqrt(f)
```

At this point, you may be wondering if there is some way to use the internal functions of **gretl** to produce the same result? As we've seen, **gretl** saves many of the results we need internally and these can in turn be called into service in subsequent computations using their accessors.

For instance, the sum of squared errors from the least squares regression can be accessed using `$ess`. The degrees of freedom and number of observations are saved as `$df` and `$nobs`, respectively. Also, you can use an internal **gretl** function to compute $\overline{income}$, `mean(income)`, and the `critical` function discussed in the preceding chapter to get the desired critical value. Hence, the prediction interval can be automated and made more precise by using the following script.

[1]Your result may vary a little depending on how many digits are carried out to the right of the decimal.

[2]You can compute this easily using the **gretl** console by typing in: `scalar f = 8013.2941 + (8013.2941/40) + 4.3818*(20-19.6047)**2`

```
ols food_exp const income
scalar yhat0=$coeff(const)+20*$coeff(income)
scalar sig2 = $ess/$df
scalar f = sig2 + sig2/$nobs + ((20-mean(income))^2)*($stderr(income)^2)
scalar lb = yhat0-critical(t,$df,0.025)*sqrt(f)
scalar ub = yhat0+critical(t,$df,0.025)*sqrt(f)
print yhat0 sig2 f lb ub
```

This produces

```
yhat0 =  287.60886
 sig2 =  8013.2941
    f =  8214.3110
   lb =  104.13228
   ub =  471.08545
```

which are the values we expect.

4.2 Coefficient of Determination

One use of regression analysis is to "explain" variation in dependent variable as a function of the independent variable. A summary statistic that is used for this purpose is the coefficient of determination, also known as R^2.

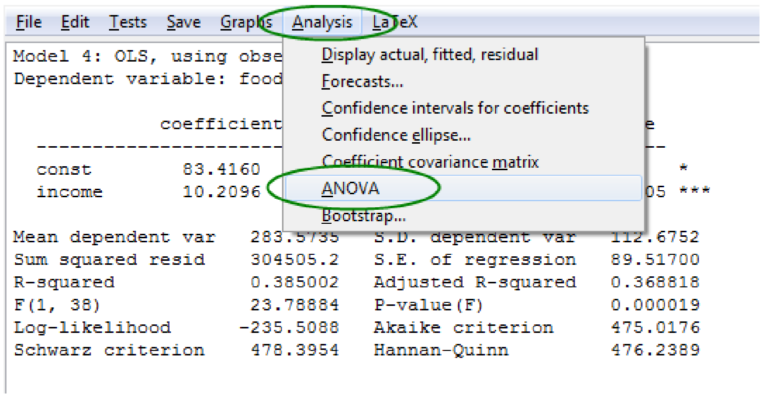
There are a number of different ways of obtaining R^2 in **gretl**. The simplest way to get R^2 is to read it directly off of **gretl**'s regression output. This is shown in Figure 4.3. Another way, and probably the most difficult, is to compute it manually using the **analysis of variance** (ANOVA) table. The ANOVA table can be produced after a regression by choosing `Analysis>ANOVA` from the **model** window's pull-down menu as shown in Figure 4.1. Or, one can simply use the `--anova` option to `ols` to produce the table from the console of as part of a script.

```
ols income const income --anova
```

The result appears in Figure 4.2.

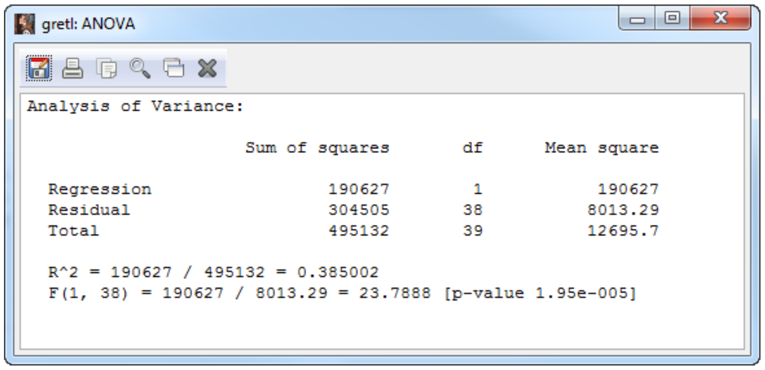
In the ANOVA table featured in Figure 4.2 the *SSR*, *SSE*, and *SST* can be found. **Gretl** also does the R^2 computation for you as shown at the bottom of the output. If you want to verify **gretl**'s computation, then

$$SST = SSR + SSE = 190627 + 304505 = 495132 \tag{4.4}$$


File
Edit
Tests
Save
Graphs
Analysis
LaTeX
Display actual, fitted, residual
Forecasts...
Confidence intervals for coefficients
Confidence ellipse...
Coefficient covariance matrix
ANOVA
Bootstrap...

```
Model 4: OLS, using obse
Dependent variable: food

             coefficient
  -----------------------
  const         83.4160                                    *
  income        10.2096                                  05 ***

Mean dependent var   283.5735   S.D. dependent var   112.6752
Sum squared resid    304505.2   S.E. of regression   89.51700
R-squared            0.385002   Adjusted R-squared   0.368818
F(1, 38)             23.78884   P-value(F)           0.000019
Log-likelihood      -235.5088   Akaike criterion     475.0176
Schwarz criterion    478.3954   Hannan-Quinn         476.2389
```

Figure 4.1: After estimating the regression, select `Analysis>ANOVA` from the model window's pull-down menu.



```
Analysis of Variance:

                    Sum of squares       df      Mean square

  Regression                190627        1           190627
  Residual                  304505       38          8013.29
  Total                     495132       39          12695.7

  R^2 = 190627 / 495132 = 0.385002
  F(1, 38) = 190627 / 8013.29 = 23.7888 [p-value 1.95e-005]
```

Figure 4.2: The ANOVA table

and

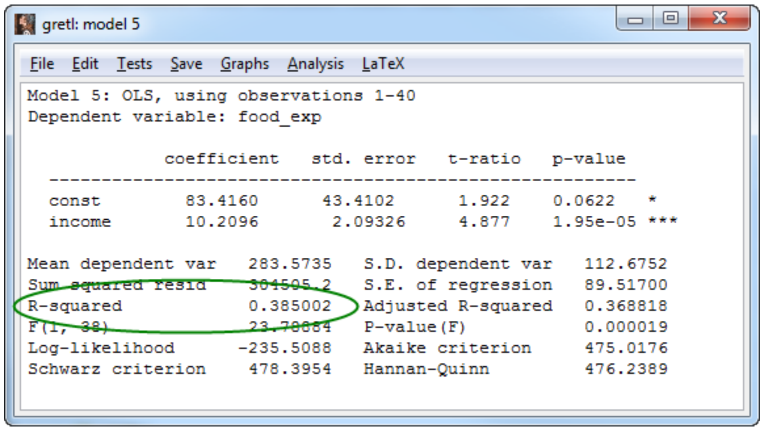
$$\frac{SSR}{SST} = 1 - \frac{SSE}{SST} = \frac{190627}{495132} = .385 \tag{4.5}$$

Different authors refer to regression sum of squares, residual sum of squares and total sum of squares by different acronyms. So, it pays to be careful when computing R^2 manually. *POE4* refers to the regression sum of squares as SSR and the residual sum of squares as SSE (sum of squared errors).

Finally, you can think of R^2 is as the squared correlation between your observations on your dependent variable, *food_exp*, and the predicted values based on your estimated model, $\widehat{food\_exp}$. A **gretl** script to compute this version of the statistic is is found below in section 4.5.4.

To use the GUI you can follow the steps listed here. Estimate the model (equation 2.1) using least squares and add the predicted values from the estimated model, $\widehat{food\_exp}$, to your data set. Then use the **gretl** correlation matrix to obtain the correlation between *food_exp* and $\widehat{food\_exp}$.

Adding the fitted values to the data set from the pull-down menu in the model window is illustrated in Figure 4.4 below. Highlight the variables `food_exp`, `income`, and `yhat1` by holding the control key down and mouse-clicking on each variable in the main **gretl** window as seen in



```
Model 5: OLS, using observations 1-40
Dependent variable: food_exp

             coefficient   std. error   t-ratio   p-value
  ---------------------------------------------------------
  const        83.4160      43.4102      1.922    0.0622   *
  income       10.2096       2.09326     4.877    1.95e-05 ***

Mean dependent var   283.5735   S.D. dependent var   112.6752
Sum squared resid    304505.2   S.E. of regression   89.51700
R-squared            0.385002   Adjusted R-squared   0.368818
F(1, 38)             23.78884   P-value(F)           0.000019
Log-likelihood      -235.5088   Akaike criterion     475.0176
Schwarz criterion    478.3954   Hannan-Quinn         476.2389
```

Figure 4.3: In addition to some other summary statistics, **Gretl** computes the unadjusted R^2 from the linear regression.

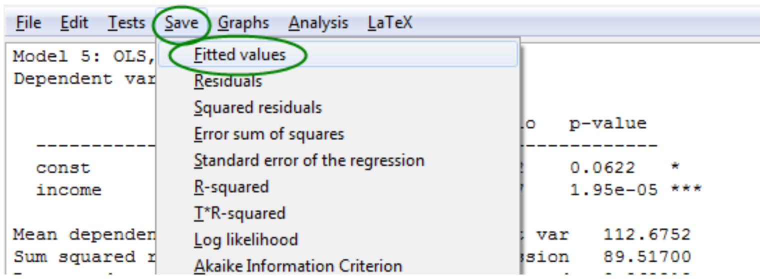

File Edit Tests Save Graphs Analysis LaTeX
Fitted values
Residuals
Squared residuals
Error sum of squares
Standard error of the regression
R-squared
T*R-squared
Log likelihood
Akaike Information Criterion

Figure 4.4: Using the pull-down menu in the Model window to add fitted values to your data set.

Figure 4.5 below. Then, `View>Correlation Matrix` will produce all the pairwise correlations between each variable you've chosen. These are arranged in a matrix as shown in Figure 4.6. Notice that the correlation between *food_exp* and *income* is the same as that between *food_exp* and $\widehat{food\_exp}$ (i.e., 0.6205). As shown in your text, this is no coincidence in the simple linear regression model. Also, squaring this number equals R^2 from your regression, $0.6205^2 = .385$.

You can generate pairwise correlations from the console using

```
c1 = corr(food_exp,$yhat)
```

In yet another example of the ease of using **gretl**, the usual `scalar` or `genr` is not used before `c1`. **Gretl** identifies correctly that the result is a scalar and you can safely omit the command. In longer scripts, however, its generally a good idea to tell **gretl** what you intend to compute and if the result doesn't match you'll get an error message.

4.3 Choosing a Functional Form

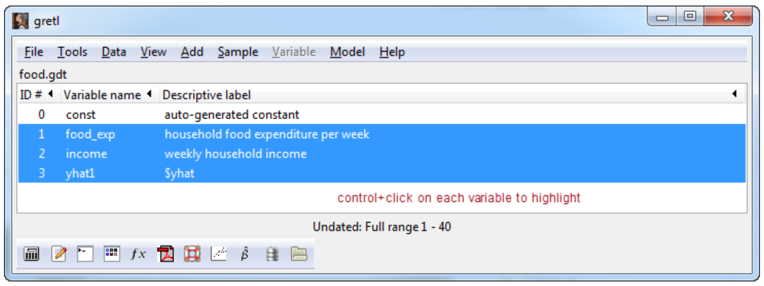
There is no reason to think that the relationship between *food_exp* and *income* is a linear one. In fact, it is likely to be nonlinear. A low wage earner might spend nearly all of an additional dollar on food whereas a high income earner might spend very little. The linear model above implies that rich and poor spend the same amount of an additional dollar of income. As seen in the previous chapters, nonlinearities can be modeled by transforming the dependent or independent variable. This complicates interpretation a bit, but some simple differential calculus can quickly sort things out.

Linear regression is considerably more flexible than its name implies. There are many relationships in economics that are known to be nonlinear. The relationship between production inputs and output is governed in the short-run by the law of diminishing returns, suggesting that a convex curve is a more appropriate function to use. Fortunately, a simple transformation of the variables (x, y, or both) can yield a model that is linear in the parameters (but not necessarily in the variables).

The important point to remember is, the functional form that you choose should be consistent with how the data are actually being generated. If you choose an inappropriate form, then your estimated model may at best not be very useful and at worst be downright misleading.

In **gretl** you are given some very useful commands for transforming variables. From the main **gretl** window the `Add` pull-down menu gives you access to a number of transformations; selecting one of these here will automatically add the transformed variable to your data set as well as its description.

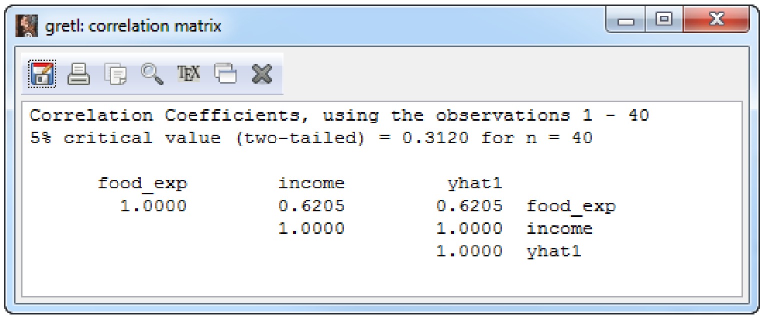
Figure 4.7 shows the available selections from this pull-down menu. In the upper part of the panel two options appear in black, the others are greyed out because they are only available is you have defined the **dataset structure** to consist of time-series observations. The available options can be used to add the natural logarithm or the squared values of any highlighted variable to your data set. If neither of these options suits you, then the next to last option `Define new variable` can be selected. This dialog uses the `scalar` command and the large number of built in functions to transform variables in different ways. Just a few of the possibilities include square roots (`sqrt`),


gretl
File Tools Data View Add Sample Variable Model Help
food.gdt

ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	food_exp	household food expenditure per week
2	income	weekly household income
3	yhat1	$yhat

control+click on each variable to highlight
Undated: Full range 1 - 40

Figure 4.5: Hold the control key and click on *food_exp*, *income*, and $\widehat{food\_exp} = yhat2$ from the food expenditure regression to select them.


gretl: correlation matrix

```
Correlation Coefficients, using the observations 1 - 40
5% critical value (two-tailed) = 0.3120 for n = 40

      food_exp        income          yhat1
        1.0000        0.6205         0.6205  food_exp
                      1.0000         1.0000  income
                                     1.0000  yhat1
```

Figure 4.6: The correlation matrix for *food_exp*, *income*, and $\widehat{food\_exp} = yhat2$ is produced by selecting `View>Correlation matrix` from the pull-down menu.

sine (`sin`), cosine (`cos`), absolute value (`abs`), exponential (`exp`), minimum (`min`), maximum (`max`), and so on. Later in the book, we'll discuss changing the dataset's structure to enable some of the other variable transformation options.

4.3.1 Linear-Log Specification

The linear-log specification of the food expenditure model uses the natural logarithm of income as the independent variable:

$$food\_exp = \beta_1 + \beta_2 \ln(income) + e \tag{4.6}$$

Taking the logarithm of income and estimating the model

```
series l_income = ln(income)
ols food_exp const l_income
```

There is a short-cut that enables you to take the natural logs of several variables at a time. The `logs` function could be use do create ln(*income*) as

```
logs income
```

This command produces a new variable called `l_income` and adds it to the variables list.

Estimation of the model yields

$$\widehat{\text{food\_exp}} = \underset{(84.237)}{-97.1864} + \underset{(28.805)}{132.166}\,\text{l\_income}$$

$$T = 40 \quad \bar{R}^2 = 0.3396 \quad F(1,38) = 21.053 \quad \hat{\sigma} = 91.567$$

(standard errors in parentheses)

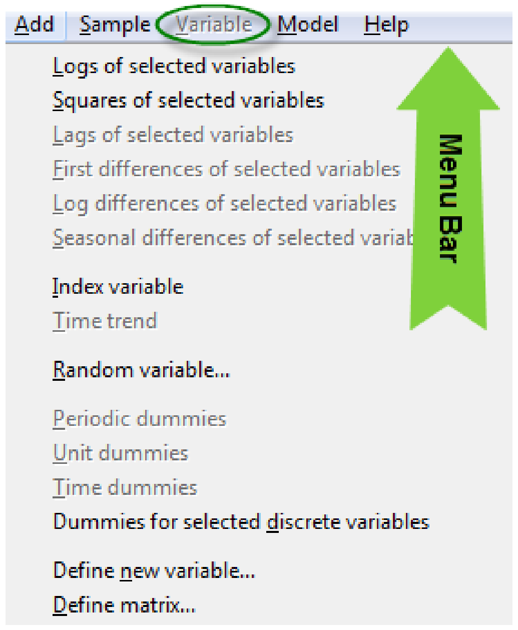

Add
Sample
Variable
Model
Help
Logs of selected variables
Squares of selected variables
Lags of selected variables
First differences of selected variables
Log differences of selected variables
Seasonal differences of selected variab
Index variable
Time trend
Random variable...
Periodic dummies
Unit dummies
Time dummies
Dummies for selected discrete variables
Define new variable...
Define matrix...
Menu Bar

Figure 4.7: The variable pull-down menu is used to add new variables to **gretl**

In Figure 4.6 of *POE4* the authors plot *food_exp* against $\widehat{food\_exp}$. A positive (nonlinear) relationship between the two is expected since the the model was estimated using the natural logarithm of income. To produce this plot, estimate the regression to open the model window. Add the predicted values of from the regression to the dataset using `Save>Fitted values` from the model window's pull-down menu. Name the fitted value, `yhat2` and click **OK**. Now, return to the main window, use the mouse to highlight the three variables (`food_exp`, `yhat2`, and `income`),[3] then select `View>Graph specified vars>X-Y scatter` from the pull-down menu.[4] This opens the **define graph** dialog box. Choose `yhat2` and `food_exp` as the Y-axis variables and `income` as the X-axis variable and click **OK**. A graph appears that looks similar to Figure 4.8

A simpler approach is to open a console or a new script window and use the following commands: To save the predicted values and plot them against the actual observations add

```
ols food_exp const l_income
series yhat2 = $yhat
gnuplot yhat2 food_exp income
```

The first line estimates the regression. The predicted values are held in the accessor, `$yhat`, and are assigned to a new variable called `yhat2` using the `series` command. Then, call **gnuplot** with the predicted values, `yhat2`, as the first variable and the actual values of food expenditure, `food_exp`,

[3]Remember, press and hold Ctrl, then click on each variable

[4]You can also right-click the mouse once the variables are selected to gain access to the scatter plot. If you choose this method, **gretl** will prompt you to specify which of the selected variables is to be used for the X-axis.

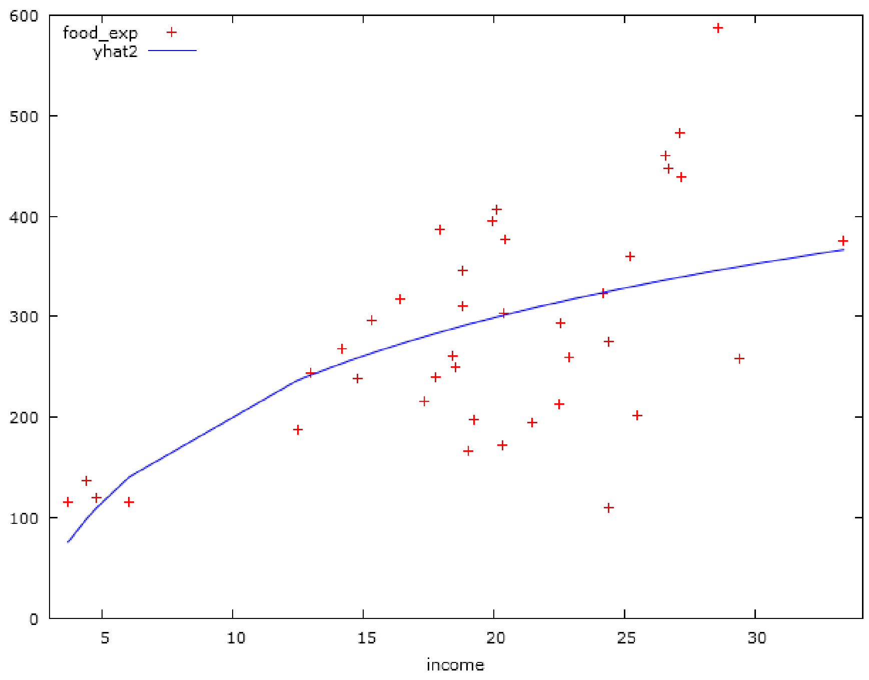


Figure 4.8: Graphing the linear-log model

as the second.

Finally, if you execute these commands using a script, the graph is written to a file on your computer rather than opened in a window. For this reason, I recommend executing these commands from the console rather than from the script file that appears at the end of this chapter.

4.3.2 Residual Plots

Inadvertently choosing an inappropriate functional form can lead to some serious problems when it comes to using your results for decision-making. There are a number of formal tests that one can do to diagnose problems of specification, but researchers often start by looking at residual plots to get a quick idea if there are any problems.

If the assumptions of the classical normal linear regression model hold (ensuring that least squares is minimum variance unbiased) then residuals should look like those found in *ch4sim1.gdt* shown in Figure 4.9 below.

```
open "@gretldir\data\poe\ch4sim1.gdt"
gnuplot e x
```

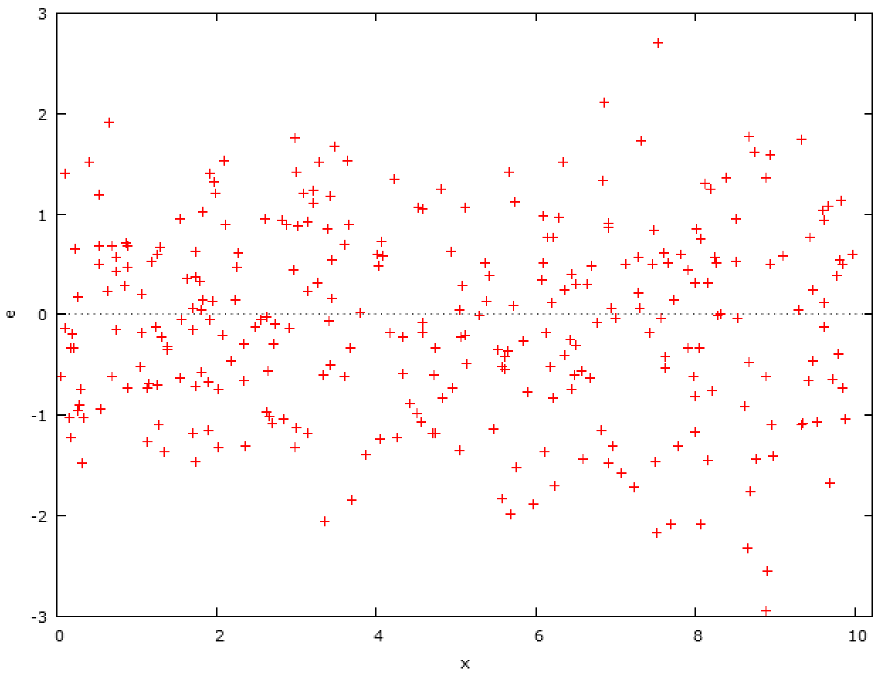


Figure 4.9: Random residuals from *ch4sim1.gdt*

If there is no apparent pattern, then chances are the assumptions required for the Gauss-Markov theorem to hold may be satisfied and the least squares estimator will be efficient among linear estimators and have the usual desirable properties.

The next plot is of the least squares residuals from the linear-log food expenditure model (Figure 4.10). These do not appear to be strictly random. Rather, they are heteroskedastic, which means that for some levels of income, food expenditure varies more than for others (more variance for high incomes). Least squares may be unbiased in this case, but it is not efficient. The validity of hypothesis tests and intervals is affected and some care must be taken to ensure proper statistical inferences are made. This is discussed at more length in chapter 8.

Finally, the *ch4sim2.gdt* dataset contains least squares residuals from a linear regression fit to quadratic data. To treat the relationship as linear would be like trying to fit a line through a parabola! This appears in Figure 4.11. The script to generate this is:

```
1 open "@gretldir\data\poe\ch4sim2.gdt"
2 ols y const x
3 series ehat = $uhat
4 gnuplot ehat x
```

Notice that another accessor has been used to store the residuals into a new variable. The residuals from the preceding regression are stored and can be accessed via `$uhat`. In line 3 these were

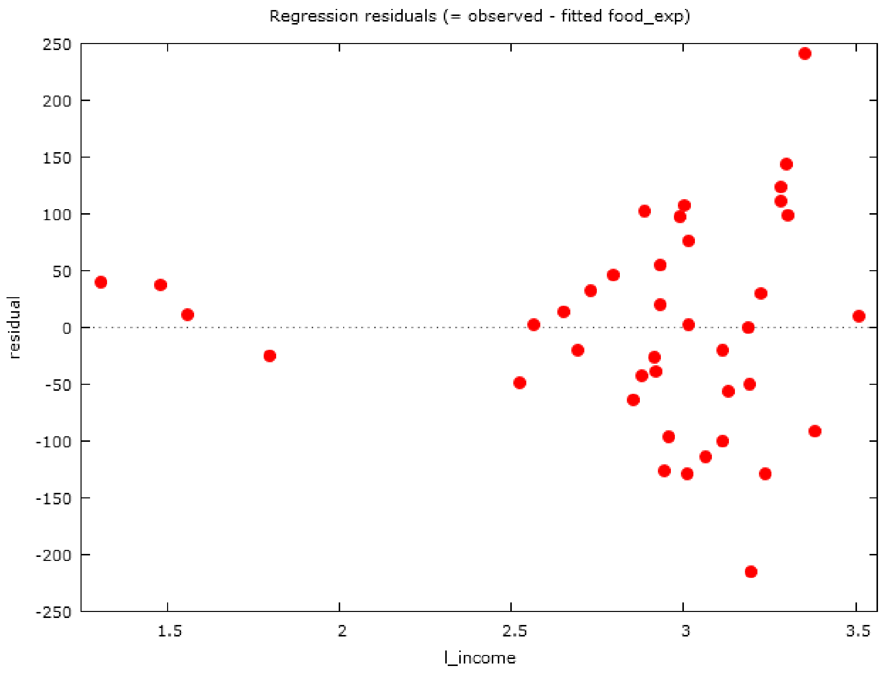


Figure 4.10: Heteroskedastic residuals from the linear-log model of food expenditures.

accessed and assigned to the variable `ehat`. Then, they can be plotted using **gnuplot**.

Looking at the plot in Figure 4.11, there is an obvious problem with model specification. The errors are supposed to look like a random scatter around zero. There are clearly parabolic and the model is NOT correctly specified.

4.3.3 Testing for Normality

Your book, *Principles of Econometrics*, discusses the Jarque-Bera test for normality which is computed using the skewness and kurtosis of the least squares residuals. To compute the Jarque-Bera statistic, you'll first need to estimate your model using least squares and then save the residuals to the data set.

From the **gretl** console

```
ols food_exp const income
series uhat1 = $uhat
summary uhat1
```

The first line is the regression. The next accesses the least squares redsiduals, $uhat, and places

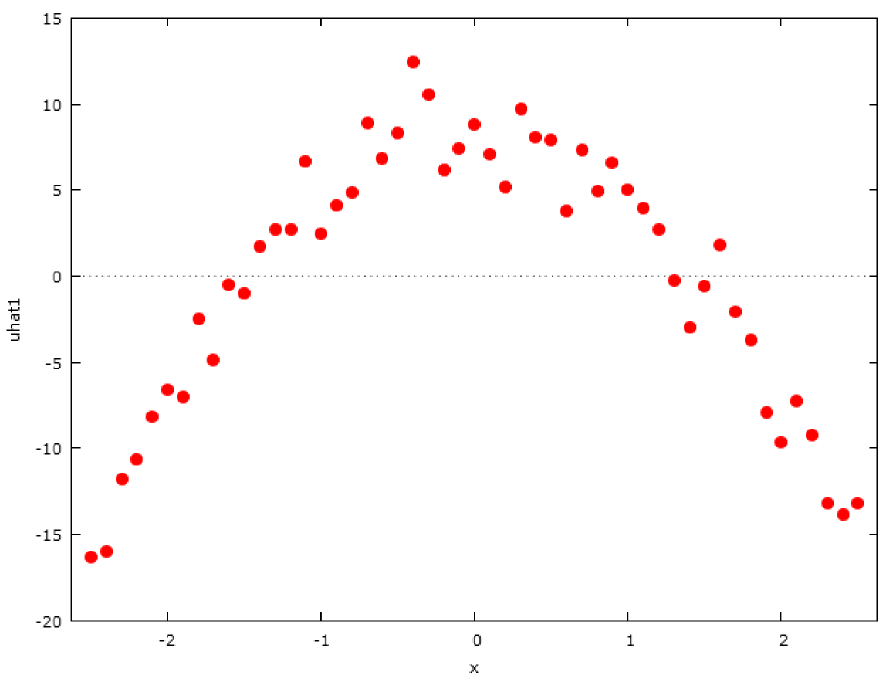


Figure 4.11: Correlated residuals from estimating a quadratic relationship using a line.

them into a new series called `uhat1`.[5] You could also use the point-and-click method to add the residuals to the data set. This is accomplished from the regression's output window. Simply choose `Save>Residuals` from the model pull-down menu to add the estimated residuals to the dataset. The last line of the script produces the summary statistics for the residuals and yields the output in Figure 4.12. One thing to note, **gretl** reports excess kurtosis rather than kurtosis. The excess

```
                 Summary Statistics, using the observations 1 - 40
                  for the variable 'uhat1' (40 valid observations)

       Mean                         0.00000
       Median                      -6.3245
       Minimum                   -223.03
       Maximum                    212.04
       Standard deviation          88.362
       C.V.                         2.4147E+015
       Skewness                    -0.097319
       Ex. kurtosis                -0.010966
```

Figure 4.12: The summary statistics for the least squares residuals.

kurtosis is measured relative to that of the normal distribution which has kurtosis of three. Hence, your computation is

$$JB = \frac{T}{6}\left(\text{Skewness}^2 + \frac{(\text{Excess Kurtosis})^2}{4}\right) \tag{4.7}$$

[5]You can't use `uhat` instead of `uhat1` because that name is reserved by **gretl**.

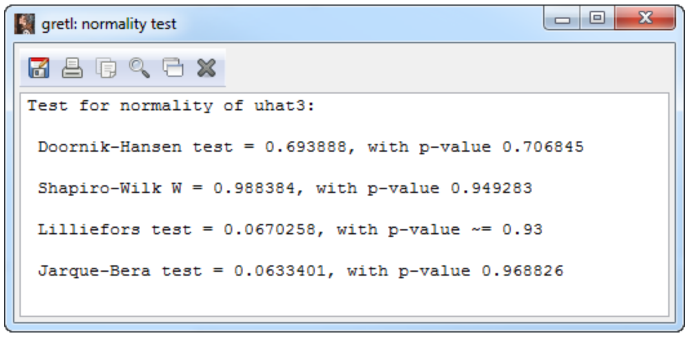
Which is

$$JB = \frac{40}{6}\left(-0.097^2 + \frac{-0.011^2}{4}\right) = .063 \tag{4.8}$$

Normal random variables have no skewness nor any excess kurtosis. The JB statistic is zero in this case. It gets larger the higher the skewness and the greater the degree of excess kurtosis displayed by the data. In section C.1 **hansl** is used to compute skewness and excess kurtosis and you could use these computations to compute your own JB test. Fortunately, there is no need to compute your own because **gretl** will compute the Jarque-Bera test for you. After saving the residuals into `$uhat1` issue the command

```
ols food_exp const income
series uhat1 = $uhat
normtest uhat1 --jbera
normtest uhat1 --all
```

This yields a value of Jarque-Bera test = 0.0633401, with p-value 0.968826, which is exactly what the manual calculation yields. **Gretl** performs other tests for the normality of residuals including one by Doornik and Hansen (2008). Computationally, it is more complex than the Jarque-Bera test. The Doornik-Hansen test has a χ^2 distribution if the null hypothesis of normality is true. It can be produced from `normtest` along with several others using the `--all` option. Output from `normtest --all` is shown in Figure 4.13. Obviously, one of the advantages of using `normtest` is



```
Test for normality of uhat3:

 Doornik-Hansen test = 0.693888, with p-value 0.706845

 Shapiro-Wilk W = 0.988384, with p-value 0.949283

 Lilliefors test = 0.0670258, with p-value ~= 0.93

 Jarque-Bera test = 0.0633401, with p-value 0.968826
```

Figure 4.13: Using `normtest` *residual* `--all` tests the variable *residual* for normality after running a linear regression.

that you can test for the normality of any series, not just residuals.

Another possibility is to use the `modtest` function after estimating a model using least squares.

```
ols food_exp const income
modtest --normality
```

The `modtest` command is actually is a generic function that allows you to test a number of different hypotheses regarding the specification of your model. This function operates on the residuals of the last model estimated. Using it after a regression with the `--normality` option produces the following output

```
Frequency distribution for uhat2, obs 1-40
number of bins = 7, mean = -2.45137e-014, sd = 89.517

       interval          midpt   frequency    rel.     cum.

           < -186.77   -223.03        1      2.50%    2.50%
   -186.77 - -114.26   -150.51        3      7.50%   10.00% **
   -114.26 - -41.747   -78.002        8     20.00%   30.00% *******
   -41.747 -  30.765   -5.4907       14     35.00%   65.00% ************
    30.765 -  103.28    67.021        8     20.00%   85.00% *******
    103.28 -  175.79    139.53        5     12.50%   97.50% ****
          >=  175.79    212.04        1      2.50%  100.00%

Test for null hypothesis of normal distribution:
Chi-square(2) = 0.694 with p-value 0.70684
```

The distribution of the residuals is collected and plotted in a basic graph and the results for the *DH* test are given. If `modtest` is executed from GUI using `Tests>Normality of residuals` in the model results window, a **gnuplot** histogram of the errors is generated with a normal density overlaid. The results of the *DH* test are again printed on the graph.

4.4 Reporting Results

In case you think **gretl** is just a toy, the program includes a very capable utility that enables it to produce professional looking output. LaTeX, usually pronounced "Lay-tek", is a typesetting program used by mathematicians and scientists to produce professional looking technical documents. It is widely used by econometricians to prepare manuscripts for wider distribution. In fact, this book is produced using LaTeX.

Although LaTeX is free and produces very professional looking documents, it is not widely used by undergraduate and masters students because 1) most degree programs don't require you to write a lot of technical papers and 2) it's a computer language and therefore it takes some time to learn its intricacies and to appreciate its nuances. Heck, I've been using it for years and still scratch my head when I try to put tables and Figures in the places I'd like them to be!

In any event, **gretl** includes a facility for producing output that can be pasted directly into LaTeX documents. For users of LaTeX, this makes generating regression output in proper format a breeze. If you don't already use LaTeX, then this will not concern you. On the other hand, if you already use it, **gretl** can be very handy in this respect.

In Figure 4.3 you will notice that on the far right hand side of the menu bar is a pull-down menu for **LaTeX**. From here, you click `LaTeX` on the menu bar and a number of options are revealed as shown in Figure 4.14. You can view, copy, or save the regression output in either tabular form or

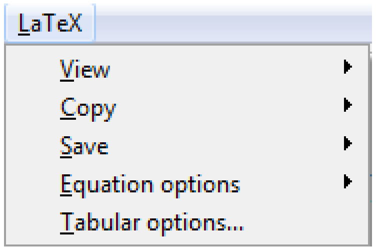


Figure 4.14: Several options for defining the output of LaTeX are available.

in equation form. You can tell **gretl** whether you want standard errors or t-ratios in parentheses below parameter estimates, and you can define the number of decimal places to be used of output. Nice indeed. Examples of tabular and equation forms of output are found in Tables 4.1 and 4.2, respectively.

OLS, using observations 1–40
Dependent variable: food_exp

	Coefficient	Std. Error	t-ratio	p-value
const	83.4160	43.4102	1.9216	0.0622
income	10.2096	2.09326	4.8774	0.0000

Mean dependent var	283.5735	S.D. dependent var	112.6752
Sum squared resid	304505.2	S.E. of regression	89.51700
R^2	0.385002	Adjusted R^2	0.368818
$F(1,38)$	23.78884	P-value(F)	0.000019
Log-likelihood	−235.5088	Akaike criterion	475.0176
Schwarz criterion	478.3954	Hannan–Quinn	476.2389

Table 4.1: This is an example of LaTeX output in tabular form.

$$\widehat{\text{food\_exp}} = \underset{(43.410)}{83.4160} + \underset{(2.0933)}{10.2096}\,\text{income}$$

$$T = 40 \quad \bar{R}^2 = 0.3688 \quad F(1,38) = 23.789 \quad \hat{\sigma} = 89.517$$

(standard errors in parentheses)

Table 4.2: Example of LaTeX output in equation form

4.5 Polynomial Models

Using polynomials to capture nonlinearity in regression is quite easy and often effective. Students of economics are quite used to seeing U-shaped cost curves and S-Shaped production functions and these shapes are simply expressed using quadratic and cubic polynomials, respectively. Since the focus so far has been on simple regression, i.e., regression models with only one independent variable, the discussion in *POE4* is simplified to include only a single squared or cubed value of the independent variable.

The general form of a quadratic equation $y = a_0 + a_1 x + a_2 x^2$ includes a constant, the level of x and its square. The latter two terms are multiplied times coefficients, a_1 and a_2 that determine the actual shape of the parabola. A cubic equation adds a cubed term, $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$. The simple regressions considered in this section include only the constant, a_0 and either the squared term in a quadratic model or the cubed term in the cubic model.

The simple quadratic regression has already been considered. The regression and its slope are

$$\begin{aligned} y &= \beta_1 + \beta_2 x^2 \\ dy/dx &= 2\beta_2 x \end{aligned}$$

From this you can see that the function's slope depends on the parameter β as well as the value of the variable x.

The cubic model and its slope are

$$\begin{aligned} y &= \beta_1 + \beta_2 x^3 \\ dy/dx &= 3\beta_2 x^2 \end{aligned}$$

Since x is squared in the slope, the algebraic sign of β_2 determines whether the slope is positive or negative. Both of these models are considered using examples below.

4.5.1 Wheat Yield

Figure 4.15 plots the average wheat yield in Greenough Shire over time (in tonnes per hectare–we're in OZ!) using the data in *wa_wheat.gdt*. The results from the example in section 4.4 of your textbook is easily produced in **gretl**. Start by loading the data and estimating the effect of time, *time* on yield *greenough* using least squares. The following script will load the data file, estimate the model using least squares, and generate a graph of the actual and fitted values of yield (`greenough`) from the model.

```
open "@gretldir\data\poe\wa-wheat.gdt"
ols greenough const time
gnuplot greenough time
```

The resulting plot appears below in Figure 4.15. The fitted line can be added. Right-clicking on

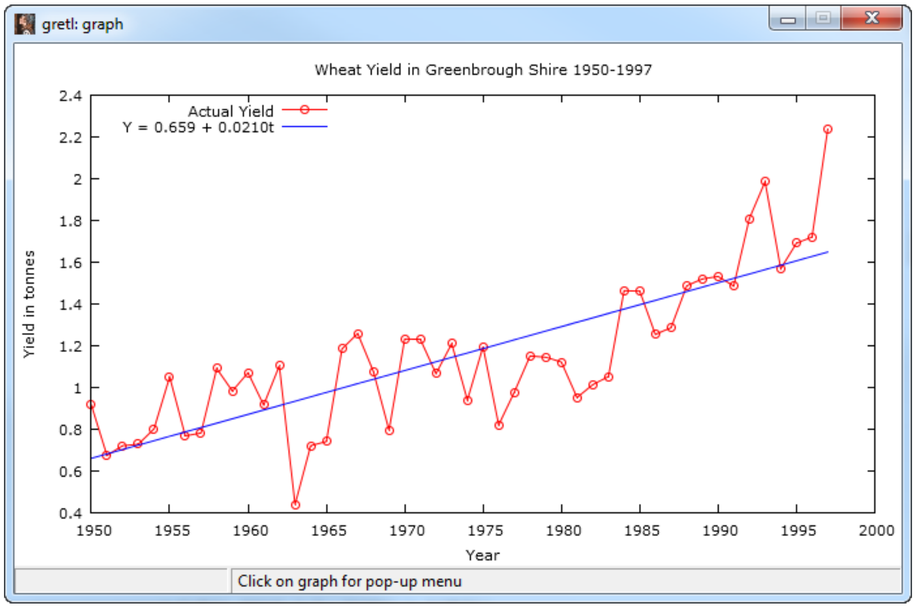


Figure 4.15: The plot of the actual yield in Greenough Shire over time

the graph brings up a menu of choices. Choose Edit and the plot controls dialog box appears as shown in Figure 4.16. There is a pull-down menu in the box called fitted line from which you can choose to fit a line, a quadratic, or a cubic equation. I chose line and the result appears in the figure. From the **lines** tab a few of the defaults; the **legend** for the series is changed to Actual Yield and the line style was changed to line/points. The **X-axis** tab was used to change the axis label to 'Year.'

The simple **gnuplot** command works well enough. However, I took advantage of having declared the dataset structure to be time-series to improve the look. I also added a description and label to be used in the graph using the `-d` and `-n` switches for `setinfo`. The commands are

```
setinfo greenough -d "Wheat yield in tonnes" -n "Yield in tonnes"
gnuplot greenough --with-lines --time-series --linear-fit
```

There are three options listed after the plot. The first (`--with-lines`) tells **gnuplot** to connect the points using lines. The second option (`--time-series`) tells **gnuplot** that the graph is of time-series. In this case, the dataset's defined time variable will be used to locate each point's position on the X-axis. The final option plots the least squares fit of a line. To make the graph look like Figure 4.15 some further manipulation was done using the plot controls.

To explore the behavior of yield further, create a new variable using the `series` command from $t3 = time^3/1,000,000$ as shown below. This rescaling of time cubed merely changes the scale of the coefficient by a corresponding amount and has no effect on the shape or fit of the model. It

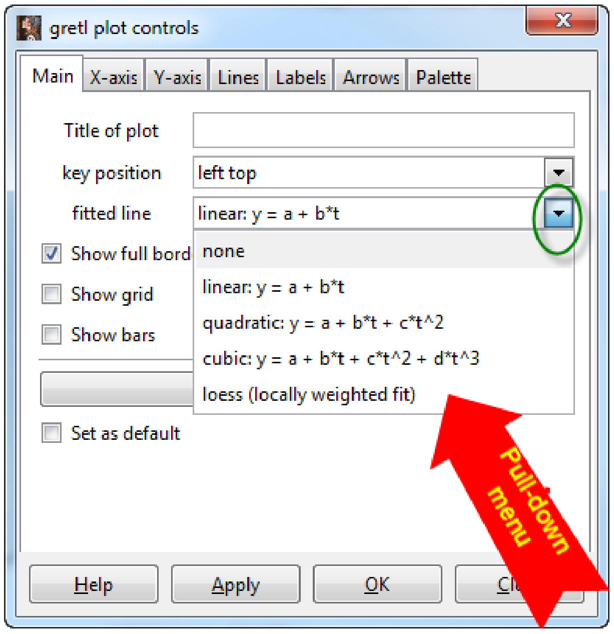


Figure 4.16: The graph dialog box can be used to change characteristics of your graphs. Use the Main tab to give the graph a new name and colors; use the X- and Y-axes tabs to refine the behavior of the axes and to provide better descriptions of the variables graphed.

is particularly useful for long time-series since cubing large integers may exceed your computer's capacity to yield accurate results (i.e., numerical overflow). The new plot appears in Figure 4.17.

```
series t3=time^3/1000000
ols greenough const t3
gnuplot greenough --with-lines --time-series
```

4.5.2 Growth Model

Below you will find a script that reproduces the results from the growth model example in section 4.5.1 of *POE4*. If yield grows at a constant rate of g, then yield at time $t = 1$ will be $yield_1 = yield_0(1 + g)$. For constant growth rates, repeated substitution produces

$$yield_t = yield_0(1 + g)^t \tag{4.9}$$

Taking the natural log

$$\ln(yield_t) = \ln(yield_0) + t\ln(1 + g) = \beta_1 + \beta_2 t \tag{4.10}$$

add an error and you have a regression model. The parameter, $\beta_2 = \ln(1 + g)$. This is an example of a log-linear model where the independent variable is time. The slope coefficient in such a model measures the approximate annual growth rate in the dependent variable.

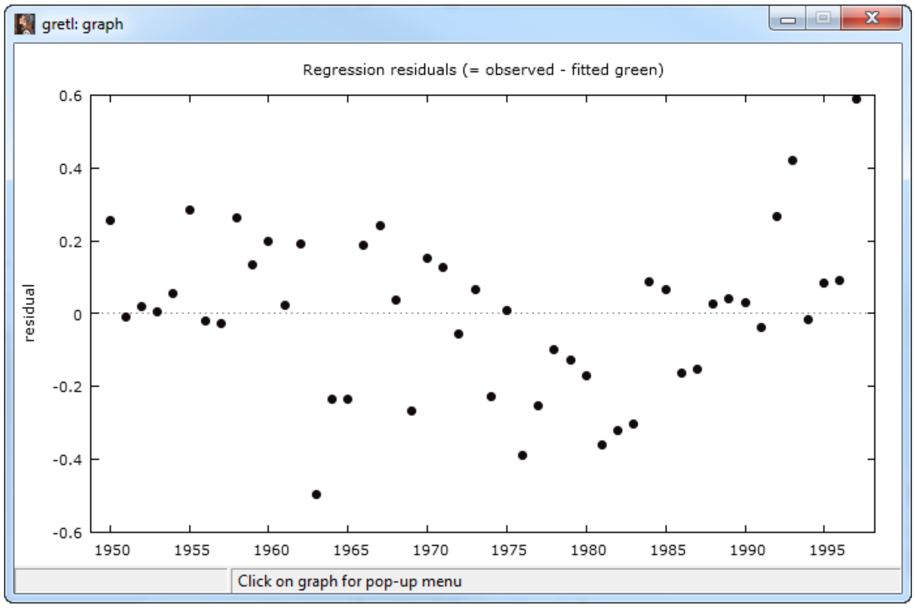


Figure 4.17: The plot of the residuals from a linear model. There is some visual evidence of serial correlation, suggesting that the linear model is misspecified.

```
open "@gretldir\data\poe\wa-wheat.gdt"
series lyield = log(greenough)
ols lyield const time
```

This produces

$$\widehat{\text{l\_greenough}} = \underset{(0.058404)}{-0.343366} + \underset{(0.0020751)}{0.0178439}\,\text{time}$$
$$T = 48 \quad \bar{R}^2 = 0.6082 \quad F(1,46) = 73.945 \quad \hat{\sigma} = 0.19916$$
$$\text{(standard errors in parentheses)}$$

The estimated coefficient $b_2 = \ln(1+g) = 0.0178$. This implies that the growth rate in wheat yield is approximately 1.78% annually over the course of the sample.[6]

4.5.3 Wage Equation

Below you will find a script that reproduces the results from the wage equation example in section 4.5.2 of *POE4*. In this example the log-linear model is used to measure the approximate

[6]For small g, $\ln(1+g) \cong g$.

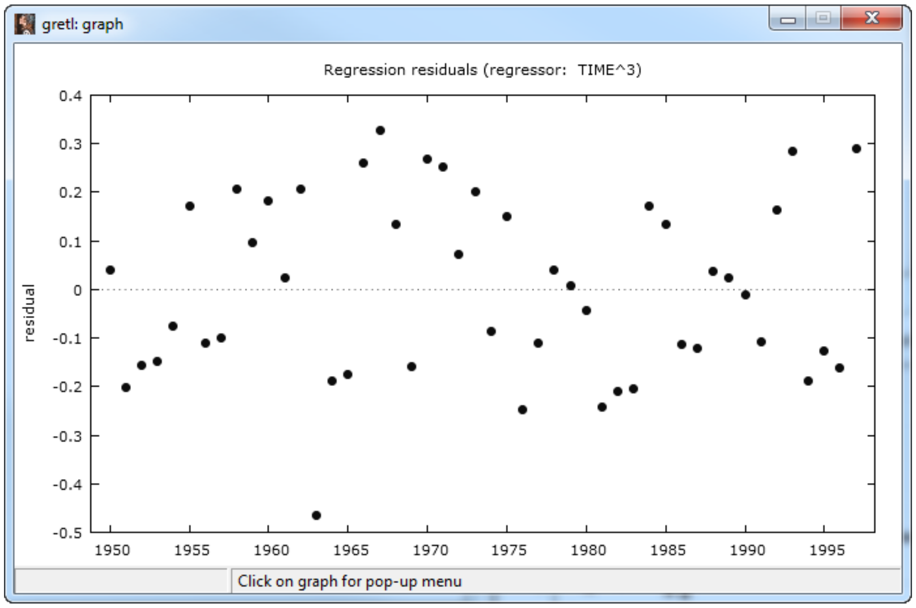


Figure 4.18: The plot of the residuals from a linear model. There is some visual evidence of serial correlation, suggesting that the linear model is misspecified.

return to another year of education. The example uses a thousand observations from the CPS monthly survey from 2008.

```
open "@gretldir\data\poe\cps4_small.gdt"
series l_wage = log(wage)
ols l_wage const educ
scalar lb = $coeff(educ) - 1.96 * $stderr(educ)
scalar ub = $coeff(educ) + 1.96 * $stderr(educ)
print lb ub
```

The regression results are:

$$\widehat{\text{l\_wage}} = \underset{(0.086423)}{1.60944} + \underset{(0.0061456)}{0.0904082}\,\text{educ}$$

$$T = 1000 \quad \bar{R}^2 = 0.1774 \quad F(1,998) = 216.41 \quad \hat{\sigma} = 0.52661$$

(standard errors in parentheses)

and the 95% confidence intervals for the slope is

Variable	Coefficient	95% confidence interval	
educ	0.0904082	0.0783484	0.102468

That is, an additional year of education is worth between 7.8% and 10.2% wage increases annually. Sign me up!

4.5.4 Generalized R^2

A generalized version of the goodness-of-fit statistic R^2 can be obtained by taking the squared correlation between the actual values of the dependent variable and those predicted by the regression. The following script reproduces the results from section 4.4.4 of your textbook.

```
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ
series y = exp($yhat)
scalar corr1 = corr(y, wage)
scalar Rsquare = corr1^2
print corr1 Rsquare
```

This yields an estimated correlation of 0.4312 and a squared correlation of 0.1859.

4.5.5 Predictions in the Log-linear Model

In this example, you use the regression to make predictions about the log wage and the level of the wage for a person having 12 years of schooling. The naive prediction of wage merely takes the antilog of the predicted $\ln(wage)$. This can be improved upon by using properties of log-normal random variables. It can be shown that if $\ln(w) \sim N(\mu, \sigma^2)$ then $E(w) = e^{\mu+\sigma^2/2}$ and $var(w) = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)$.

That means that the corrected prediction is $\hat{y}^c = \exp\left(b_1 + b_2 x + \hat{\sigma}^2/2\right) = e^{(b_1+b_2x)}e^{\hat{\sigma}^2/2}$. The script to generate these is given below.

```
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ
scalar l_wage_12 = $coeff(const)+$coeff(educ)*12
scalar nat_pred = exp(l_wage_12)
scalar corrected_pred = nat_pred*exp($sigma^2/2)
print l_wage_12 nat_pred corrected_pred
```

The results from the script are

```
        l_wage_12 =  2.6943434
         nat_pred =  14.795801
   corrected_pred =  16.996428
```

That means that for a worker with 12 years of schooling the predicted wage is $14.80/hour using the natural predictor and $17.00/hour using the corrected one. In large samples we would expect the corrected predictor to be a bit better. Among the 1000 individuals in the sample, 328 of them have 12 years of schooling. Among those, the average wage is $15.99. Hence the corrected prediction overshoots by about a dollar/hour. Still, it is closer than the uncorrected figure.

To get the average wage for those with 12 years of schooling, we can restrict the sample using the script below:

```
smpl educ=12 --restrict
summary wage
smpl full
```

The syntax is relatively straightforward. The `smpl` command instructs **gretl** that something is being done to the sample. The second statement `educ=12` is a condition that **gretl** looks for within the sample. The `--restrict` option tells **gretl** what to do for those observations that satisfy the condition. The summary wage statement produces

Summary Statistics, using the observations 1–328
for the variable wage (328 valid observations)

Mean	Median	Minimum	Maximum
15.9933	14.2050	2.50000	72.1300
Std. Dev.	**C.V.**	**Skewness**	**Ex. kurtosis**
8.84371	0.552963	2.31394	9.08474

which shows that the mean for the 328 observations is almost $16.00. The last line `smpl full` restores the full sample.

4.5.6 Prediction Interval

To generate a complete confidence interval for every year of schooling between 1 and 21 years, you can use the following script. The result looks very similar to Figure 4.15 in *POE4*.

```
1  open "@gretldir\data\poe\cps4_small.gdt"
2  logs wage
3  ols l_wage const educ
4  scalar sig2 = $ess/$df
5  matrix sem = zeros(21,5)
6  loop for i = 1..21 --quiet
7      scalar yh = ($coeff(const) + $coeff(educ)*i)
8      scalar f = sig2 + sig2/$nobs + ((i-mean(educ))^2)*($stderr(educ)^2)
9      sem[i,1]=i
10     sem[i,2]= yh
11     sem[i,3]=sqrt(f)
12     sem[i,4]=exp(yh-critical(t,$df,0.025)*sqrt(f))
13     sem[i,5]=exp(yh+critical(t,$df,.025)*sqrt(f))
14 endloop
15 print sem
16
17 nulldata 21 --preserve
18 series ed = sem[,1]
19 series wage = exp(sem[,2])
20 series lb = sem[,4]
21 series ub = sem[,5]
```

Although there are probably more elegant ways to do this, this script works. It will take a bit of explanation, however. In lines 1-4 the dataset is opened, log wage is created, the regression is estimated as is the overall variance of the model.

In line 5 a matrix of zeros is created that will be used to store results created in a loop. The loop starts at `i=1` and iterates, by one, to `21`. These are the possible years of schooling that individuals have in our dataset. For each number of years the forecast and its forecast variance are estimated (lines 7 and 8). Notice that these will have different values at each iteration of the loop thanks to their dependence on the index, `i`. In line 9 the matrix `sem` gets `i` placed on the i^{th} row of the first column. The next line puts the prediction in the second column. In the third column I've placed the forecast standard error and in the next two the lower and upper boundaries for the interval. The loop ends at `i=21`, at which point the matrix `sem` is full; then it is printed.

Although you can plot the columns of matrices, I find it easier to put the columns into a dataset and use the regular **gretl** commands to make plots. First, create an empty dataset using `nulldata 21`. The `21` puts 21 observations into the dataset. The `--preserve` option is required because without it the contents of the matrix `sem` would be emptied–definitely not what we want. In the next lines the `series` command is used to put each column of the matrix into a data series. Once this is done, the variables will show up in the data window and you can graph them as usual. Below in Figure 4.19 is the graph that I created (with a little editing).

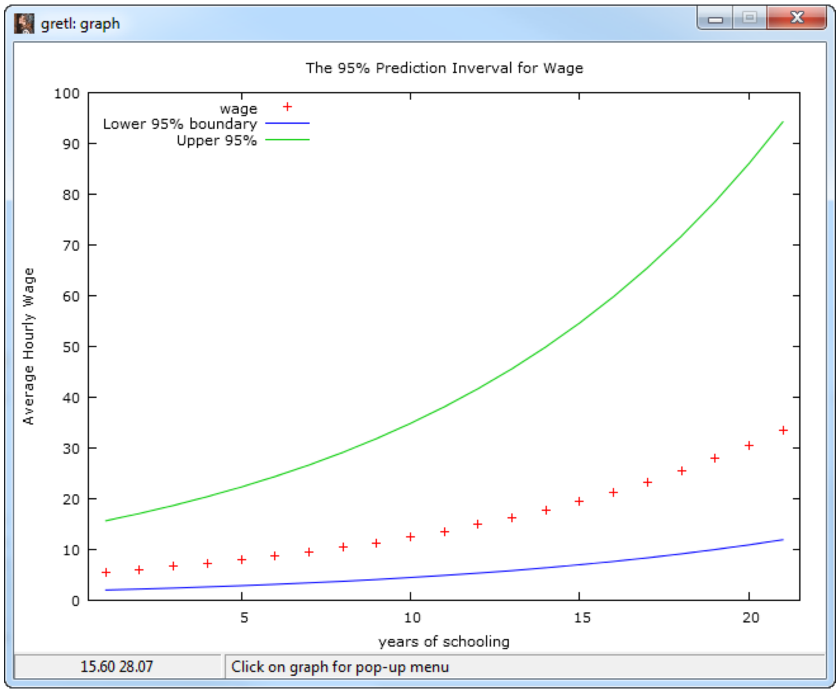


Figure 4.19: This is a plot generated using a loop to estimate forecast standard errors.

4.5.7 Log-Log Model

Finally, a log-log model is used. This functional form is often used to estimate demand equations as it implies a constant price elasticity for the commodity in question. This example uses the newbroiler.gdt which is adapted from Epple and McCallum (2006). The variable Q is per capita consumption of chicken, in pounds and P is the real price in dollars. The sample is from 1950-2001. The estimated log-log model is

$$\widehat{\text{l\_q}} = \underset{(0.022359)}{3.71694} - \underset{(0.048756)}{1.12136}\,\text{l\_p}$$
$$T = 52 \quad \bar{R}^2 = 0.9119 \quad F(1,50) = 528.96 \quad \hat{\sigma} = 0.11799$$
$$\text{(standard errors in parentheses)}$$

The coefficient on logarithm of P is 1.121 which means that a 1% increase in the real price of chicken will decrease quantity demanded by 1.121%.

Once again, the predictor of quantity needs to be corrected since the model is estimated in logarithms. $\hat{Q}^c = \exp\left(b_1 + b_2 \ln(x) + \hat{\sigma}^2/2\right) = e^{\widehat{\ln(Q)}} e^{\hat{\sigma}^2/2}$. The R^2 statistic can be computed as the squared correlation between Q and $\hat{Q}$. The script for this exercise is:

```
open "@gretldir\data\poe\newbroiler.gdt"
logs q p
ols l_q const l_p
series yht=$yhat
series pred = exp(yht)
series corrected_pred=pred*exp($sigma^2/2)
scalar r2= corr(corrected_pred,q)^2
gnuplot corrected_pred q p
```

The results are

```
? scalar r2= corr(corrected_pred,q)^2
Generated scalar r2 = 0.881776
```

and the corresponding graph is found in Figure 4.20.

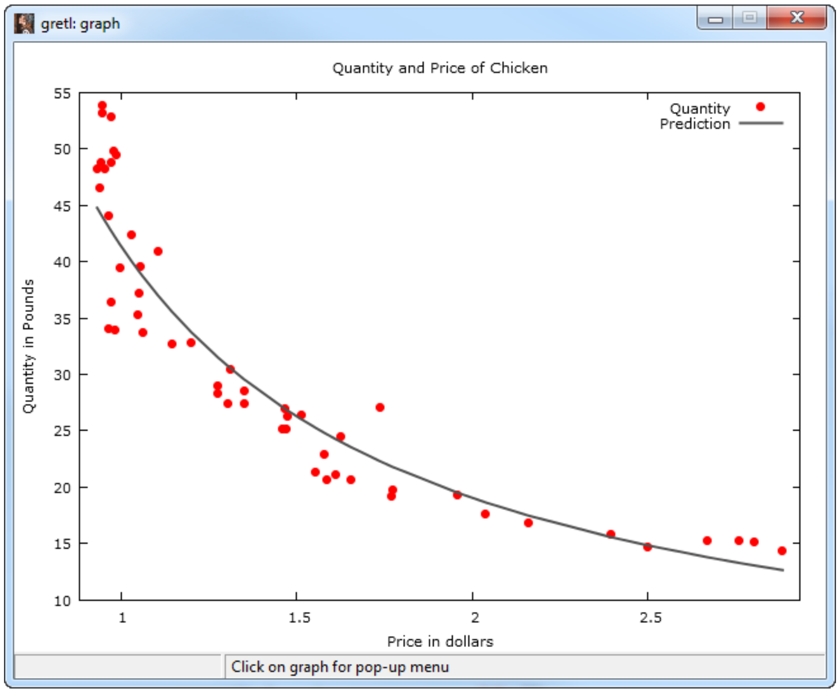


Figure 4.20: This is a plot generated from a log-log model of chicken demand.

The figure looks good. The nonlinear relationship between weight and price is quite evident and the fit is reasonable good.

4.6 Script

```
set echo off
# estimate model by LS and predict food_exp
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
scalar yhat0 = $coeff(const) + $coeff(income)*20

# prediction interval
ols food_exp const income
scalar yhat0 = $coeff(const) + $coeff(income)*20
scalar f=8013.2941+(8013.2941/40)+4.3818*(20-19.6047)^2
scalar ub=yhat0+2.0244*sqrt(f)
scalar lb=yhat0-2.0244*sqrt(f)

# prediction interval using accessors
ols food_exp const income
scalar yhat0=$coeff(const)+20*$coeff(income)
scalar sig2 = $ess/$df
scalar f = sig2 + sig2/$nobs + ((20-mean(income))^2)*($stderr(income)^2)
scalar lb = yhat0-critical(t,$df,0.025)*sqrt(f)
scalar ub = yhat0+critical(t,$df,0.025)*sqrt(f)

# correlations
ols food_exp const income --anova
c1 = corr(food_exp,$yhat)

# linear-log model
series l_income = ln(income)
ols food_exp const l_income
series yhat2 = $yhat
gnuplot yhat2 food_exp income

# simple data plot
open "@gretldir\data\poe\ch4sim1.gdt"
gnuplot e x

# residual plot
open "@gretldir\data\poe\ch4sim2.gdt"
ols y const x
series ehat = $uhat
gnuplot ehat x

# normality tests
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
series uhat1 = $uhat
summary uhat1
normtest uhat1 --jbera
normtest uhat1 --all
modtest --normality

# polynomial
```

```
open "@gretldir\data\poe\wa-wheat.gdt"
ols greenough const time
gnuplot greenough time

setinfo greenough -d "Wheat yield in tonnes" -n "Yield in tonnes"
gnuplot greenough --with-lines --time-series

series t3=time^3/1000000
ols greenough const t3
gnuplot greenough --with-lines --time-series

open "@gretldir\data\poe\wa-wheat.gdt"
series lyield = log(greenough)
ols lyield const time

# log-linear model
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ
scalar lb = $coeff(educ) - 1.96 * $stderr(educ)
scalar ub = $coeff(educ) + 1.96 * $stderr(educ)
print lb ub

open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ
series y = exp($yhat)
scalar corr1 = corr(y, wage)
scalar Rsquare = corr1^2
print corr1 Rsquare

# simple prediction in log-linear model
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ
scalar l_wage_12 = $coeff(const)+$coeff(educ)*12
scalar nat_pred = exp(l_wage_12)
scalar corrected_pred = nat_pred*exp($sigma^2/2)
print l_wage_12 nat_pred corrected_pred

smpl educ=12 --restrict
summary wage
smpl full

# prediction intervals using a loop
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ
scalar sig2 = $ess/$df
matrix sem = zeros(21,5)
loop for i = 1..21 --quiet
```

```
    scalar yh = ($coeff(const) + $coeff(educ)*i)
    scalar f = sig2 + sig2/$nobs + ((i-mean(educ))^2)*($stderr(educ)^2)
    sem[i,1]=i
    sem[i,2]= yh
    sem[i,3]=sqrt(f)
    sem[i,4]=exp(yh-critical(t,$df,0.025)*sqrt(f))
    sem[i,5]=exp(yh+critical(t,$df,.025)*sqrt(f))
endloop
print sem

nulldata 21 --preserve
series ed=sem[,1]
series wage=exp(sem[,2])
series lb=sem[,4]
series ub=sem[,5]

# corrected predictions in log-linear model
open "@gretldir\data\poe\newbroiler.gdt"
logs q p
ols l_q const l_p
series yht=$yhat
series pred = exp(yht)
series corrected_pred=pred*exp($sigma^2/2)
scalar r2= corr(corrected_pred,q)^2
gnuplot corrected_pred q p
```

Chapter 5

Multiple Regression Model

The multiple regression model is an extension of the simple model discussed in chapter 2. The main difference is that the multiple linear regression model contains more than one explanatory variable. This changes the interpretation of the coefficients slightly and requires another assumption. The general form of the model is shown in equation (5.1) below.

$$y_i = \beta_1 + \beta_2 x_{i2} + \cdots + \beta_K x_{iK} + e_i \quad i = 1, 2, \ldots, N \tag{5.1}$$

where y_i is your dependent variable, x_{ik} is the i^{th} observation on the k^{th} independent variable, $k = 2, 3, \ldots, K$, e_i is random error, and $\beta_1, \beta_2, \ldots, \beta_K$ are the parameters you want to estimate. Just as in the simple linear regression model, each error, e_i, has an average value of zero for each value of the independent variables; each has the same variance, σ^2, and are uncorrelated with any of the other errors. In order to be able to estimate each of the βs, none of the independent variables can be an exact linear combination of the others. This serves the same purpose as the assumption that each independent variable of the simple linear regression take on at least two different values in your dataset. The error assumptions can be summarized as $e_i | x_{i2}, x_{i3}, \ldots x_{iK}$ *iid* $(0, \sigma^2)$. Recall from chapter 2 that expression *iid* means that the errors are statistically independent from one another (and therefore uncorrelated) and each has the same probability distribution. Taking a random sample from a single population accomplishes this.

The parameters $\beta_2, \beta_3, \ldots, \beta_K$ are referred to as *slopes* and each slope measures the effect of a 1 unit change in x_{ik} on the average value of y_i, *holding all other variables in the equation constant.* The conditional interpretation of the coefficient is important to remember when using multiple linear regression.

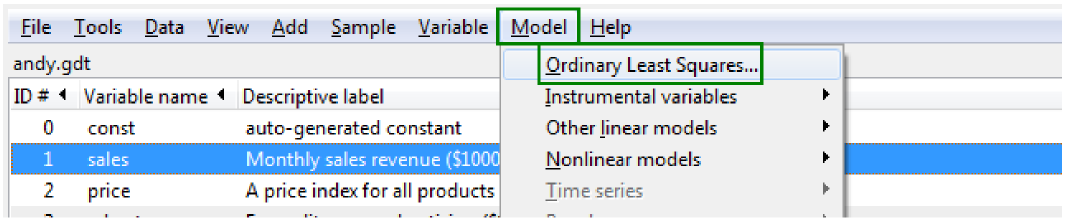
The example used in this chapter models the sales for Big Andy's Burger Barn. The model includes two explanatory variables and a constant.

$$sales_i = \beta_1 + \beta_2 price_i + \beta_3 advert_i + e_i \quad i = 1, 2, \ldots, N \tag{5.2}$$

where $sales_i$ is monthly sales in a given city and is measured in \$1,000 increments, $price_i$ is price of a hamburger measured in dollars, and $advert_i$ is the advertising expenditure also measured in thousands of dollars.

5.1 Linear Regression

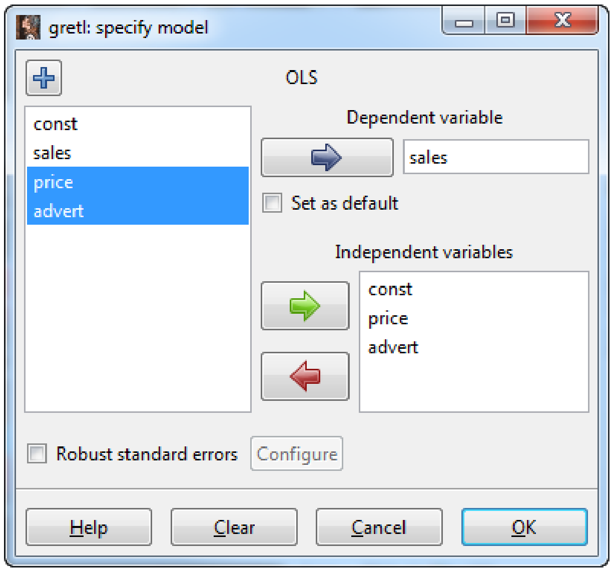
The parameters of the model are estimated using least squares which can be done using the pull-down menus and dialog boxes (GUI) or by using **gretl**'s handy scripting language (affectionately called **hansl**). Both of these will be demonstrated below. The GUI makes it easy to estimate this model using least squares. There are actually two ways to open the dialog box. The first is to use the pull-down menu. Select `Model>Ordinary Least Squares` from the main **gretl** window as shown below in Figure 5.1. This brings up the dialog box shown in Figure 5.2. As in chapter 2



andy.gdt

ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	sales	Monthly sales revenue ($1000
2	price	A price index for all products

Figure 5.1: Using the pull-down menu to open the ordinary least squares dialog box.



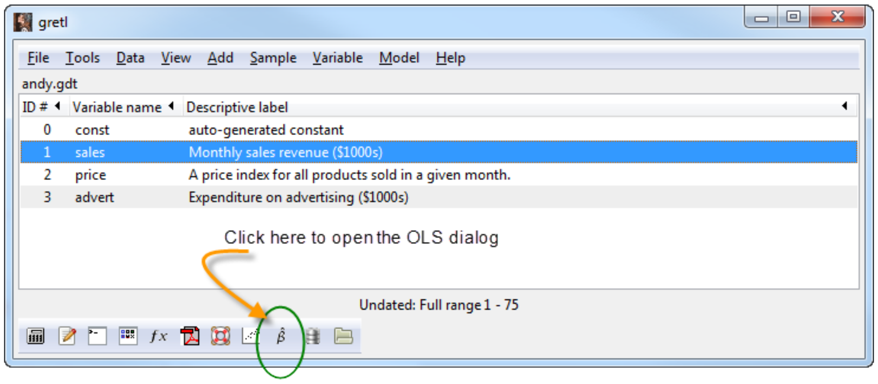
Dependent variable: sales

Independent variables: const, price, advert

Figure 5.2: The `specify model` dialog box for ordinary least squares (OLS)

you need to put the dependent variable (`sales`) and the independent variables (`const`, `price`, and `advert`) in the appropriate boxes. Click **OK** and the model is estimated. The results appear in Table 5.1 below.

There is a shortcut to get to the specify model dialog box. On the toolbar located at the bottom of the main **gretl** window is a button labeled $\hat{\beta}$. Clicking on this button as shown in Figure 5.3 will open the OLS specify model dialog box in Figure 5.2.


gretl
File Tools Data View Add Sample Variable Model Help
andy.gdt
Click here to open the OLS dialog
Undated: Full range 1 - 75

ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	sales	Monthly sales revenue ($1000s)
2	price	A price index for all products sold in a given month.
3	advert	Expenditure on advertising ($1000s)

Figure 5.3: The OLS shortcut button on the toolbar.

5.2 Big Andy's Burger Barn

Hansl is used to estimate the model for Big Andy's. The following two lines are typed into a script file, which is executed by clicking your mouse on the "gear" button of the script window.

```
open "@gretldir\data\poe\andy.gdt"
ols sales const price advert
scalar S_hat = $coeff(const) + $coeff(price)*5.5 + $coeff(advert)*1.2
```

This assumes that the **gretl** data set *andy.gdt* is installed at `c:\ProgramFiles(x86)\gretl\data\poe`. The results, in tabular form, are in Table 5.1 and match those in *POE4*.

In addition to providing information about how average sales change when price or advertising changes, the estimated equation can be used for prediction. To predict sales revenue for a price of \$5.50 and an advertising expenditure of \$1,200 we can use `genr` or `scalar` to do the computations. From the console,

```
Generated scalar S_hat (ID 4) = 77.6555
```

which also matches the result in *POE4*.

Model 1: OLS, using observations 1–75
Dependent variable: sales

	Coefficient	Std. Error	t-ratio	p-value
const	118.914	6.35164	18.7217	0.0000
price	−7.90785	1.09599	−7.2152	0.0000
advert	1.86258	0.683195	2.7263	0.0080

Mean dependent var	77.37467	S.D. dependent var	6.488537
Sum squared resid	1718.943	S.E. of regression	4.886124
R^2	0.448258	Adjusted R^2	0.432932
$F(2, 72)$	29.24786	P-value(F)	5.04e–10
Log-likelihood	−223.8695	Akaike criterion	453.7390
Schwarz criterion	460.6915	Hannan–Quinn	456.5151

Table 5.1: The regression results from Big Andy's Burger Barn

5.2.1 Variances and Covariances of Least Squares

The variances and covariances of the least squares estimator give us information about how precise our knowledge of the parameters is from estimating them. Smaller standard errors mean that our knowledge is more precise.

The precision of least squares (LS) depends on a number of factors.

1. Smaller variation in the dependent variable about its mean, σ^2, makes LS more precise.
2. Larger samples, N, improve LS precision.
3. More variation in the independent variables about their respective means makes LS more precise.
4. Less correlation between the least squares estimates, $corr(b_2, b_3)$, also improves LS precision.

The precision of least squares (and other estimators) is summarized by the **variance-covariance matrix**, which includes a measurement of the variance of the intercept, each slope, and covariance between each pair. The variances of the least squares estimator fall on the diagonal of this square matrix and the covariances in the off-diagonal elements.

$$cov(b_1, b_2, b_3) = \begin{bmatrix} var(b_1) & cov(b_1, b_2) & cov(b_1, b_3) \\ cov(b_1, b_2) & var(b_2) & cov(b_2, b_3) \\ cov(b_1, b_3) & cov(b_2, b_3) & var(b_2) \end{bmatrix} \tag{5.3}$$

All of these have to be estimated from the data, and generally depends on your estimate of the overall variance of the model, $\hat{\sigma}^2$ and correlations among the independent variables. To print an estimate of the variance-covariance matrix following a regression use the `--vcv` option with your regression in **gretl** :

```
ols sales const price advert --vcv
```

The result is

Coefficient covariance matrix

const	price	advert	
40.343	−6.7951	−0.74842	const
	1.2012	−0.01974	price
		0.46676	advert

For instance, the estimated variance of b_1–the intercept–is 40.343 and the estimated covariance between the LS estimated slopes b_2 and b_3 is -0.01974.

A (estimated) **standard error** of a coefficient is the square root of its (estimated) variance, $\widehat{se}(b_2) = \sqrt{\widehat{var}(b_2)}$. These are printed in the output table along with the least squares estimates, t-ratios, and their p-values.

5.2.2 Confidence Intervals

Confidence intervals are obtained using the `scalar` command in the same way as in chapter 3. A 95% confidence interval for β_2, the coefficient of the price variable is generated:

```
ols sales const price advert --vcv
scalar bL = $coeff(price) - critical(t,$df,0.025) * $stderr(price)
scalar bU = $coeff(price) + critical(t,$df,0.025) * $stderr(price)
printf "\nThe lower = %.2f and upper = %.2f confidence limits\n", bL, bU
```

The output from the script is:

```
The lower = -10.09 and upper = -5.72 confidence limits
```

This nifty piece of output uses the function called, `printf`. `printf` stands for **print format** and it is used to gain additional control over how results are printed to the screen. In this instance we've combined descriptive text and numerical results. The syntax is a bit tricky, so I will explain a little about it. I will be using it extensively in the rest of this book so that you get the used to it. Once you use it, its mystery quickly evaporates–the syntax is really quite elegant.

The `printf` function is divided into two parts. The first part consists of what you want to write to the screen, and the second contains the numbers from your output that you want placed within the first part.

Dressing up printed output with printf
```
printf "\nThe lower = %.2f and upper = %.2f confidence limits\n", bL, bU
```

The first part, called the **format string**, is enclosed in double quotes. The `\n` command stands for 'new line' and it basically tells **gretl** to issue a line feed (in old computer lingo, that means go to a new line). It is used at the beginning and the end of the format string and is not strictly necessary. In this case, a line feed is given before and after the format string to give a little more white space to your printed output. If you want line feeds, be sure to put these inside the double quotes that enclose the format string.

Within this 'sentence' or 'format string' are two **format commands**. A format command tells **gretl** how the numerical results are to be printed. A format command begins with the `%` symbol and is followed by instructions about how many digits of the numerical result you want it to print. These formats are basically adopted from the C programming language. `%f` a fixed point format and the number that falls between the `%.` and `f` indicates how many decimal places to print. So, `%.2f` tells **gretl** to print only two numbers to the right of the decimal.

Recognized numeric formats are `%s`, `%e`, `%E`, `%f`, `%g`, `%G` and `%d`[1], 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 `%s` should be used for strings.

The second part of the `printf` command contains the values to be printed at the each of the format commands. There has to be one result for each format command. These are separated by commas. Since there are two format commands, **gretl** is expecting two results to be listed. The result computed and stored in `bL` will be printed at the first format command, `%.2f`, and the one in `bU` will be printed at the second `%.2f`. 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, (b) expressions that are valid for the `genr` command, or (c) the special functions `varname()` or `date()`.

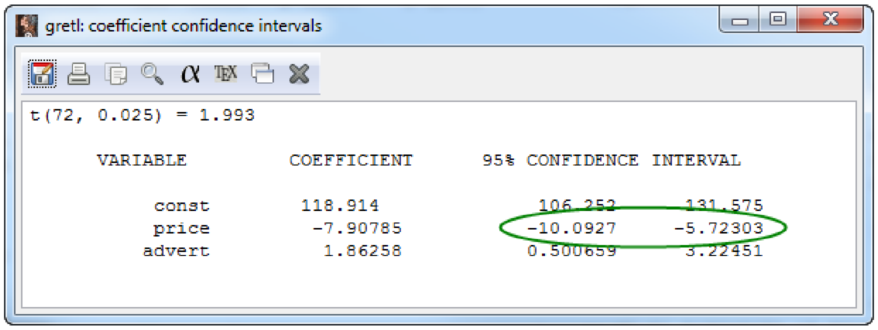
Remember, you can also summon the 95% confidence intervals from the model window using the pull-down menu by choosing `Analysis>Confidence intervals for coefficients`. The confidence interval for β_2 is shown below in Figure 5.4.

You can also estimate intervals for linear combinations of parameters as we did in chapter 4. Suppose Big Andy wants to increase sales next week by lowering price and spending more on advertising. If he increases advertising by \$800 and lowers price by 40 cents the change in expected sales would be

$$\lambda = E(sales_1) - E(sales_0) = -0.4\beta_2 + 0.8\beta_3 \tag{5.4}$$

The estimate of λ is obtained by replacing the unknown parameters with the least squares estimates. The standard error of this linear combination can be calculated in the same fashion as discussed in section 3.6. A 90% interval is constructed using the script:

[1]`%e` is for scientific notation with lower case e, `%E` is scientific upper case, `%g` picks the shorter of `%e` or `%f`, and `%G` picks the shorter of `%E` or `%f`. The format command `%d` is for a signed decimal integer.



```
t(72, 0.025) = 1.993

     VARIABLE         COEFFICIENT      95% CONFIDENCE INTERVAL

         const         118.914          106.252      131.575
         price          -7.90785        -10.0927      -5.72303
        advert           1.86258          0.500659     3.22451
```

Figure 5.4: The confidence intervals produced from the GUI through the model window. In the model window, choose `Analysis>Confidence intervals for coefficients`

```
scalar chg = -0.4*$coeff(price)+0.8*$coeff(advert)
scalar se_chg=sqrt((-0.4)^2*$vcv[2,2]+(0.8^2)*$vcv[3,3]+2*(-0.4)*(0.8)*$vcv[2,3])
scalar lb = chg-critical(t,$df,.05)*se_chg
scalar ub = chg+critical(t,$df,.05)*se_chg
printf "\nExpected Change = %.4f and SE = %.4f\n",chg,se_chg
printf "\nThe 90%% confidence interval is [%.3f, %.3f]\n",lb,ub
```

This produces the expected result:

```
Expected Change = 4.6532 and SE = 0.7096
The 90% confidence interval is [3.471, 5.836]
```

The only trick here is to get the percent % symbol into the print statement; to do so it must be preceded by another percent symbol, %; hence, 90%% appears in line 6 to print 90%.

5.2.3 *t*-Tests, Critical Values, and *p*-values

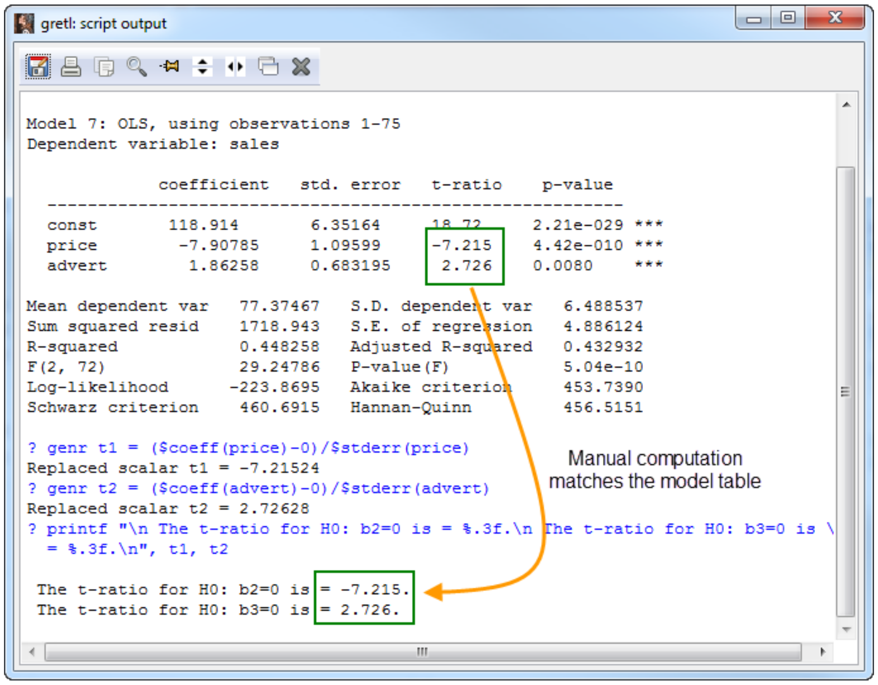
In section 3.4 the GUI was used to obtain test statistics, critical values and *p*-values. However, it is often much easier to use the the `genr` or `scalar` commands from either the console or as a script to compute these. In this section, the scripts will be used to test various hypotheses about the sales model for Big Andy.

Significance Tests

Multiple regression models includes several independent variables because one believes that each as an independent effect on the mean of the dependent variable. To confirm this belief it is customary to perform tests of individual parameter significance. If the parameter is zero, then the variable does not belong in the model. In **gretl** the t-ratio associated with the null hypothesis that $\beta_k = 0$ against the alternative $\beta_k \neq 0$ is printed in the regression results along side the associated p-value. For the sake of completeness, these can be computed manually using a script as found below. For t-ratios and one- and two-sided hypothesis tests the appropriate commands are:

```
ols sales const price advert
scalar t1 = ($coeff(price)-0)/$stderr(price)
scalar t2 = ($coeff(advert)-0)/$stderr(advert)
printf "\n The t-ratio for H0: b2=0 is = %.3f.\n\
The t-ratio for H0: b3=0 is = %.3f.\n", t1, t2
```

The results shown in Figure 5.5 As you can see, the automatic results and the manually generated



```
Model 7: OLS, using observations 1-75
Dependent variable: sales

              coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const        118.914       6.35164      18.72    2.21e-029 ***
  price         -7.90785     1.09599      -7.215   4.42e-010 ***
  advert         1.86258     0.683195      2.726   0.0080    ***

Mean dependent var   77.37467   S.D. dependent var   6.488537
Sum squared resid    1718.943   S.E. of regression   4.886124
R-squared            0.448258   Adjusted R-squared   0.432932
F(2, 72)             29.24786   P-value(F)           5.04e-10
Log-likelihood      -223.8695   Akaike criterion     453.7390
Schwarz criterion    460.6915   Hannan-Quinn         456.5151

? genr t1 = ($coeff(price)-0)/$stderr(price)
Replaced scalar t1 = -7.21524
? genr t2 = ($coeff(advert)-0)/$stderr(advert)
Replaced scalar t2 = 2.72628
? printf "\n The t-ratio for H0: b2=0 is = %.3f.\n The t-ratio for H0: b3=0 is \
  = %.3f.\n", t1, t2

 The t-ratio for H0: b2=0 is = -7.215.
 The t-ratio for H0: b3=0 is = 2.726.
```

Figure 5.5: Notice that the usual model estimation results produced by **gretl** prints the t-ratios needed for parameter significance by default. These match the manual computation.

ones match perfectly.

One of the advantages of doing t-tests manually is that you can test hypotheses other than parameter significance. You can test hypothesis that the parameter is different from values other than zero, test a one-sided hypotheses, or test a hypotheses involving a linear combinations of parameters.

One-tail Alternatives

If a decrease in price increases sales revenue then we can conclude that demand is elastic. So, if $\beta_2 \geq 0$ demand is elastic and if $\beta_2 < 0$ it is inelastic. To test $H_0 : \beta_2 \geq 0$ versus $H_1 : \beta < 0$, the test statistic is the usual t-ratio.

```
scalar t1 = ($coeff(price)-0)/$stderr(price)
pvalue t $df t1
```

The rejection region for this test lies to the left of $-t_c$, which is the α level critical value from the distribution of t. This is a perfect opportunity to use the pvalue function. The result is:

```
t(72): area to the right of -7.21524 =~ 1
(to the left: 2.212e-010)
(two-tailed value = 4.424e-010; complement = 1)
```

You can see that the area to the left of -7.21524 is close to zero. That is less than 5% nominal level of the test and therefore we reject that β_2 is non-negative.

A test of whether a dollar of additional advertising will generate at least a dollar's worth of sales is expressed parametrically as $H_0 : \beta_3 \leq 1$ versus $H_1 : \beta_3 > 1$. This requires a new t-ratio and again we use the `pvalue` function to conduct the test.

```
scalar t3 = ($coeff(advert)-1)/$stderr(advert)
pvalue t $df t3
```

The results are

```
t(72): area to the right of 1.26257 = 0.105408
(two-tailed value = 0.210817; complement = 0.789183)
```

The rejection region for this alternative hypothesis lies to the right of the computed t-ratio. That implies that the p-value is 0.105. At 5% level of significance, this null hypothesis cannot be rejected.

Linear Combinations of Parameters

Big Andy's advertiser claims that dropping the price by 20 cents will increase sales more than spending an extra \$500 on advertising. This can be translated into a parametric hypothesis that can be tested using the sample. If the advertiser is correct then $-0.2\beta_2 > 0.5\beta_3$. The hypothesis to be tested is:

$$H_O : -0.2\beta_2 - 0.5\beta_3 \leq 0$$
$$H_1 : -0.2\beta_2 - 0.5\beta_3 > 0$$

The test statistic is

$$t = \frac{-0.2b_2 - 0.5b_3}{se(-0.2b_2 - 0.5b_3)} \sim t_{72} \tag{5.5}$$

provided the null hypothesis is true. The script is

```
ols sales const price advert --vcv
scalar chg = -0.2*$coeff(price)-0.5*$coeff(advert)
scalar se_chg=sqrt((-0.2)^2*$vcv[2,2]+((-0.5)^2)*$vcv[3,3]\
             +2*(-0.2)*(-0.5)*$vcv[2,3])
scalar t_ratio = chg/se_chg
pvalue t $df t_ratio
```

which generates the needed information to perform the one-sided test.

```
t(72): area to the right of 1.62171 = 0.0546189
(two-tailed value = 0.109238; complement = 0.890762)
```

The p-value is $P(t_{72} > 1.62171) = 0.0546189$. At 5% we cannot reject the null (but we could at 10%).

5.3 Polynomials

One way to allow for nonlinear relationships between independent and dependent variables is to introduce polynomials of the regressors into the model. In this example the marginal effect of an additional dollar of advertising is expected to diminish as more advertising is used. The model becomes:

$$sales_i = \beta_1 + \beta_2 price_i + \beta_3 advert_i + \beta_4 advert^2 + e_i \quad i = 1, 2, \ldots, N \tag{5.6}$$

To estimate the parameters of this model, one creates the new variable, $advert^2$, adds it to the model, and uses least squares.

```
series a2 = advert^2
ols sales price advert a2
```

which produces

OLS, using observations 1–75
Dependent variable: sales

	Coefficient	Std. Error	t-ratio	p-value
const	109.719	6.79905	16.1374	0.0000
price	−7.64000	1.04594	−7.3044	0.0000
advert	12.1512	3.55616	3.4170	0.0011
a2	−2.76796	0.940624	−2.9427	0.0044

Mean dependent var	77.37467	S.D. dependent var	6.488537
Sum squared resid	1532.084	S.E. of regression	4.645283
R^2	0.508235	Adjusted R^2	0.487456
$F(3, 71)$	24.45932	P-value(F)	5.60e–11
Log-likelihood	−219.5540	Akaike criterion	447.1080
Schwarz criterion	456.3780	Hannan–Quinn	450.8094

The variable `a2`, which is created by squaring `advert`, is a simple example of what is sometimes referred to as an **interaction** variable. The simplest way to think about an interaction variable is that you believe that its effect on the dependent variable depends on another variable–the two variables interact to determine the average value of the dependent variable. In this example, the effect of advertising on average sales depends on the level of advertising itself.

Another way to square variables is to use the square command

```
square advert
```

This creates a variable `sq_advert` and adds it to the variable list. Notice that **gretl** just adds the `sq_` prefix to the existing variable name. You can square multiple variables at a time by just by adding them to the `square` command's list.

5.3.1 Marginal Effects

When variables interact, the marginal effect of one variable on the mean of another has to be computed manually based on calculus. Taking the partial derivative of average sales with respect to advertising yields produces the marginal effect on average sales of an increase in advertising;

$$\frac{\partial E(sales)}{\partial advert} = \beta_3 + 2\beta_4 advert \tag{5.7}$$

The magnitude of the marginal effect depends on the parameters as well as on the level of advertising. In the example marginal effect is evaluated at two points, *advert*=.5 and *advert*=2. The code is:

```
scalar me1 = $coeff(advert)+2*(0.5)*$coeff(a2)
scalar me2 = $coeff(advert)+2*2*$coeff(a2)
printf "\nThe marginal effect at $500 (advert=.5) is \
%.3f and at $2000 (advert=2) is %.3f\n",me1,me2
```

and the result is:

```
The marginal effect at $500 (advert=.5) is 9.383 and at $2000 (advert=2) is 1.079
```

5.3.2 Optimal Advertising: Nonlinear Combinations of Parameters

The optimal level of advertising, $advert_o$, is defined in this example to be the amount that maximizes net sales. Andy will advertise up to the point where another dollar of expenditure adds at least one dollar of additional sales-and no more. At this point the marginal effect is equal to one,

$$\beta_3 + 2\beta_4 advert_o = 1 \tag{5.8}$$

Solving *advert* in terms of the parameters

$$\lambda = advert_o = \frac{1-\beta_3}{2\beta_4} \tag{5.9}$$

which is nonlinear in the parameters of the model. A consistent estimate of the optimal level of advertising can be obtained by substituting the least squares estimates for the parameters on the right-hand side. Estimating the standard error via the Delta method requires some calculus, but it is quite straightforward to do in **gretl**.

The Delta method is based on a first-order Taylor's series expansion of a function that involves the parameters of the model. It relies on the asymptotic normality of the estimator you are using. Let $\boldsymbol{\beta}$ be a 2×1 vector of parameters; an intercept and slope. Consider a possibly nonlinear function of a parameters $g(\boldsymbol{\beta})$. Also, let's say that we estimate a set of parameters $\boldsymbol{\beta}$ using an estimator called $\boldsymbol{b}$ and that $\boldsymbol{b} \overset{a}{\sim} N(\boldsymbol{\beta}, V)$. So far, we've described the least squares estimator of the simple regression. Then, by the Delta theorem, the nonlinear function evaluated at the estimates has the following approximate distribution:

$$g(\boldsymbol{b}) \overset{a}{\sim} N(g(\boldsymbol{\beta}), G(\boldsymbol{\beta})VG(\boldsymbol{\beta})^T) \tag{5.10}$$

where $G(\boldsymbol{\beta}) = \partial g(\boldsymbol{\beta})/\partial\boldsymbol{\beta}^T$. In order to use the Delta Method, you have to take the partial derivatives of the function, which in our example is a hypothesis, with respect to each parameter in the model. That is, you need the Jacobian.

In the example, $g(\boldsymbol{\beta}) = 1 - \beta_3/2\beta_4$. Taking the derivatives with respect to each of the parameters, β_1, β_2, β_3, and β_4 yields:

$$\begin{aligned} d_1 &= \frac{\partial advert_o}{\partial \beta_1} = 0 \\ d_2 &= \frac{\partial advert_o}{\partial \beta_2} = 0 \\ d_3 &= \frac{\partial advert_o}{\partial \beta_3} = -\frac{1}{2\beta_4} \qquad (5.11) \\ d_4 &= \frac{\partial advert_o}{\partial \beta_4} = -\frac{1-\beta_3}{2\beta_4^2} \qquad (5.12) \end{aligned}$$

Note that the derivatives with respect to β_1 and β_2 are 0. To use the Delta method, simply replace the unknown parameters in equation (5.9) with least squares estimates. Then to get the estimated standard error of $\hat{\lambda}$, substituted estimates into the derivatives d_3 and d_4, and compute

$$var(\hat{\lambda}) = \begin{pmatrix} 0 & 0 & \hat{d}_3 & \hat{d}_4 \end{pmatrix} [\widehat{cov}(b_1, b_2, b_3, b_4)] \begin{pmatrix} 0 \\ 0 \\ \hat{d}_3 \\ \hat{d}_4 \end{pmatrix} \qquad (5.13)$$

This looks harder to do than it actually is. The **gretl** script to compute the variance and standard error is:

```
1  ols sales const price advert a2 --vcv
2  matrix b = $coeff
3  matrix cov = $vcv
4  scalar lambda = (1-b[3])/(2*b[4])
5  scalar d3 = -1/(2*b[4])
6  scalar d4 = -1*(1-b[3])/(2*b[4]^2)
7  matrix d = { 0, 0, d3, d4}
8  scalar v = d*cov*d'
9  scalar se = sqrt(v)
10 scalar lb = lambda - critical(t,$df,.025)*se
11 scalar ub = lambda + critical(t,$df,.025)*se
12 printf "\nThe estimated optimal level of advertising is $%.2f.\n",1000*lambda
13 printf "\nThe 95%% confidence interval is ($%.2f, $%.2f).\n",1000*lb,1000*ub
```

The first line estimates the model using least sqaures and the `--vcv` option is used to print the covariance matrix. In line 2 the entire set of coefficents is saved into a vector (a one row matrix in this case) called `b`. This will make the syntax that follows easier since each coefficient can be referred to by its position in the vector, e.g., the third coefficient in `b` is `b[3]`. In line 3 the covariance matrix is saved as `cov`. In line 4 the least squares estimates are substituted for the unknown parameters. In lines 5 and 6 the analytical derivatives are evaluated at the estimates. The matrix `d` is 1×4 and contains the derivatives of the hypothesis with respect to each of the parameters. The next line computes variance in equation (5.13). Finally, the square root is taken to get the standard error and the confidence bounds are computed in lines 10 and 11 and printed in 12 and 13.

```
The estimated optimal level of advertising is $2014.34.
The 95% confidence interval is ($1757.67, $2271.01).
```

According to this estimate the optimal level of advertising is \$2014.34 and the 95% confidence interval is (\$1758, \$2271).

5.4 Interactions

Interaction among variables was introduced in the preceding section for creating polynomial terms. The concept is very general can be applied to any situation where the effect of a change in one variable on the mean of the dependent variable depends on another variable.

5.4.1 Basic Interactions of Continuous Variables

The basic model considered is

$$pizza = \beta_1 + \beta_2 age + \beta_3 income + e \tag{5.14}$$

It is proposed that as a person grows older, his or her marginal propensity to spend on pizza declines-this implies that the coefficient β_3 depends on a person's age.

$$\beta_3 = \beta_4 + \beta_5 age \tag{5.15}$$

Substituting this into the model produces

$$pizza = \beta_1 + \beta_2 age + \beta_3 income + \beta_4(income \times age) + e \tag{5.16}$$

This introduces a new variable, $income \times age$, which is an interaction variable. The marginal effect of unit increase in *age* in this model depends on *income* and the marginal effect of an increase in *income* depends on *age*.

The interaction could be created in **gretl** using the `genr` or `series` command. The data for the following example are found in the *pizza4.gdt* dataset.

```
open "@gretldir\data\poe\pizza4.gdt"
genr inc_age = income*age
ols pizza const age income inc_age
```

The result is

Model 1: OLS, using observations 1–40
Dependent variable: pizza

	Coefficient	Std. Error	t-ratio	p-value
const	161.465	120.663	1.3381	0.1892
age	−2.97742	3.35210	−0.8882	0.3803
income	6.97991	2.82277	2.4727	0.0183
inc_age	−0.123239	0.0667187	−1.8471	0.0730

Mean dependent var	191.5500	S.D. dependent var	155.8806
Sum squared resid	580608.7	S.E. of regression	126.9961
R^2	0.387319	Adjusted R^2	0.336262
$F(3, 36)$	7.586038	P-value(F)	0.000468
Log-likelihood	−248.4166	Akaike criterion	504.8332
Schwarz criterion	511.5887	Hannan–Quinn	507.2758

The marginal effect of age on pizza expenditure can be found by taking the partial derivative of the regression function with respect to age

$$\frac{\partial E(pizza)}{age} = \beta_2 + \beta_4 income \tag{5.17}$$

Comparing the marginal effect of another year on average expenditures for two individuals, one with $25,000 in income

$$= b_2 + b_4 \times 25 = -2.977 + (-0.1232)25 = -6.06. \tag{5.18}$$

To carry this out in a script with income at $25,000 and $90,000

```
open "@gretldir\data\poe\pizza4.gdt"
series inc_age=income*age
ols pizza const age income inc_age
scalar me1 = $coeff(age)+$coeff(inc_age)*25
scalar me2 = $coeff(age)+$coeff(inc_age)*90
printf "\nThe marginal effect of age for one \
with $25,000/year income is %.2f.\n",me1
printf "\nThe marginal effect of age for one \
with $90,000/year income is %.2f.\n",me2
```

This yields:

```
The marginal effect of age for one with $25,000/year income is -6.06.

The marginal effect of age for one with $90,000/year income is -14.07.
```

5.4.2 Log-Linear Model

In this example the simple regression first considered in chapter 4 is modified to include more variables and an interaction. The model adds experience to the model

$$\ln(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + e \tag{5.19}$$

In this model suppose that the marginal effect of another year of schooling depends on how much experience the worker has. This requires adding an interaction

$$\ln(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 (educ \times exper) + e \tag{5.20}$$

The marginal effect of another year of experience is

$$\frac{\partial E[\ln(wage)]}{exper}|_{educ\ fixed} = \beta_3 + \beta_4 educ \tag{5.21}$$

In percentage terms the marginal effect of another year of experience is $100(\beta_3 + \beta_4 educ)$. The model can be estimated and the marginal effect computed easily with **hansl**

```
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
series ed_exp=educ*exper
ols l_wage const educ exper ed_exp
scalar me8 = $coeff(exper)+8*$coeff(ed_exp)
scalar me16 = $coeff(exper)+16*$coeff(ed_exp)
printf "\nThe marginal effect of exper for one \
with 8 years of schooling is %.3f%%\n",100*me8
printf "\nThe marginal effect of exper for one \
with 16 years of schooling is %.3f%%.\n",100*me16
```

The result is

```
The marginal effect of exper for one with 8 years of schooling is 0.604%.
The marginal effect of exper for one with 16 years of schooling is 0.575%.
```

5.5 Goodness-of-Fit

Other important output is included in Table 5.1. For instance, you'll find the sum of squared errors (SSE) which **gretl** refers to as "Sum squared resid." In this model $SSE = 1718.94$. To obtain the estimated variance, $\hat{\sigma}^2$, divide SSE by the available degrees of freedom to obtain

$$\hat{\sigma}^2 = \frac{SSE}{N-K} = \frac{1718.94}{75-3} = 23.874 \tag{5.22}$$

The square root of this number is referred to by **gretl** as the "S.E. of regression" and is reported to be 4.88612. **Gretl** also reports R^2 in this table. If you want to compute your own versions of these statistics using the total sum of squares from the model, you'll have to use `Analysis>ANOVA` from the model's pull-down menu to generate the ANOVA table. Refer to section 4.2 for details.

To compute your own from the standard **gretl** output recall that

$$\hat{\sigma}_y = \sqrt{\frac{SST}{N-1}} \tag{5.23}$$

The statistic $\hat{\sigma}_y$ is printed by **gretl** and referred to as "S.D. of dependent variable" which is reported to be 6.48854. A little algebra reveals

$$SST = (N-1)\hat{\sigma}_y^2 = 74 * 6.48854 = 3115.485 \tag{5.24}$$

Then,

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{1718.94}{3115.485} = 0.448 \tag{5.25}$$

Otherwise, the goodness-of-fit statistics printed in the **gretl** regression output or the ANOVA table are perfectly acceptable.

Gretl also reports the **adjusted** R^2 in the standard regression output. The adjusted R^2 imposes a small penalty to the usual R^2 when a variable is added to the model. Adding a variable with any correlation to y always reduces SSE and increases the size of the usual R^2. With the adjusted version, the improvement in fit may be outweighed by the penalty and adjusted R^2 could become smaller as variables are added. The formula is:

$$\bar{R}^2 = 1 - \frac{SSE/(N-K)}{SST/(N-1)} \tag{5.26}$$

This sometimes referred to as "R-bar squared," (i.e., $\bar{R}^2$) although in **gretl** it is called "adjusted R-squared." For Big Andy's Burger Barn the adjusted R-squared is equal to 0.4329.

Once again the the `printf` command gives us some additional control over the format of the output. At the end of line 3 you'll find an extra \. This is **gretl**'s line continuation command. It tells **gretl** to continue reading the next line. It basically joins lines 3 and 4. The continuation command makes programs easier to print on a page. It looks slightly odd here since it immediately follows the line feed `\n`, but `\n\` actually consists of two commands: a line feed and a continuation.

In this the critical values for the t_{72} and the p-values for the two statistics can be easily obtained using the command

```
scalar c=critical(t,$df,0.025)
pvalue t $df t1
pvalue t $df t2
```

These last three commands produce the output shown below:

```
Generated scalar c (ID 8) = 1.99346
t(72): area to the right of -7.21524 =~ 1
(to the left: 2.212e-010)

(two-tailed value = 4.424e-010; complement = 1)
t(72): area to the right of 1.26257 = 0.105408
(two-tailed value = 0.210817; complement = 0.789183)
```

It is interesting to note that when a negative t-ratio is used in the `pvalue` function, **gretl** returns both the area to its right, the area to its left and the sum of the two areas. So, for the alternative hypothesis that the coefficient on `price` is less than zero (against the null that it is zero), the p-value is the area to the left of the computed statistic, which in this example is essentially zero.

The `pvalue` command can also be used as a function to return a scalar that can be stored by **gretl**.

```
scalar c=critical(t,$df,0.025)
scalar p1=pvalue(t, $df, t1)

scalar p2=pvalue(t, $df, t2)
printf "\nThe .025 critical value from the t with %d degrees of freedom \
is %.3f.\n The pvalue from H0: b2=0 is %.3f and \
from H0: b3=1 is %.3f.\n",$df,c,p1,p2
```

This prints the following to the screen:

```
The .025 critical value from the t with 72 degrees of freedom  is 1.993.
 The pvalue from H0: b2=0 is 1.000 and  from H0: b3=1 is 0.105
```

5.6 Script

```
set echo off
open "@gretldir\data\poe\andy.gdt"
#Change the descriptive labels and graph labels
setinfo sales -d "Monthly Sales revenue ($1000)" -n "Monthly Sales ($1000)"
setinfo price -d "$ Price" -n "Price"
setinfo advert -d "Monthy Advertising Expenditure ($1000)" -n \
    "Monthly Advertising ($1000)

# print the new labels to the screen
labels

# summary statistics
summary sales price advert

# confidence intervals
ols sales const price advert --vcv
scalar bL = $coeff(price) - critical(t,$df,0.025) * $stderr(price)
scalar bU = $coeff(price) + critical(t,$df,0.025) * $stderr(price)
printf "The lower = %.2f and upper = %.2f confidence limits", bL, bU

# linear combination of parameters
ols sales const price advert --vcv
scalar chg = -0.4*$coeff(price)+0.8*$coeff(advert)
scalar se_chg=sqrt((-0.4)^2*$vcv[2,2]+(0.8^2)*$vcv[3,3]\
              +2*(-0.4)*(0.8)*$vcv[2,3])
scalar lb = chg-critical(t,$df,.05)*se_chg
scalar ub = chg+critical(t,$df,.05)*se_chg
printf "\nExpected Change = %.4f and SE = %.4f\n",chg,se_chg
printf "\nThe 90%% confidence interval is [%.3f,%.3f]\n",lb,ub

# significance tests
ols sales const price advert
scalar t1 = ($coeff(price)-0)/$stderr(price)
scalar t2 = ($coeff(advert)-0)/$stderr(advert)
printf "\n The t-ratio for H0: b2=0 is = %.3f.\n\
The t-ratio for H0: b3=0 is = %.3f.\n", t1, t2

pvalue t $df t1
scalar t3 = ($coeff(advert)-1)/$stderr(advert)
pvalue t $df t3

# t-test of linear combination
ols sales const price advert --vcv
scalar chg = -0.2*$coeff(price)-0.5*$coeff(advert)
scalar se_chg=sqrt((-0.2)^2*$vcv[2,2]+((-0.5)^2)*$vcv[3,3]\
              +2*(-0.2)*(-0.5)*$vcv[2,3])
scalar t_ratio = chg/se_chg
pvalue t $df t_ratio
```

```

# interaction creates nonlinearity
series a2 = advert*advert
ols sales const price advert a2 --vcv
scalar me1 = $coeff(advert)+2*(0.5)*$coeff(a2)
scalar me2 = $coeff(advert)+2*2*$coeff(a2)
printf "\nThe marginal effect at \$500 (advert=.5) is %.3f \
and at \$2000 is %.3f\n",me1,me2

# delta method for nonlinear hypotheses
ols sales const price advert a2 --vcv
matrix b = $coeff
matrix cov = $vcv
scalar lambda = (1-b[3])/(2*b[4])
scalar d3 = -1/(2*b[4])
scalar d4 = -1*(1-b[3])/(2*b[4]^2)
matrix d = { 0, 0, d3, d4}
scalar v = d*cov*d'
scalar se = sqrt(v)
scalar lb = lambda - critical(t,$df,.025)*se
scalar ub = lambda + critical(t,$df,.025)*se
printf "\nThe estimated optimal level of advertising is $%.2f.\n",1000*lambda
printf "\nThe 95%% confidence interval is ($%.2f, $%.2f).\n",1000*lb,1000*ub

# interaction and marginal effects
open "@gretldir\data\poe\pizza4.gdt"
series inc_age=income*age
ols pizza const age income inc_age
scalar me1 = $coeff(age)+$coeff(inc_age)*25
scalar me2 = $coeff(age)+$coeff(inc_age)*90
printf "\nThe marginal effect of age for someone\
with $25,000/year income is %.2f.\n",me1
printf "\nThe marginal effect of age for someone\
with $90,000/year income is %.2f.\n",me2

open "@gretldir\data\poe\cps4_small.gdt"
logs wage
series ed_exp=educ*exper
ols l_wage const educ exper ed_exp
scalar me8 = $coeff(exper)+8*$coeff(ed_exp)
scalar me16 = $coeff(exper)+16*$coeff(ed_exp)
printf "\nThe marginal effect of exper for someone\
with 8 years of schooling is %.3f%%.\n",100*me8
printf "\nThe marginal effect of exper for someone\
with 16 years of schooling is %.3f%%.\n",100*me16
```

Chapter 6

Further Inference in the Multiple Regression Model

In this chapter several extensions of the multiple linear regression model are considered. First, we test joint hypotheses about parameters in a model and then learn how to impose linear restrictions on the parameters. A condition called *collinearity* is also explored.

6.1 *F*-test

An F-statistic can be used to test multiple hypotheses in a linear regression model. In linear regression there are several different ways to derive and compute this statistic, but each yields the same result. The one used here compares the sum of squared errors (SSE) in a regression model estimated under the null hypothesis (H_0) to the SSE of a model under the alternative (H_1). If the sum of squared errors from the two models are similar, then there is not enough evidence to reject the restrictions. On the other hand, if imposing restrictions implied by H_0 alter SSE substantially, then the restrictions it implies don't fit the data and we reject them.

In the Big Andy's Burger Barn example we estimated the model

$$sales_i = \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 + e \tag{6.1}$$

Suppose we wish to test the hypothesis that advertising has no effect on average sales against the alternative that it does. Thus, $H_0 : \beta_3 = \beta_4 = 0$ and $H_1 : \beta_3 \neq 0$ or $\beta_4 \neq 0$. Another way to express this is in terms of the models each hypothesis implies.

$$\begin{aligned} H_0 :& \beta_1 + \beta_2 price + e \\ H_1 :& \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 + e \end{aligned}$$

The model under H_0 is **restricted** compared to the model under H_1 since in it $\beta_3 = 0$ and $\beta_4 = 0$. The F-statistic used to test H_0 versus H_1 estimates each model by least squares and compares their respective sum of squared errors using the statistic:

$$F = \frac{(SSE_r - SSE_u)/J}{SSE_u/(N-K)} \sim F_{J,N-K} \qquad \text{if } H_0 \text{ is true} \tag{6.2}$$

The sum of squared errors from the unrestricted model (H_1) is denoted SSE_u and that of the restricted model (H_0) is SSE_r. The numerator is divided by the number of hypotheses being tested, J. In this case that is 2 since there are two restrictions implied by H_0. The denominator is divided by the total number of degrees of freedom in the unrestricted regression, $N - K$. N is the sample size and K is the number of parameters in the unrestricted regression. When the errors of your model are (1) independently and identically distributed (*iid*) normals with zero mean and constant variance (e_t *iid* $N(0, \sigma^2)$) and (2) H_0 is true, then this statistic has an F distribution with J numerator and $N - K$ denominator degrees of freedom. Choose a significance level and compute this statistic. Then compare its value to the appropriate critical value from the F table or compare its p-value to the chosen significance level.

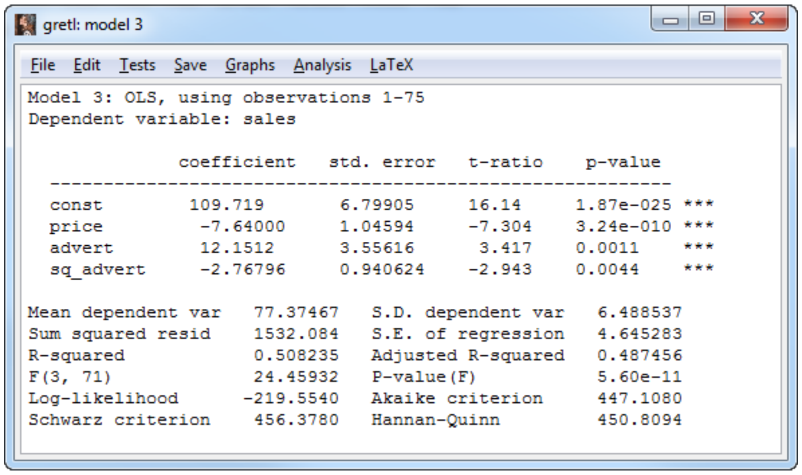
The script to estimate the models under H_0 and H_1 and to compute the test statistic is given below.

```
1 open "@gretldir\data\poe\andy.gdt"
2 square advert
3 ols sales const price advert sq_advert
4 scalar sseu = $ess
5 scalar unrest_df = $df
6 ols sales const price
7 scalar sser = $ess
8 scalar Fstat=((sser-sseu)/2)/(sseu/(unrest_df))
9 pvalue F 2 unrest_df Fstat
```

The first thing to notice is that a **gretl** function is used to create *advert*^{2}. In line 2 the `square` command will square any variable or variables that follow. In doing so, the string `sq_` is appended as a prefix to the original variable name, so that squared advertising (`advert`^{2}) becomes `sq_advert`.

Gretl refers to the sum of squared residuals (SSE) as the "error sum of squares" and it is retrieved from the regression results using the accessor `$ess` (i.e., in line 3 `scalar sseu = $ess`. In this case, the accessor `$ess` points to the error sum of squares computed in the regression that precedes it. You'll also want to save the degrees of freedom in the unrestricted model so that you can use it in the computation of the p-value for the F-statistic. In this case, the F-statistic has 2 known parameters (J=1 and $N-K$=unrest_df) that are used as arguments in the `pvalue` function.

There are a number of other ways within **gretl** to do this test. These are available through scripts, but it may be useful to demonstrate how to access them through the GUI. First, you'll want to estimate the model using least squares. From the pull-down menu (see Figure 5.1) select `Model>Ordinary Least Squares`, specify the unrestricted model (Figure 5.2), and run the regression. This yields the result shown in Figure 6.1.



```
Model 3: OLS, using observations 1-75
Dependent variable: sales

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const       109.719       6.79905      16.14     1.87e-025 ***
  price        -7.64000     1.04594      -7.304    3.24e-010 ***
  advert       12.1512      3.55616       3.417    0.0011    ***
  sq_advert    -2.76796     0.940624     -2.943    0.0044    ***

Mean dependent var    77.37467   S.D. dependent var    6.488537
Sum squared resid     1532.084   S.E. of regression    4.645283
R-squared             0.508235   Adjusted R-squared    0.487456
F(3, 71)              24.45932   P-value(F)            5.60e-11
Log-likelihood       -219.5540   Akaike criterion      447.1080
Schwarz criterion     456.3780   Hannan-Quinn          450.8094
```

Figure 6.1: The model results from least squares regression using the pull-down menu

You'll notice that along the menu bar at the top of this window there are a number of options that are available to you. Choose `Tests` and the pull-down menu shown in Figure 6.2 will be revealed. The first four options in 6.2 are highlighted and these are the ones that are most pertinent

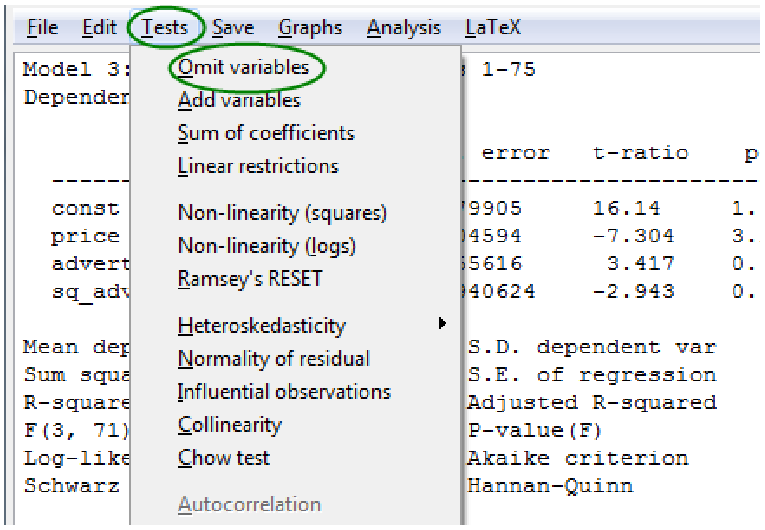
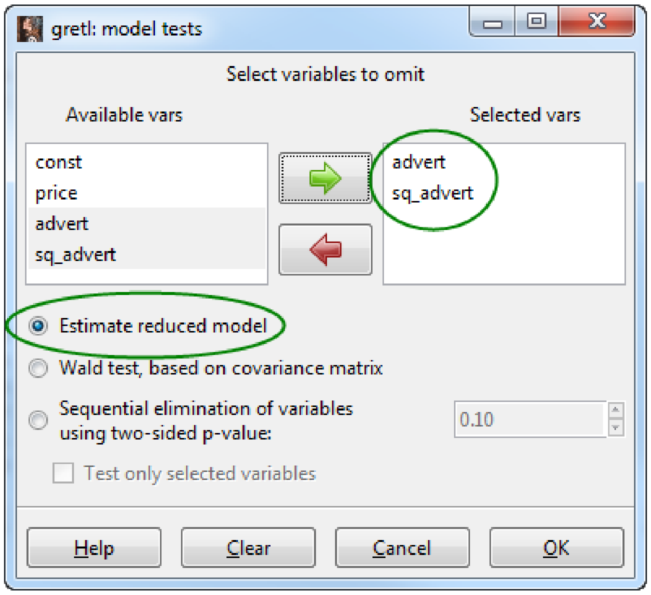


Figure 6.2: Choosing `Tests` from the pull-down menu of the model window reveals several testing options

to the discussion here. This menu provides you an easy way to omit variables from the model, add variables to the model, test a sum of your coefficients, or to test arbitrary linear restrictions on the parameters of your model.

Since this test involves imposing a zero restriction on the coefficient of the variable *price*, we can use the `Omit variables` option. This brings up the dialog box shown in Figure 6.3. Notice the two radio buttons at the bottom of the window. The first is labeled *Estimate reduced model* and this is the one you want to use to compute equation 6.2. If you select the other, no harm is done. It is computed in a different way, but produces the same answer in a linear model. The only advantage of the Wald test (second option) is that the restricted model does not have to be estimated in order to perform the test. Consequently, when you use the `--wald` option, the restricted model is not estimated and the unrestricted model remains in **gretl**'s memory where its statistics can be accessed.


gretl: model tests

Select variables to omit

Available vars: const, price, advert, sq_advert

Selected vars: advert, sq_advert

[✓ Estimate reduced model]

Wald test, based on covariance matrix

Sequential elimination of variables using two-sided p-value: 0.10

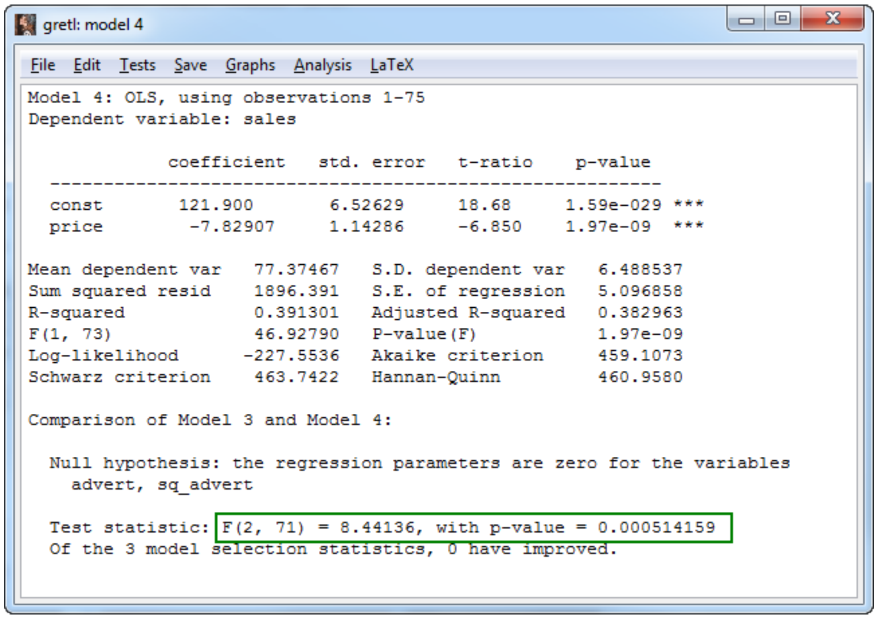
Test only selected variables

Figure 6.3: The `Omit variables` dialog box available from the `Tests` pull-down menu in the model window.

Select the variable P and click **OK** to reveal the result shown in Figure 6.4. The interesting thing about this option is that it mimics your manual calculation of the F statistic from the script. It computes the sum of squared errors in the unrestricted and restricted models and computes equation (6.2) based on those regressions. Most pieces of software choose the alternative method (Wald) to compute the test, but you get the same result.[1]

You can also use the `linear restrictions` option from the pull-down menu shown in Figure 6.2. This produces a large dialog box that requires a bit of explanation. The box appears in Figure 6.5. The restrictions you want to impose (or test) are entered here. Each restriction in the set should be expressed as an equation, with a linear combination of parameters on the left and a numeric value to the right of the equals sign. Parameters are referenced in the form `b[variable number]`, where `variable number` represents the position of the regressor in the equation, which starts with 1. This means that β_3 is equivalent to `b[3]`. Restricting $\beta_3 = 0$ is done by issuing `b[3]=0` and setting $\beta_4 = 0$ by `b[4]=0` in this dialog. Sometimes you'll want to use a restriction that involves a

[1]Of course, if you had used the `--robust` option with `ols`, then a Wald calculation is done. This is discussed in chapter 8.



```
Model 4: OLS, using observations 1-75
Dependent variable: sales

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const       121.900       6.52629     18.68     1.59e-029 ***
  price        -7.82907     1.14286     -6.850    1.97e-09  ***

Mean dependent var   77.37467   S.D. dependent var   6.488537
Sum squared resid    1896.391   S.E. of regression   5.096858
R-squared            0.391301   Adjusted R-squared   0.382963
F(1, 73)             46.92790   P-value(F)           1.97e-09
Log-likelihood      -227.5536   Akaike criterion     459.1073
Schwarz criterion    463.7422   Hannan-Quinn         460.9580

Comparison of Model 3 and Model 4:

  Null hypothesis: the regression parameters are zero for the variables
    advert, sq_advert

  Test statistic: F(2, 71) = 8.44136, with p-value = 0.000514159
  Of the 3 model selection statistics, 0 have improved.
```

Figure 6.4: The results using the `Omit variables` dialog box to test zero restrictions on the parameters of a linear model.

multiple of a parameter e.g., $3\beta_3 = 2$. The basic principle is to place the multiplier first, then the parameter, using * to multiply. So, in this case the restriction in **gretl** becomes `3*b[3] = 2`.

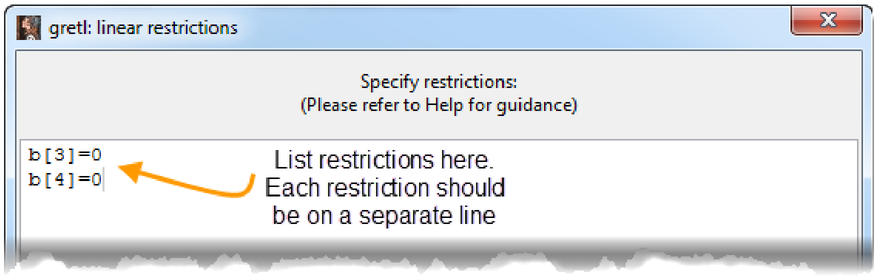
When you use the console or a script instead of the pull-down menu to impose restrictions, you'll have to tell **gretl** where the restrictions start and end. The restrictions start with a `restrict` statement and end with `end restrict`. The statement will look like this:

```
open "@gretldir\data\poe\andy.gdt"
ols sales const price advert sq_advert
restrict
    b[3] = 0
    b[4] = 0
end restrict
```

Put each restriction on its own line. Here is another example of a set of restrictions from a **gretl** script:

```
restrict
    b[1] = 0
    b[2] - b[3] = 0
    b[4] + 2*b[5] = 1
end restrict
```

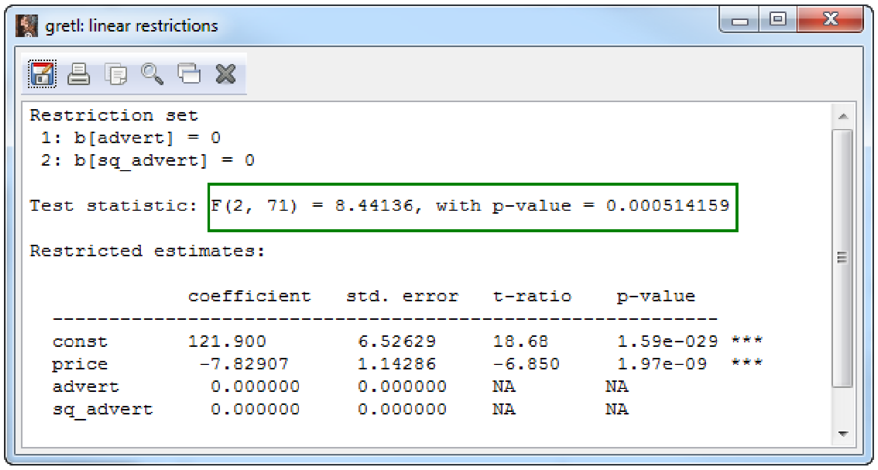
If you use the pull-down menu to impose these you can omit the `restrict` and `end restrict`



```
b[3]=0
b[4]=0
```

Figure 6.5: The linear restriction dialog box obtained using the `Linear restrictions` option in the `Tests` pull-down menu.

statements. The results you get from using the restrict statements appear in Figure 6.6. The test statistic and its p-value are highlighted in green. Notice also that the restricted estimates are printed; the coefficients on `advert` and `sq_advert` are zero.



```
Restriction set
 1: b[advert] = 0
 2: b[sq_advert] = 0

Test statistic: F(2, 71) = 8.44136, with p-value = 0.000514159

Restricted estimates:

              coefficient   std. error   t-ratio    p-value
  ----------------------------------------------------------
  const        121.900        6.52629     18.68      1.59e-029 ***
  price         -7.82907      1.14286     -6.850     1.97e-09  ***
  advert         0.000000     0.000000    NA         NA
  sq_advert      0.000000     0.000000    NA         NA
```

Figure 6.6: The results obtained from using the restrict dialog box.

6.2 Regression Significance

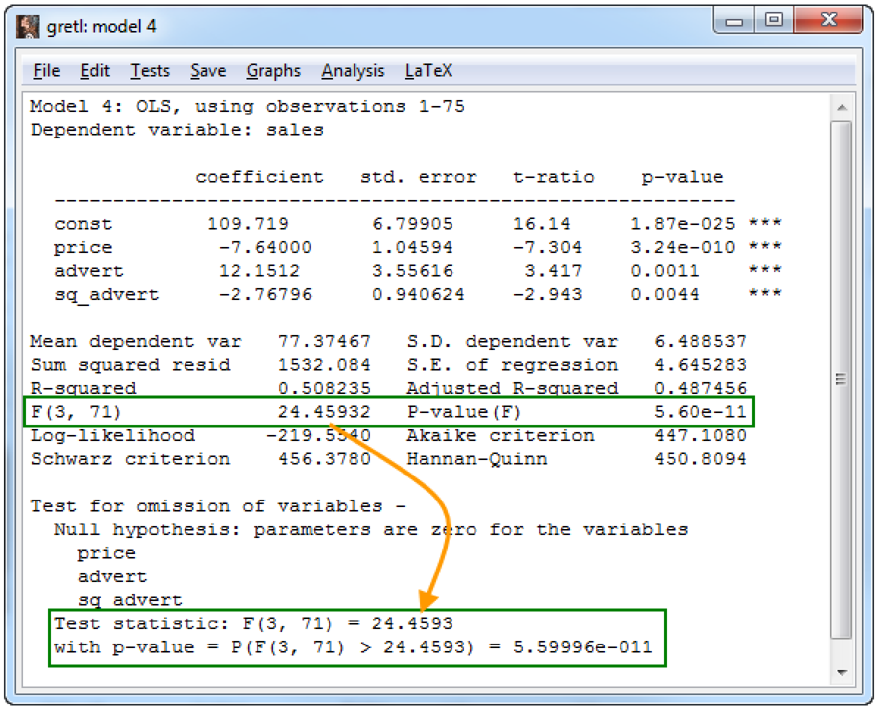
To statistically determine whether the regression is actually a model of the average behavior of your dependent variable, you can use the F-statistic. In this case, H_0 is the proposition that y does not depend on any of the independent variables, and H_1 is that it does.

$$\begin{aligned} H_o: &\quad \beta_1 + e_i \\ H_1: &\quad \beta_1 + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + e_i \end{aligned}$$

The null hypothesis can alternately be expressed as $\beta_2, \beta_3, \ldots, \beta_K = 0$, a set of $K-1$ linear restrictions. In Big Andy's Burger Barn the script is

```
open "@gretldir\data\poe\andy.gdt"
square advert
ols sales const price advert sq_advert
restrict
  b[2] = 0
  b[3] = 0
  b[4] = 0
end restrict
```

In lines 3-8 the model is estimated and the three slopes are restricted to be zero. The test result is shown in Figure 6.7 below. You can see that the F-statistic for this test is equal to 24.4593.



```
Model 4: OLS, using observations 1-75
Dependent variable: sales

              coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const        109.719       6.79905     16.14     1.87e-025 ***
  price         -7.64000     1.04594     -7.304    3.24e-010 ***
  advert        12.1512      3.55616      3.417    0.0011    ***
  sq_advert     -2.76796     0.940624    -2.943    0.0044    ***

Mean dependent var   77.37467   S.D. dependent var   6.488537
Sum squared resid    1532.084   S.E. of regression   4.645283
R-squared            0.508235   Adjusted R-squared   0.487456
F(3, 71)             24.45932   P-value(F)           5.60e-11
Log-likelihood      -219.5540   Akaike criterion     447.1080
Schwarz criterion    456.3780   Hannan-Quinn         450.8094

Test for omission of variables -
  Null hypothesis: parameters are zero for the variables
    price
    advert
    sq_advert
  Test statistic: F(3, 71) = 24.4593
  with p-value = P(F(3, 71) > 24.4593) = 5.59996e-011
```

Figure 6.7: The results obtained from using the restrict statements via the dialog box to conduct the overall F-test of regression significance.

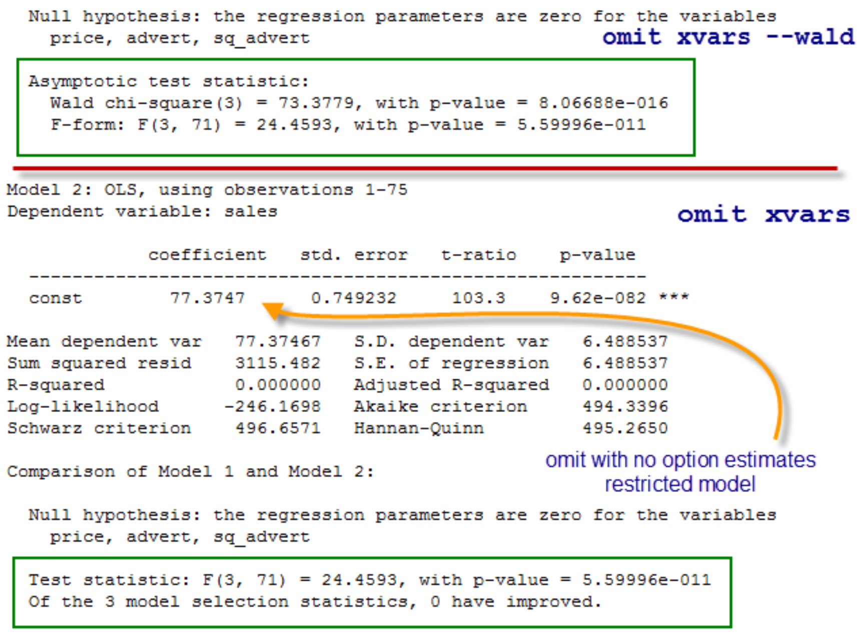
You should also notice that the same number appears in the regression results as $F(3, 71)$. This is not coincidental. The test of regression significance is important enough that it appears on the default output of every linear regression estimated using **gretl**. The statistic and its p-value are highlighted in Figure 6.7. Since the p-value is less than $= 0.05$, we reject the null hypothesis that the model is insignificant at the five percent level.

This is also a good opportunity to use the `omit` statement and to show the effect of the `--wald` option. Consider the script

```
open "@gretldir\data\poe\andy.gdt"
square advert
list xvars = price advert sq_advert
ols sales const xvars --quiet
omit xvars --wald
omit xvars
```

The regressors that carry slopes are collected into the list called `xvars`. Then, the overall F-test can be performed by simply omitting the `xvars` from the model. This tests the hypothesis that each coefficient is zero against the alternative that at least one is not. The `--wald` option will perform the test without imposing the restrictions. The chi-square form is actually very similar to the F-form; divide the chi-square form by its degrees of freedom and you will get the F. Their are slight differences in the χ^2_J/J and the $F_{J,N-K}$ distributions, which accounts for the small difference in the reported p-values.

The second `omit xvars` statement will then repeat the test, this time imposing the restrictions on the model. The output is shown if Figure 6.8. You can see that the F-form in the top portion



```
  Null hypothesis: the regression parameters are zero for the variables
    price, advert, sq_advert

  Asymptotic test statistic:
    Wald chi-square(3) = 73.3779, with p-value = 8.06688e-016
    F-form: F(3, 71) = 24.4593, with p-value = 5.59996e-011

Model 2: OLS, using observations 1-75
Dependent variable: sales

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const        77.3747      0.749232     103.3    9.62e-082 ***

Mean dependent var   77.37467   S.D. dependent var   6.488537
Sum squared resid    3115.482   S.E. of regression   6.488537
R-squared            0.000000   Adjusted R-squared   0.000000
Log-likelihood      -246.1698   Akaike criterion     494.3396
Schwarz criterion    496.6571   Hannan-Quinn         495.2650

Comparison of Model 1 and Model 2:

  Null hypothesis: the regression parameters are zero for the variables
    price, advert, sq_advert

  Test statistic: F(3, 71) = 24.4593, with p-value = 5.59996e-011
  Of the 3 model selection statistics, 0 have improved.
```

Figure 6.8: The results obtained from using the `omit` statements to conduct the overall F-test of regression significance.

of the output and the test statistic at the bottom match each other as well as the one obtained using `restrict`. No regression output follows the first version because of the `--wald` option. In the second instance, the model is restricted and the estimate of the constant (the series mean in this case) is given before printing the test result.

One can also perform the test manually using saved results from the estimated model. The script to do so is:

```
ols sales const price advert sq_advert
scalar sseu = $ess
scalar unrest_df = $df
ols sales const
scalar sser = $ess
scalar rest_df = $df

scalar J = rest_df - unrest_df
scalar Fstat=((sser-sseu)/J)/(sseu/(unrest_df))
pvalue F J unrest_df Fstat
```

Since there are three hypotheses to test jointly the numerator degrees of freedom for the F-statistic is $J = K - 1 = 3$. The saved residual degrees of freedom from the restricted model can be used to obtain the number of restrictions imposed. Each unique restriction in a linear model reduces the number of parameters in the model by one. So, imposing one restriction on a three parameter unrestricted model (e.g., Big Andy's), reduces the number of parameters in the restricted model to two. Let K_r be the number of regressors in the restricted model and K_u the number in the unrestricted model. Subtracting the degrees of freedom in the unrestricted model $(N - K_u)$ from those of the restricted model $(N - K_r)$ will yield the number of restrictions you've imposed, i.e., $(N - K_r) - (N - K_u) = (K_u - K_r) = J$.

6.2.1 Relationship Between *t*- and *F*-tests

You can certainly use an F-test to test the significance of individual variables in a regression. Consider once again the model for Big Andy

$$sales_i = \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 + e \tag{6.3}$$

and suppose we want to test whether *price* affects *sales*. Using the `omit` command produces the F-test

```
ols sales const price advert sq_advert
omit price
```

The output window is shown in Figure 6.7. The F(1, 71) statistic is equal to 53.3549 and has a p-value that is much smaller than 0.05; the coefficient is significant at the 5% level. Notice also that in the unrestricted model (Model 6 in the output) that the usual t-ratio is -7.304, also significant at 5%. The t-ratio has a t_{71} distribution if the coefficient is zero. Squaring $(-7.304)^2 = 53.3549$, suggesting that there is a relationship between these two statistics. In fact, t_n^2 is equivalent to $F(1, n)$. This hold for any degrees of freedom parameter, n.

6.2.2 Optimal Level of Advertising

The optimal level of advertising is that amount where the last dollar spent on advertising results in only 1 dollar of additional sales (we are assuming here that the marginal cost of producing and selling another burger is zero!). Find the level of level of advertising, $advert_o$, that solves:

$$\frac{\partial E[sales]}{\partial advert} = \beta_3 + 2\beta_4 advert_o = \$1 \tag{6.4}$$

Plugging in the least squares estimates from the model and solving for $advert_o$ can be done in **gretl**. A little algebra yields

$$advert_o = \frac{\$1 - \beta_3}{2\beta_4} \tag{6.5}$$

The script in **gretl** to compute this follows.

```
open "@gretldir\data\poe\andy.gdt"
square advert
ols sales const price advert sq_advert
scalar Ao =(1-$coeff(advert))/(2*$coeff(sq_advert))
```

which generates the result:

```
? scalar Ao =(1-$coeff(advert))/(2*$coeff(sq_advert))
  Generated scalar Ao (ID 7) = 2.01434
```

This implies that the optimal level of advertising is estimated to be approximately \$2014.

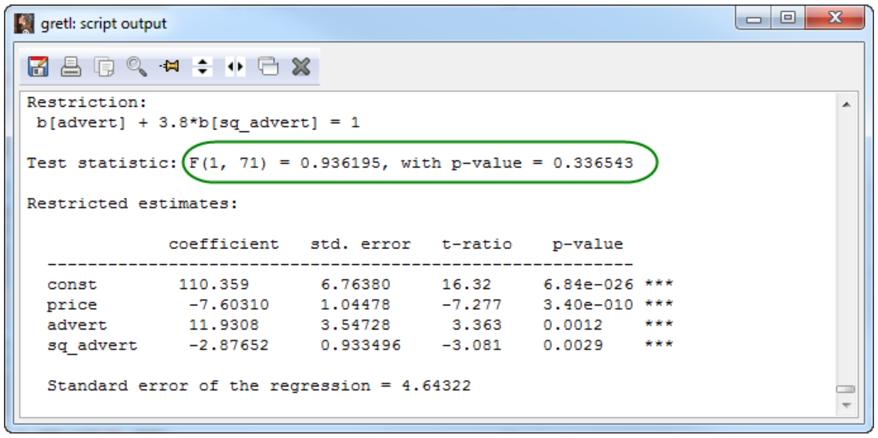
To test the hypothesis that \$1900 is optimal (remember, *advert* is measured in \$1000)

$$\begin{aligned} H_o: &\quad \beta_3 + 2\beta_4 1.9 = 1 \\ H_1: &\quad \beta_3 + 2\beta_4 1.9 \neq 1 \end{aligned}$$

you can use a t-test or an F-test. Following the regression, use

```
restrict
  b[3] + 3.8*b[4]=1
end restrict
```

Remember that `b[3]` refers to the coefficient of the third variable in the regression (`A`) and `b[4]` to the fourth. The output from the script is shown in Figure 6.9. The F-statistic is =0.936 and has a p-value of 0.33. We cannot reject the hypothesis that \$1900 is optimal at the 5% level.



```
Restriction:
 b[advert] + 3.8*b[sq_advert] = 1

Test statistic: F(1, 71) = 0.936195, with p-value = 0.336543

Restricted estimates:

             coefficient   std. error   t-ratio    p-value
  ------------------------------------------------------------
  const        110.359      6.76380      16.32     6.84e-026 ***
  price         -7.60310    1.04478      -7.277    3.40e-010 ***
  advert        11.9308     3.54728       3.363    0.0012    ***
  sq_advert     -2.87652    0.933496     -3.081    0.0029    ***

  Standard error of the regression = 4.64322
```

Figure 6.9: Testing whether $1900 in advertising is optimal using the `restrict` statement.

A one-tailed test would be a better option in this case. Andy decides he wants to test whether the optimal amount is greater than $1900.

$$H_0 : \beta_3 + 3.8\beta_4 \le 1$$
$$H_1 : \beta_3 + 3.8\beta_4 > 1$$

A one-sided alternative has to be tested using a t-ratio rather than the F-test. The script below computes such a test statistic much in the same way that we did in chapter 5.

```
1 # One-sided t-test
2 ols sales const price advert sq_advert --vcv
3 scalar r = $coeff(advert)+3.8*$coeff(sq_advert)-1
4 scalar v = $vcv[3,3]+((3.8)^2)*$vcv[4,4]+2*(3.8)*$vcv[3,4]
5 scalar t = r/sqrt(v)
6 pvalue t $df t
```

Notice that in line 3 we had to compute the variance of a linear combination of parameters. This was easily done in the script. The results are:

```
t(71): area to the right of 0.967572 = 0.168271
(two-tailed value = 0.336543; complement = 0.663457)
```

The t-ratio is .9676 and the area to the right is 0.168. Once again, this is larger than 5% and the hypothesis cannot be rejected at that level.

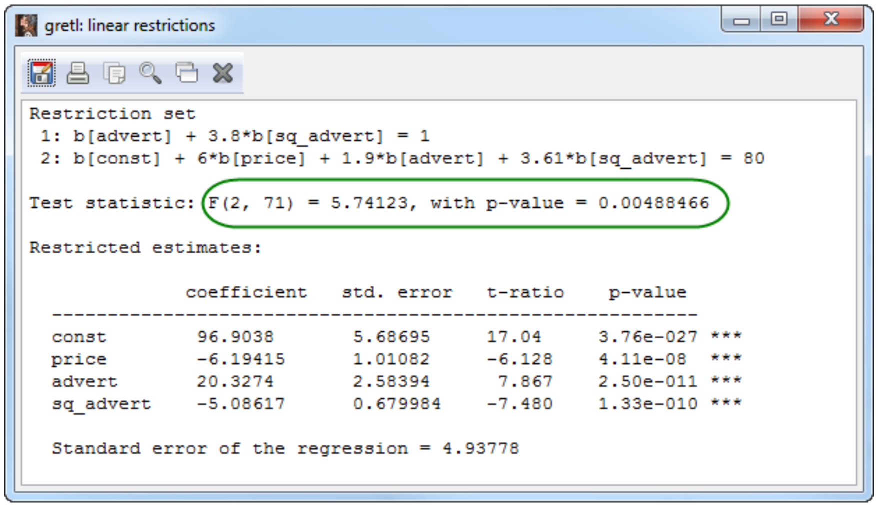
Finally, Big Andy makes another conjecture about sales. He is planning to charge $6 and use $1900 in advertising and expects sales to be $80,000. Combined with the optimality of $1900 in advertising leads to the following joint test:

$$H_0 : \beta_3 + 3.8\beta_4 = 1 \text{ and } \beta_1 + 6\beta_2 + 1.9\beta_3 + 1.9^2\beta_4 = 80$$
$$H_1 : \text{not } H_0$$

The model is estimated and the hypotheses tested:

```
ols sales const price advert sq_advert
restrict
    b[3]+3.8*b[4]=1
    b[1]+6*b[2]+1.9*b[3]+3.61*b[4]=80
end restrict
```

The result is shown in Figure 6.10 below. Andy is disappointed with this outcome. The null



```
Restriction set
 1: b[advert] + 3.8*b[sq_advert] = 1
 2: b[const] + 6*b[price] + 1.9*b[advert] + 3.61*b[sq_advert] = 80

Test statistic: F(2, 71) = 5.74123, with p-value = 0.00488466

Restricted estimates:

             coefficient   std. error   t-ratio    p-value
  ----------------------------------------------------------
  const        96.9038      5.68695     17.04     3.76e-027 ***
  price        -6.19415     1.01082     -6.128    4.11e-08  ***
  advert       20.3274      2.58394      7.867    2.50e-011 ***
  sq_advert    -5.08617     0.679984    -7.480    1.33e-010 ***

  Standard error of the regression = 4.93778
```

Figure 6.10: Andy muses about whether \$1900 in advertising is optimal and whether this will generate \$80000 in sales given price is \$6. It is not supported by the data.

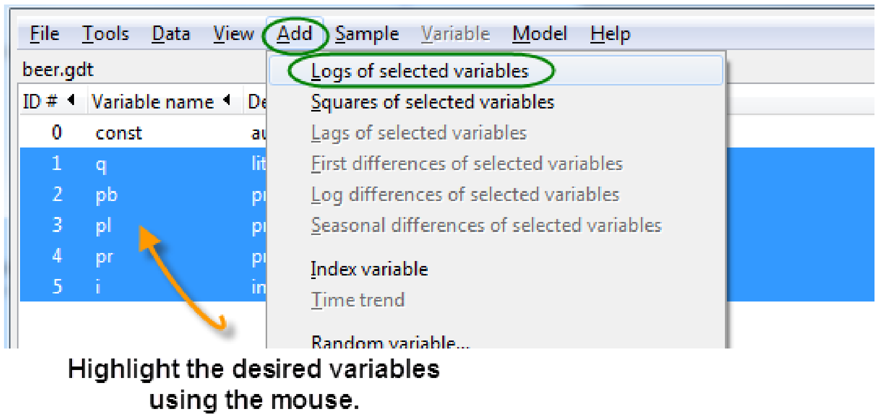
hypothesis is rejected since the p-value associated with the test is $0.0049 < .05$. Sorry Andy!

6.3 Nonsample Information

In this section we'll estimate a beer demand model. The data are in *beer.gdt* and are in level form. The model to be estimated is

$$\ln(q) = \beta_1 + \beta_2 \ln(pb) + \beta_3 \ln(pl) + \beta_4 \ln(pr) + \beta_5 \ln(i) + e \tag{6.6}$$

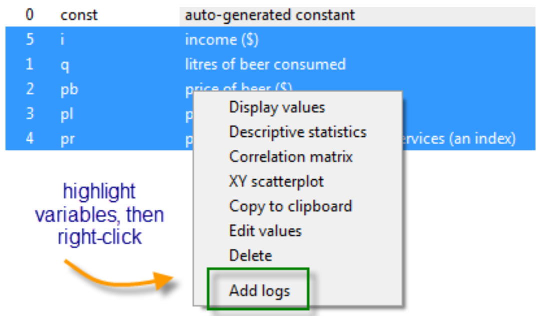
The first thing to do is to convert each of the variables into natural logs. **Gretl** has a built in function for this that is very slick. From the main window, highlight the variables you want to transform with the cursor. Then go to `Add>Logs of selected variables` from the pull-down menu as shown in Figure 6.11. This can also be done is a script or from the console using the


File
Tools
Data
View
Add
Sample
Variable
Model
Help
beer.gdt
ID #
Variable name
Logs of selected variables
Squares of selected variables
Lags of selected variables
First differences of selected variables
Log differences of selected variables
Seasonal differences of selected variables
Index variable
Time trend
Random variable...
Highlight the desired variables using the mouse.

ID #	Variable name
0	const
1	q
2	pb
3	pl
4	pr
5	i

Figure 6.11: Use the pull-down menu to add the natural logs of each variable

command `logs q pb pl pr i`. The natural log of each of the variables is obtained and the result stored in a new variable with the prefix `l_` ("el" underscore). An even easier way to add the logs is to highlight the variables and right-click the mouse. A pop-up menu appears and the `Add logs` option is available.



0	const	auto-generated constant
5	i	income ($)
1	q	litres of beer consumed
2	pb	price of beer ($)
3	pl	p
4	pr	p...rvices (an index)

A no money illusion restriction can be parameterized in this model as $\beta_2 + \beta_3 + \beta_4 + \beta_5 = 0$. This is easily estimated within **gretl** using the restrict dialog or a script as shown below.

```
open "@gretldir\data\poe\beer.gdt"
logs q pb pl pr i
ols l_q const l_pb l_pl l_pr l_i --quiet
restrict
    b2+b3+b4+b5=0
end restrict
```

```
Restriction:
 b[l_pb] + b[l_pl] + b[l_pr] + b[l_i] = 0

Test statistic: F(1, 25) = 2.49693, with p-value = 0.126639
Restricted estimates:

Restricted estimates:

             coefficient   std. error   t-ratio   p-value
  --------------------------------------------------------
  const       -4.79780     3.71390      -1.292    0.2078
  l_pb        -1.29939     0.165738     -7.840    2.58e-08 ***
  l_pl         0.186816    0.284383      0.6569   0.5170
  l_pr         0.166742    0.0770752     2.163    0.0399   **
  l_i          0.945829    0.427047      2.215    0.0357   **

  Standard error of the regression = 0.0616756
```

Figure 6.12: **gretl** output for the beer demand

The syntax for the restrictions is new. Instead of using `b[2]+b[3]+b[4]+b[5]=0`, a simpler form is used. This is undocumented in the **gretl** version I am using (1.9.5cvs) and I am uncertain of whether this will continue to work. It does for now and I've shown it here. Apparently **gretl** is able to correctly parse the variable number from the variable name without relying on the brackets. The output from the **gretl** script output window appears in Figure 6.12.

6.4 Model Specification

There are several issues of model specification explored here. First, it is possible to omit relevant independent variables from your model. A **relevant independent variable** is one that affects the mean of the dependent variable. When you omit a relevant variable that happens to be correlated with any of the other included regressors, least squares suffers from omitted variable bias.

The other possibility is to include **irrelevant variables** in the model. In this case, you include extra regressors that either don't affect y or, if they do, they are not correlated with any of the other regressors. Including irrelevant variables in the model makes least squares less precise than it otherwise would be–this increases standard errors, reduces the power of your hypothesis tests, and increases the size of your confidence intervals.

The example used in the text uses the dataset *edu_inc.gdt.* The first regression

$$faminc = \beta_1 + \beta_2 he + \beta_3 we + \beta_4 kl6 + \beta_5 x_{i5} + \beta_6 x_{i6} + e_i \tag{6.7}$$

where *faminc* is family income, *he* is husband's years of schooling, *we* is woman's years of schooling, and *kl6* are the number of children in the household under age 6. Several variations of this model are estimated. The first includes only *he*, another only *he* and *we*, and one includes the two irrelevant variables, x_5 and x_6. The **gretl** script to estimate these models and test the implied hypothesis restrictions follows. If you type this in yourself, omit the line numbers.

```
1  list all_x = const he we kl6 xtra_x5 xtra_x6
2  ols faminc all_x
3  modeltab add
4  omit xtra_x5 xtra_x6
5  modeltab add
6  omit kl6
7  modeltab add
8  omit we
9  modeltab add
10 modeltab show
```

The models can be estimated and saved as icons (`File>Save to session as icon`) within **gretl**. Once they've all been estimated and saved as icons, open a **session window** (Figure 1.12) and drag each model onto the *model table* icon. Click on the model table icon to reveal the output shown in Figure 6.13.

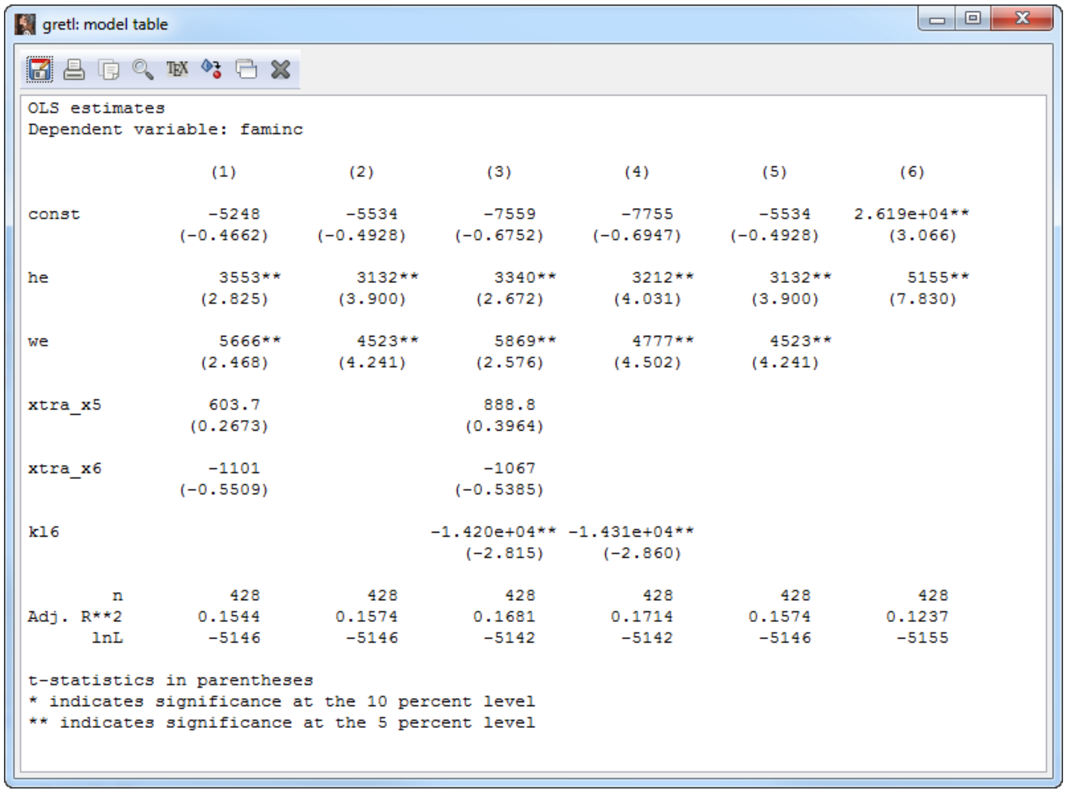
In the above script, we have used the `modeltab` function after each estimated model to add it to the model table. The final line tells **gretl** to display (`show`) the resulting model table.

One word of caution is in order about the given script and its interpretation. The `omit` statement tests the implied restriction (the coefficient on the omitted variable is zero) versus the estimated model that **immediately precedes it**. Thus, when we test that the coefficient on `kl6` is zero in line 6, the alternative model is the restricted model from line 4, which already excludes `xtra_x5`, and `xtra_x6`. Thus, only one restriction is being tested. If your intention is to test all of the restrictions (`omit xtra_x5`, `xtra_x6` and `kl6`) versus the the completely unrestricted model in line 2 that includes all of the variables, you'll need to modify your code. I'll leave this an an exercise.

6.5 Model Selection: Introduction to gretl Functions

Choosing an appropriate model is part art and part science. Omitting relevant variables that are correlated with regressors causes least squares to be biased and inconsistent. Including irrelevant variables reduces the precision of least squares. So, from a purely technical point, it is important to estimate a model that has all of the necessary relevant variables and none that are irrelevant. It is also important to use a suitable functional form. There is no set of mechanical rules that one can follow to ensure that the model is correctly specified, but there are a few things you can do to increase your chances of having a suitable model to use for decision-making.

Here are a few rules of thumb:



```
OLS estimates
Dependent variable: faminc

                  (1)           (2)           (3)           (4)           (5)           (6)

const            -5248         -5534         -7559         -7755         -5534       2.619e+04**
               (-0.4662)     (-0.4928)     (-0.6752)     (-0.6947)     (-0.4928)      (3.066)

he                3553**        3132**        3340**        3212**        3132**        5155**
                 (2.825)       (3.900)       (2.672)       (4.031)       (3.900)       (7.830)

we                5666**        4523**        5869**        4777**        4523**
                 (2.468)       (4.241)       (2.576)       (4.502)       (4.241)

xtra_x5           603.7                       888.8
                (0.2673)                     (0.3964)

xtra_x6          -1101                       -1067
               (-0.5509)                    (-0.5385)

kl6                                        -1.420e+04**  -1.431e+04**
                                             (-2.815)      (-2.860)

          n        428           428           428           428           428           428
Adj. R**2       0.1544        0.1574        0.1681        0.1714        0.1574        0.1237
        lnL      -5146         -5146         -5142         -5142         -5146         -5155

t-statistics in parentheses
* indicates significance at the 10 percent level
** indicates significance at the 5 percent level
```

Figure 6.13: Save each model as an icon. Open the session window and drag each model to the model table icon. Click on the model table icon to reveal this output.

1. Use whatever economic theory you have to select a functional form. For instance, if you are estimating a short-run production function then economic theory suggests that marginal returns to factors of production diminish. That means you should choose a functional form that permits this (e.g., log-log).

2. If the estimated coefficients have the wrong signs or unreasonable magnitudes, then you probably want to reevaluate either the functional form or whether relevant variables are omitted.

3. You can perform joint hypothesis tests to detect the inclusion of irrelevant sets of variables. Testing is not fool-proof since there is always positive probability that type 1 or type 2 error is being committed.

4. You can use model selection rules to find sets of regressors that are 'optimal' in terms of an estimated bias/precision trade-off.

5. Use a RESET test to detect possible misspecification of functional form.

In this section, I will give you some **gretl** commands to help with the last two: model selection and RESET.

In this section we consider three model selection rules: $\bar{R}^2$, AIC, and SC. I'm not necessarily recommending that these be used, since there are plenty of statistical problems caused by using the sample to both specify, estimate, and then test hypotheses in a model, but sometimes you have little other choice. Lag selection discussed later in this book is a reasonable application for these.

6.5.1 Adjusted R^2

The adjusted R^2 was introduced in chapter 5. The usual R^2 is 'adjusted' to impose a small penalty when a variable is added to the model. Adding a variable with any correlation to y always reduces SSE and increases the size of the usual R^2. With the adjusted version, the improvement in fit may be outweighed by the penalty and it could become smaller as variables are added. The formula is:

$$\bar{R}^2 = 1 - \frac{SSE/(N-K)}{SST/(N-1)} \tag{6.8}$$

This sometimes referred to as "R-bar squared," (i.e., $\bar{R}^2$) although in **gretl** it is called "adjusted R-squared." The biggest drawback of using $\bar{R}^2$ as a model selection rule is that the penalty it imposes for adding regressors is too small on average. It tends to lead to models that contain irrelevant variables. There are other model selection rules that impose larger penalties for adding regressors and two of these are considered below.

6.5.2 Information Criteria

The two model selection rules considered here are the Akaike Information Criterion (AIC) and the Schwarz Criterion (SC). The SC is sometimes called the Bayesian Information Criterion (BIC). Both are computed by default in **gretl** and included in the standard regression output. The values that **gretl** reports are based on maximizing a log-likelihood function (normal errors). There are other variants of these that have been suggested for use in linear regression and these are presented in the equations below:

$$AIC = \ln(SSE/N) + 2K/N \tag{6.9}$$

$$BIC = SC = \ln(SSE/N) + K\ln(N)/N \tag{6.10}$$

The rule is, compute AIC or SC for each model under consideration and choose the model that minimizes the desired criterion. The models should be evaluated using the same number of observations, i.e., for the same value of N. You can convert the ones **gretl** reports to the ones in (6.9) using a simple transformation; add $(1 + \ln(2\pi))$ and then multiply everything by N. Since sample size should be held constant when using model selection rules, you can see that the two different computations will lead to exactly the same model choice.

Since the functions have to be evaluated for each model estimated, it is worth writing a function in **gretl** that can be reused. The use of functions to perform repetitive computations makes programs shorter and reduced errors (unless your function is wrong, in which case every computation is incorrect!) In the next section, I will introduce you to **gretl** functions and offer one that will compute the three model selection rules discussed above.

6.5.3 A gretl Function to Produce Model Selection Rules

Gretl offers a mechanism for defining **functions**, which may be called via the command line, in the context of a script, or (if packaged appropriately via the programs graphical interface. The syntax for defining a function looks like this:

```
function return-type function-name (parameters)
   function body
end function
```

The opening line of a function definition contains these elements, in strict order:

1. The keyword `function`.
2. return-type, which states the type of value returned by the function, if any. This must be one of void (if the function does not return anything), scalar, series, matrix, list or string.
3. function-name, the unique identifier for the function. Names must start with a letter. They have a maximum length of 31 characters; if you type a longer name it will be truncated. Function names cannot contain spaces. You will get an error if you try to define a function having the same name as an existing **gretl** command. Also, be careful not to give any of your variables (scalars, matrices, etc.) the same name as one of your functions.
4. The functionss parameters, in the form of a comma-separated list enclosed in parentheses. This may be run into the function name, or separated by white space as shown.

The model selection function is designed to do two things. First, we want it to print values of the model selection rules for $\bar{R}^2$, AIC and SC. While we are at it we should also print how many regressors the model has (and their names) and the sample size. The second thing we want is to be able to send the computed statistics to a matrix. This will allow us to collect results from several candidates into a single table.

The basic structure of the model selection function is

```
function matrix modelsel (series y, list xvars)
    [some computations]
    [print results]
    [return results]
end function
```

As required, it starts with the keyword `function`. The next word, `matrix`, tells the function that a matrix will be returned as output. The next word is `modelsel`, which is the name that we are giving to our function. The `modelsel` function has two arguments that will be used as inputs. The first is a data `series` that we will refer to inside the body of the function as `y`. The second is a `list` that will be referred to as `xvars`. The inputs are separated by a comma and there are spaces between the list of inputs. Essentially what we are going to do is feed the function a dependent variable and a list of the independent variables as inputs. Inside the function a regression is estimated, the criteria are computed based on it, the statistics are printed to the screen, and collected into a matrix that will be returned. The resulting matrix is then available for further manipulation outside of the function.

```
1  function matrix modelsel (series y, list xvars)
2      ols y xvars --quiet
3      scalar sse = $ess
4      scalar N = $nobs
5      scalar K = nelem(xvars)
6      scalar aic = ln(sse/N)+2*K/N
7      scalar bic = ln(sse/N)+K*N/N
8      scalar rbar2 = 1-((1-$rsq)*(N-1)/$df)
9      matrix A = { K, N, aic, bic, rbar2 }
10     printf "\nRegressors: %s\n",varname(xvars)
11     printf "K = %d, N = %d, AIC = %.4f, SC = %.4f, and\
12 Adjusted R2 = %.4f\n", K, N, aic, bic, rbar2
13     return A
14 end function
```

In line 2 the function inputs `y` and the list `xvars` are used to estimate a linear model by least squares. The `--quiet` option is used to suppress the least squares output. In lines 3-5 the sum of squared errors, *SSE*, the number of observations, *N*, and the number of regressors, *K*, are put into scalars. In lines 6-8 the three criteria are computed. Line 9 puts various scalars into a matrix called `A`. Lines 10 and 11 sends the names of the regressors to the screen. Line 11 sends formatted output to the screen. Line 12 sends the matrix `A` as a return from the function. The last line closes the function.[2]

At this point, the function can be highlighted and run.

To use the function create a `list` that will include the desired independent variables (called `x` in this case). Then to use the function you will create a matrix called `a` that will include the output from `modelsel`.

```
list x = const he we xtra_x5 xtra_x6
matrix a = modelsel(faminc,x)
```

[2]To get the **gretl** value of AIC: `scalar aic_g = (1+ln(2*pi)+aic)*N`

The output is:

```
Regressors: const,he,we,kl6,xtra_x5,xtra_x6
K = 6, N = 428, AIC = 21.2191, SC = 27.1911, and Adjusted R2 = 0.1681
```

You can see that each of the regressor names is printed out on the first line of output. This is followed by the values of K, N, AIC, SC, and $\bar{R}^2$.

To put the function to use, consider the following script where we create four sets of variables and use the model selection rules to pick the desired model.

```
1  list x1 = const he
2  list x2 = const he we
3  list x3 = const he we kl6
4  list x4 = const he we xtra_x5 xtra_x6
5  matrix a = modelsel(faminc,x1)
6  matrix b = modelsel(faminc,x2)
7  matrix c = modelsel(faminc,x3)
8  matrix d = modelsel(faminc,x4)
9  matrix MS = a|b|c|d
10 colnames(MS,"K N AIC SC Adj_R2" )
11 printf "%10.5g",MS
12 function modelsel clear
```

In this example the model selection rules will be computed for four different models. Lines 1-4 construct the variable list for each of these. The next four lines run the model selection function for each set of variables. Each set of results is saved in a separate matrix (`a, b, c, d`). The `colnames` function is used to give each column of the matrix a meaningful name. Then, the `printf` statement prints the matrix. The last line removes the `modelsel` function from memory. This is not strictly necessary. If you make changes to your function, just recompile it. The biggest problem with function proliferation is that you may inadvertently try to give a variable the same name as one of your functions that is already in memory. If that occurs, clear the function or rename the variable.

The first part of the output prints the results from the individual calls to `modelsel`.

```
Regressors: const,he
K = 2, N = 428, AIC = 21.2618, SC = 21.2807, and Adjusted R2 = 0.1237

Regressors: const,he,we
K = 3, N = 428, AIC = 21.2250, SC = 21.2534, and Adjusted R2 = 0.1574

Regressors: const,he,we,kl6
K = 4, N = 428, AIC = 21.2106, SC = 21.2485, and Adjusted R2 = 0.1714
```

```
Regressors: const,he,we,xtra_x5,xtra_x6
K = 5, N = 428, AIC = 21.2331, SC = 21.2805, and Adjusted R2 = 0.1544
```

The last part prints the matrix `MS`.

```
K          N       AIC        SC     Adj_R2
2        428    21.262    21.281    0.12375
3        428    21.225    21.253    0.15735
4        428    21.211    21.248    0.17135
5        428    21.233    21.281    0.15443
```

In this example all three criteria select the same model: $K = 4$ and the regressors are `const`, `he`, `we`, `kl6`. This model minimized AIC and SC and maximizes the adjusted R^2.

Later in the book, this model selection function will be refined to make it more general.

6.5.4 RESET

The **RESET** test is used to assess the adequacy of your functional form. The null hypothesis is that your functional form is adequate. The alternative is that it is not. The test involves running a couple of regressions and computing an F-statistic.

Consider the model

$$y_i = \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} + e_i \tag{6.11}$$

and the hypothesis

$$\begin{aligned} H_0: &\quad E[y|x_{i2}, x_{i3}] = \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} \\ H_1: &\quad \text{not } H_0 \end{aligned}$$

Rejection of H_0 implies that the functional form is not supported by the data. To test this, first estimate (6.11) using least squares and save the predicted values, $\hat{y}_i$. Then square and cube $\hat{y}$ and add them back to the model as shown below:

$$\begin{aligned} y_i &= \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} + \gamma_1 \hat{y}_i^2 + e_i \\ y_i &= \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} + \gamma_1 \hat{y}_i^2 + \gamma_2 \hat{y}_i^3 + e_i \end{aligned}$$

The null hypotheses to test (against alternative, 'not H_0') are:

$$\begin{aligned} H_0: &\quad \gamma_1 = 0 \\ H_0: &\quad \gamma_1 = \gamma_2 = 0 \end{aligned}$$

Estimate the auxiliary models using least squares and test the significance of the parameters of $\hat{y}^2$ and/or $\hat{y}^3$. This is accomplished through the following script. Note, the `reset` command issued after the first regression computes the test associated with $H_0 : \gamma_1 = \gamma_2 = 0$. It is included here so that you can compare the 'canned' result with the one you compute using the two step procedure suggested above. The two results should match.

```
ols faminc x3 --quiet
reset --quiet
reset --quiet --squares-only
```

The results of the RESET for the family income equation is

```
RESET test for specification (squares and cubes)
Test statistic: F = 3.122581,
with p-value = P(F(2,422) > 3.12258) = 0.0451

RESET test for specification (squares only)
Test statistic: F = 5.690471,
with p-value = P(F(1,423) > 5.69047) = 0.0175
```

The adequacy of the functional form is rejected at the 5% level for both tests. It's back to the drawing board!

6.6 Cars Example

The data set *cars.gdt* is included in package of datasets that are distributed with this manual. In most cases it is a good idea to print summary statistics of any new dataset that you work with. This serves several purposes. First, if there is some problem with the dataset, the summary statistics may give you some indication. Is the sample size as expected? Are the means, minimums and maximums reasonable? If not, you'll need to do some investigative work. The other reason is important as well. By looking at the summary statistics you'll gain an idea of how the variables have been scaled. This is vitally important when it comes to making economic sense out of the results. Do the magnitudes of the coefficients make sense? It also puts you on the lookout for discrete variables, which also require some care in interpreting.

The `summary` command is used to get summary statistics. These include mean, minimum, maximum, standard deviation, the coefficient of variation, skewness and excess kurtosis. The `corr` command computes the simple correlations among your variables. These can be helpful in gaining an initial understanding of whether variables are highly collinear or not. Other measures are more useful, but it never hurts to look at the correlations. Either of these commands can be used with a variable list afterwards to limit the list of variables summarized of correlated.

Consider the cars example from *POE4*. The script is

```
open "c:\Program Files\gretl\data\poe\cars.gdt"
summary
corr
ols mpg const cyl eng wgt
vif
```

The summary statistics appear below:

Summary Statistics, using the observations 1–392

Variable	Mean	Median	Minimum	Maximum
mpg	23.4459	22.7500	9.00000	46.6000
cyl	5.47194	4.00000	3.00000	8.00000
eng	194.412	151.000	68.0000	455.000
wgt	2977.58	2803.50	1613.00	5140.00

Variable	Std. Dev.	C.V.	Skewness	Ex. kurtosis
mpg	7.80501	0.332894	0.455341	−0.524703
cyl	1.70578	0.311733	0.506163	−1.39570
eng	104.644	0.538259	0.698981	−0.783692
wgt	849.403	0.285266	0.517595	−0.814241

and the correlation matrix

Correlation coefficients, using the observations 1–392
5% critical value (two-tailed) = 0.0991 for n = 392

mpg	cyl	eng	wgt	
1.0000	−0.7776	−0.8051	−0.8322	mpg
	1.0000	0.9508	0.8975	cyl
		1.0000	0.9330	eng
			1.0000	wgt

The variables are quite highly correlated in the sample. For instance the correlation between weight and engine displacement is 0.933. Cars with big engines are heavy. What a surprise!

The regression results are:

OLS, using observations 1–392
Dependent variable: mpg

	Coefficient	Std. Error	t-ratio	p-value
const	44.3710	1.48069	29.9665	0.0000
cyl	−0.267797	0.413067	−0.6483	0.5172
eng	−0.0126740	0.00825007	−1.5362	0.1253
wgt	−0.00570788	0.000713919	−7.9951	0.0000

Mean dependent var	23.44592	S.D. dependent var	7.805007
Sum squared resid	7162.549	S.E. of regression	4.296531
R^2	0.699293	Adjusted R^2	0.696967
$F(3, 388)$	300.7635	P-value(F)	7.6e–101
Log-likelihood	−1125.674	Akaike criterion	2259.349
Schwarz criterion	2275.234	Hannan–Quinn	2265.644

The test of the individual significance of `cyl` and `eng` can be read from the table of regression results. Neither are significant at the 5% level. The joint test of their significance is performed using the omit statement. The F-statistic is 4.298 and has a p-value of 0.0142. The null hypothesis is rejected in favor of their joint significance.

The new statement that requires explanation is `vif`. **vif** stands for *variance inflation factor* and it is used as a collinearity diagnostic by many programs, including **gretl**. The *vif* is closely related to the statistic suggested by Hill et al. (2011) who suggest using the R^2 from auxiliary regressions to determine the extent to which each explanatory variable can be explained as linear functions of the others. They suggest regressing x_j on all of the other independent variables and comparing the R^2_j from this auxiliary regression to 10. If the R^2_j exceeds 10, then there is evidence of a collinearity problem.

The vif_j actually reports the same information, but in a less straightforward way. The *vif* associated with the j^{th} regressor is computed

$$vif_j = \frac{1}{1 - R^2_j} \tag{6.12}$$

which is, as you can see, simply a function of the R^2_j from the j^{th} regressor. Notice that when $R^2_j > .80$, the $vif_j > 10$. Thus, the rule of thumb for the two rules is actually the same. A vif_j greater than 10 is equivalent to an R^2 greater than .8 from the auxiliary regression.

The output from **gretl** is shown below:

```
Variance Inflation Factors

Minimum possible value = 1.0
Values > 10.0 may indicate a collinearity problem

            cyl   10.516
            eng   15.786
```

```
          wgt    7.789

VIF(j) = 1/(1 - R(j)^2), where R(j) is the multiple correlation coefficient
between variable j and the other independent variables

Properties of matrix X'X:

 1-norm = 4.0249836e+009
 Determinant = 6.6348526e+018
 Reciprocal condition number = 1.7766482e-009
```

Once again, the **gretl** output is very informative. It gives you the threshold for high collinearity ($vif_j) > 10$) and the relationship between vif_j and R^2_j. Clearly, these data are highly collinear. Two variance inflation factors above the threshold and the one associated with `wgt` is fairly large as well.

The variance inflation factors can be produced from the dialogs as well. Estimate your model then, in the model window, select `Tests>Collinearity` and the results will appear in **gretl**'s output.

6.7 Script

```
set echo off
# f-test
open "@gretldir\data\poe\andy.gdt"
square advert
ols sales const price advert sq_advert
scalar sseu = $ess
scalar unrest_df = $df
ols sales const price
scalar sser = $ess
scalar Fstat=((sser-sseu)/2)/(sseu/(unrest_df))
pvalue F 2 unrest_df Fstat

# f-test using omit
ols sales const price advert sq_advert
omit advert sq_advert

# f-test using restrict
ols sales const price advert sq_advert
restrict
b[3]=0
b[4]=0
end restrict

# overall f
open "@gretldir\data\poe\andy.gdt"
```

```
square advert
ols sales const price advert sq_advert
restrict
  b[2] = 0
  b[3] = 0
  b[4] = 0
end restrict

ols sales const price advert sq_advert
scalar sseu = $ess
scalar unrest_df = $df
ols sales const
scalar sser = $ess
scalar rest_df = $df

scalar J = rest_df - unrest_df
scalar Fstat=((sser-sseu)/J)/(sseu/(unrest_df))
pvalue F J unrest_df Fstat

# t-test
ols sales const price advert sq_advert
omit price

# optimal advertising
open "@gretldir\data\poe\andy.gdt"
square advert
ols sales const price advert sq_advert
scalar Ao =(1-$coeff(advert))/(2*$coeff(sq_advert))
# test of optimal advertising
restrict
b[3]+3.8*b[4]=1
end restrict

open "@gretldir\data\poe\andy.gdt"
square advert
ols sales const price advert sq_advert
scalar Ao =(1-$coeff(advert))/(2*$coeff(sq_advert))

# One-sided t-test
ols sales const price advert sq_advert --vcv
scalar r = $coeff(advert)+3.8*$coeff(sq_advert)-1
scalar v = $vcv[3,3]+((3.8)^2)*$vcv[4,4]+2*(3.8)*$vcv[3,4]
scalar t = r/sqrt(v)
pvalue t $df t

# joint test
ols sales const price advert sq_advert
restrict
    b[3]+3.8*b[4]=1
    b[1]+6*b[2]+1.9*b[3]+3.61*b[4]=80
end restrict
```

```

# restricted estimation
open "@gretldir\data\poe\beer.gdt"
logs q pb pl pr i
ols l_q const l_pb l_pl l_pr l_i --quiet
restrict
    b2+b3+b4+b5=0
end restrict
restrict
    b[2]+b[3]+b[4]+b[5]=0
end restrict

# model specification -- relevant and irrelevant vars
open "@gretldir\data\poe\edu_inc.gdt"
ols faminc const he we
omit we

corr

list all_x = const he we kl6 xtra_x5 xtra_x6
ols faminc all_x

# reset test
ols faminc const he we kl6
reset --quiet --squares-only
reset --quiet

# model selection rules and a function
function matrix modelsel (series y, list xvars)
    ols y xvars --quiet
    scalar sse = $ess
    scalar N = $nobs
    scalar K = nelem(xvars)
    scalar aic = ln(sse/N)+2*K/N
    scalar bic = ln(sse/N)+K*ln(N)/N
    scalar rbar2 = 1-((1-$rsq)*(N-1)/$df)
    matrix A = { K, N, aic, bic, rbar2}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "K = %d, N = %d, AIC = %.4f, SC = %.4f, and\
Adjusted R2 = %.4f\n", K, N, aic, bic, rbar2
    return A
end function

list x1 = const he
list x2 = const he we
list x3 = const he we kl6
list x4 = const he we xtra_x5 xtra_x6
matrix a = modelsel(faminc,x1)
matrix b = modelsel(faminc,x2)
matrix c = modelsel(faminc,x3)
matrix d = modelsel(faminc,x4)

matrix MS = a|b|c|d
colnames(MS,"K N AIC SC Adj_R2" )
printf "%10.5g",MS
function modelsel clear

ols faminc all_x
modeltab add
omit xtra_x5 xtra_x6
modeltab add
omit kl6
modeltab add
omit we
modeltab add
modeltab show


ols faminc x3 --quiet
reset

# collinearity
open "@gretldir\data\poe\cars.gdt"
summary
corr

ols mpg const cyl
ols mpg const cyl eng wgt --quiet
omit cyl
ols mpg const cyl eng wgt --quiet
omit eng
ols mpg const cyl eng wgt --quiet
omit eng cyl

# Auxiliary regressions for collinearity
# Check: r2 >.8 means severe collinearity
ols cyl const eng wgt
scalar r1 = $rsq
ols eng const wgt cyl
scalar r2 = $rsq
ols wgt const eng cyl
scalar r3 = $rsq
printf "R-squares for the auxillary regresions\nDependent Variable:\
 \n cylinders %3.3g\n engine displacement %3.3g\n weight %3.3g\n", r1, r2, r3

ols mpg const cyl eng wgt
vif
```

Chapter 7

Using Indicator Variables

In this chapter we will explore the use of indicator variables in regression analysis. The discussion will include how to create them, estimate models using them, and how to interpret results that include them in the model. Several applications will be discussed as well. These include using them to create interactions, regional indicators, and to perform Chow tests of regression equivalence across different categories. Finally, their use in linear probability estimation is discussed and their use in evaluating treatment effects and the differences-in-difference estimators that are used in their estimation.

7.1 Indicator Variables

Indicator variables allow us to construct models in which some or all of the parameters of a model can change for subsets of the sample. As discussed in chapter 2, an **indicator variable** basically indicates whether a certain condition is met. If it does the variable is equal to 1 and if not, it is 0. They are often referred to as dummy variables, and **gretl** uses this term in a utility that is used to create indicator variables.

The example used in this section is again based on the *utown.gdt* real estate data. First we will open the dataset and examine the data.

```
open "@gretldir\data\poe\utown.gdt"
smpl 1 8
print price sqft age utown pool fplace --byobs
smpl full
summary
```

The sample is limited to the first 8 observations in line 2. The two numbers that follow the `smpl` command indicate where the subsample begins and where it ends. Logical statements can be used as well to restrict the sample. Examples of this will be given later. In the current case, eight observations are enough to see that `price` and `sqft` are continuous, that `age` is discrete, and that `utown`, `pool`, and `fplace` are likely to be indicator variables. The `print` statement is used with the `--byobs` option so that the listed variables are printed in columns.

```
           price         sqft          age        utown         pool       fplace

1        205.452        23.46            6            0            0            1
2        185.328        20.03            5            0            0            1
3        248.422        27.77            6            0            0            0
4        154.690        20.17            1            0            0            0
5        221.801        26.45            0            0            0            1
6        199.119        21.56            6            0            0            1
7        272.134        29.91            9            0            0            1
8        250.631        27.98            0            0            0            1
```

The sample is restored to completeness, and the summary statistics are printed. These give an idea of the range and variability of `price`, `sqft` and `age`. The means tell us about the proportions of homes that are near the University and that have pools or fireplaces.

Summary Statistics, using the observations 1–1000

Variable	Mean	Median	Minimum	Maximum
price	247.656	245.833	134.316	345.197
sqft	25.2097	25.3600	20.0300	30.0000
age	9.39200	6.00000	0.000000	60.0000
utown	0.519000	1.00000	0.000000	1.00000
pool	0.204000	0.000000	0.000000	1.00000
fplace	0.518000	1.00000	0.000000	1.00000

Variable	Std. Dev.	C.V.	Skewness	Ex. kurtosis
price	42.1927	0.170368	0.0905617	−0.667432
sqft	2.91848	0.115768	−0.0928347	−1.18500
age	9.42673	1.00370	1.64752	3.01458
utown	0.499889	0.963177	−0.0760549	−1.99422
pool	0.403171	1.97633	1.46910	0.158242
fplace	0.499926	0.965108	−0.0720467	−1.99481

You can see that half of the houses in the sample are near the University (519/1000). It is also pretty clear that prices are measured in units of \$1000 and square feet in units of 100. The oldest house is 60 years old and there are some new ones in the sample (age=0). Minimums and maximums of 0 and 1, respectively usually mean that you have indicator variables. This confirms what we concluded by looking at the first few observations in the sample.

7.1.1 Creating indicator variables

It is easy to create indicator variables in **gretl** . Suppose that we want to create a dummy variable to indicate that a house is large. Large in this case means one that is larger than 2500 square feet.

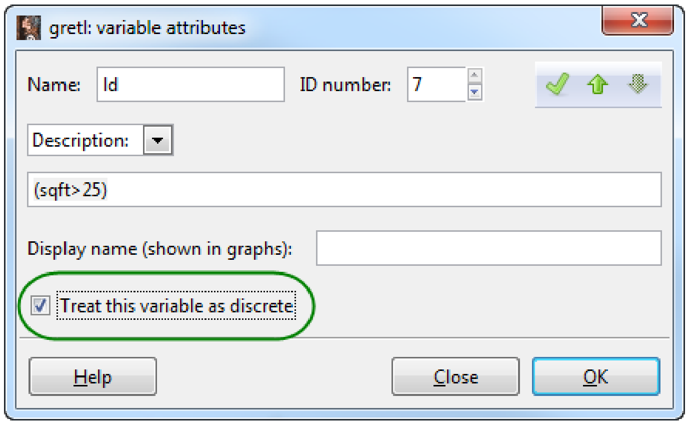
```
series ld = (sqft>25)
discrete ld
print ld sqft --byobs
```

The first line generates a variable called `ld` that takes the value 1 if the condition in parentheses is satisfied. It will be zero otherwise. The next line declares the variable to be discrete. Often this is unnecessary. "**Gretl** uses a simple heuristic to judge whether a given variable should be treated as discrete, but you also have the option of explicitly marking a variable as discrete, in which case the heuristic check is bypassed." (Cottrell and Lucchetti, 2011, p. 53) That is what we did here. Also from the **Gretl** Users Guide:

> To mark a variable as discrete you have two options.
>
> 1. From the graphical interface, select "Variable, Edit Attributes" from the menu. A dialog box will appear and, if the variable seems suitable, you will see a tick box labeled "Treat this variable as discrete". This dialog box [see Figure 7.1 below] can also be invoked via the context menu (right-click on a variable and choose Edit attributes) or by pressing the F2 key.
> 2. From the command-line interface, via the `discrete` command. The command takes one or more arguments, which can be either variables or list of variables.

So, the `discrete` declaration for `ld` in line 2 is not strictly necessary. Printing the indicator and square feet by observation reveals that the homes where $sqft > 25$ in fact are the same as those where $ld = 1$.

```
           ld         sqft
 1          0        23.46
 2          0        20.03
 3          1        27.77
 4          0        20.17
 5          1        26.45
 6          0        21.56
```



(sqft>25)

Display name (shown in graphs):

[✓ Treat this variable as discrete]

Figure 7.1: From the main **gretl** window, F2 brings up the variable attributes dialog. From here you can declare a variable to be discrete. The keyboard shortcut CRTL+e also initiates this dialog.

7.1.2 Estimating a Regression

The regression is also based on the University town real estate data. The regression is:

$$\begin{aligned} price &= \beta_1 + \delta_1 utown + \beta_2 sqft + \gamma(sqft \times utown) \\ &+ \beta_3 age + \delta_2 pool + \delta_3 fplace + e \end{aligned}$$

The estimated model is

OLS, using observations 1–1000
Dependent variable: price

	Coefficient	Std. Error	t-ratio	p-value
const	24.5000	6.19172	3.9569	0.0001
utown	27.4530	8.42258	3.2594	0.0012
sqft	7.61218	0.245176	31.0477	0.0000
sqft_utown	1.29940	0.332048	3.9133	0.0001
age	−0.190086	0.0512046	−3.7123	0.0002
pool	4.37716	1.19669	3.6577	0.0003
fplace	1.64918	0.971957	1.6968	0.0901

Mean dependent var	247.6557	S.D. dependent var	42.19273
Sum squared resid	230184.4	S.E. of regression	15.22521
R^2	0.870570	Adjusted R^2	0.869788
$F(6, 993)$	1113.183	P-value(F)	0.000000
Log-likelihood	−4138.379	Akaike criterion	8290.758
Schwarz criterion	8325.112	Hannan–Quinn	8303.815

The coefficient on the slope indicator variable *sqft* $\times$ *utown* is significantly different from zero at the 5% level. This means that size of a home near the university has a different impact on average home price. Based on the estimated model, the following conclusions are drawn:

- The location premium for lots near the university is \$27,453
- The change in expected price per additional square foot is \$89.12 near the university and \$76.12 elsewhere
- Homes depreciate \$190.10/year
- A pool is worth \$4,377.30
- A fireplace is worth \$1649.20

The script that generates these is:

```
scalar premium = $coeff(utown)*1000
scalar sq_u = 10*($coeff(sqft)+$coeff(sqft_utown))
scalar sq_other = 10*$coeff(sqft)
scalar depr = 1000*$coeff(age)
scalar sp = 1000*$coeff(pool)
scalar firep = 1000*$coeff(fplace)
printf "\n University Premium = $%8.7g\n\
Marginal effect of sqft near University = $%7.6g\n\
Marginal effect of sqft elsewhere = $%7.6g\n\
Depreciation Rate = $%7.2f\n\
Pool = $%7.2f\n\
Fireplace = $%7.2f\n",premium,sq_u,sq_other,depr,sp,firep
```

Notice that most of the coefficients was multiplied by 1000 since home prices are measured in \$1000 increments. Square feet are measured in increments of 100, therefore its marginal effect is multiplied by $1000/100 = 10$. It is very important to know the units in which the variables are recorded. This is the only way you can make ecnomic sense from your results.

7.2 Applying Indicator Variables

In this section a number of examples will be given about estimation and interpretation of regressions that include indicator variables.

7.2.1 Interactions

Consider the simple wage equation

$$\begin{aligned} wage = \beta_1 + \beta_2 educ + \delta_1 black + \delta_2 female \\ + \gamma(female \times black) + e \end{aligned}$$

where *black* and *female* are indicator variables. Taking the expected value of ln(*wage*) reveals each of the cases considered in the regression

$$E[wage] = \begin{cases} \beta_1 + \beta_2 educ & \textit{White, Males} \\ \beta_1 + \delta_1 + \beta_2 educ & \textit{Black, Males} \\ \beta_1 + \delta_2 + \beta_2 educ & \textit{White, Females} \\ \beta_1 + \delta_1 + \delta_2 + \gamma + \beta_2 educ & \textit{Black, Females} \end{cases} \tag{7.1}$$

The **reference group** is the one where all indicator variables are zero, i.e., white males. The parameter δ_1 measures the effect of being black, relative to the reference group; δ_2 measures the effect of being female relative to the reference group, and γ measures the effect of being both black and female.

The model is estimated using the *cps4_small.gdt* data which is from 2008. The results appear below:

Model 3: OLS, using observations 1–1000
Dependent variable: wage

	Coefficient	Std. Error	t-ratio	p-value
const	−5.28116	1.90047	−2.7789	0.0056
educ	2.07039	0.134878	15.3501	0.0000
black	−4.16908	1.77471	−2.3492	0.0190
female	−4.78461	0.773414	−6.1863	0.0000
blk_fem	3.84429	2.32765	1.6516	0.0989

Mean dependent var	20.61566	S.D. dependent var	12.83472
Sum squared resid	130194.7	S.E. of regression	11.43892
R^2	0.208858	Adjusted R^2	0.205677
$F(4, 995)$	65.66879	P-value(F)	2.53e–49
Log-likelihood	−3853.454	Akaike criterion	7716.908
Schwarz criterion	7741.447	Hannan–Quinn	7726.234

Holding the years of schooling constant, black males earn \$4.17/hour less than white males. For the same schooling, white females earn \$4.78 less, and black females earn \$5.15 less. The coefficient on the interaction term is not significant at the 5% level however.

A joint test of the hypothesis that $\delta_1 = \delta_2 = \gamma = 0$ is performed via the script

```
open "@gretldir\data\poe\cps4_small.gdt"
series blk_fem = black*female
ols wage const educ black female blk_fem
restrict
  b[3]=0
  b[4]=0
  b[5]=0
end restrict
```

and the result is

```
Restriction set
 1: b[black] = 0
 2: b[female] = 0
 3: b[blk_fem] = 0

Test statistic: F(3, 995) = 14.2059, with p-value = 4.53097e-009

Restricted estimates:

             coefficient   std. error   t-ratio    p-value
  ----------------------------------------------------------
  const       -6.71033      1.91416     -3.506     0.0005    ***
  educ         1.98029      0.136117    14.55      1.25e-043 ***
  black        0.000000     0.000000    NA        NA
  female       0.000000     0.000000    NA        NA
  blk_fem      0.000000     0.000000    NA        NA

  Standard error of the regression = 11.6638
```

The F-statistic is 14.21 and has a p-value less than 5%. The null hypothesis is rejected. At least one of the coefficients is nonzero. The test could be done even more easily using the `omit` statement after the regression since each of the coefficients in the linear restrictions is equal to zero.

7.2.2 Regional indicators

In this example a set of regional indicator variables is added to the model. There are four mutually exclusive regions to consider. A reference group must be chosen, in this case for the northeast. The model becomes:

$$wage = \beta_1 + \beta_2 educ + \delta_1 south + \delta_2 midwest + \delta_3 west + e$$

where *black* and *female* are indicator variables. Taking the expected value of $\ln(wage)$ reveals each of the cases considered in the regression

$$E[wage] = \begin{cases} \beta_1 + \beta_2 educ & \text{Northeast} \\ \beta_1 + \delta_1 + \beta_2 educ & \text{South} \\ \beta_1 + \delta_2 + \beta_2 educ & \text{Midwest} \\ \beta_1 + \delta_3 + \beta_2 educ & \text{West} \end{cases} \tag{7.2}$$

Once again, the omitted case (Northeast) becomes the reference group.

The regional dummy variables are added to the wage model for black females and is estimated by least squares. The regional indicator variables are tested jointly for significance using the `omit` statement.

```
ols wage const educ black female blk_fem south midwest west
omit south midwest west
series sser = $ess
```

The results appear below:

Model 4: OLS, using observations 1–1000
Dependent variable: wage

	Coefficient	Std. Error	t-ratio	p-value
const	−5.28116	1.90047	−2.7789	0.0056
educ	2.07039	0.134878	15.3501	0.0000
black	−4.16908	1.77471	−2.3492	0.0190
female	−4.78461	0.773414	−6.1863	0.0000
blk_fem	3.84429	2.32765	1.6516	0.0989

Mean dependent var	20.61566	S.D. dependent var	12.83472
Sum squared resid	130194.7	S.E. of regression	11.43892
R^2	0.208858	Adjusted R^2	0.205677
$F(4, 995)$	65.66879	P-value(F)	2.53e–49
Log-likelihood	−3853.454	Akaike criterion	7716.908
Schwarz criterion	7741.447	Hannan–Quinn	7726.234

First, notice that the sum-of-squared errors has been saved for future use. The only regional indicator that is individually significant at 5% is *midwest.* The joint test results are

```
Comparison of Model 3 and Model 4:
```

```
Null hypothesis: the regression parameters are zero for the variables
  south, midwest, west

Test statistic: F(3, 992) = 4.24557, with p-value = 0.00542762
Of the 3 model selection statistics, 1 has improved.
```

The test statistic has an $F(3, 992)$ distribution under the null and is equal to 4.25. The p-value is less than 5% and we conclude that the indicators are jointly significant.

7.2.3 Testing Equivalence of Two Regions

The question arises, is the wage equation different for the south than for the rest of the country? There are two ways to do this in **gretl**. One is very easy and the other not so easy, but makes for a useful example of how to use loops to create interactions among variables.

A Chow test is used to test for structural breaks or changes in a regression. In other words, one subsample has different intercept and slopes than another. It can be used to detect structural breaks in time-series models or to determine whether, in our case, the south's wages are determined differently from those in the rest of the country. The easy method uses **gretl**'s built-in `chow` command to test for a change in the regression. It must follow a regression and you must specify the indicator variable that identifies the two subsets.

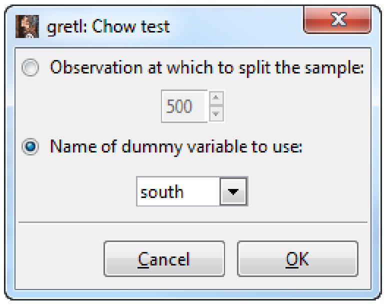
To illustrate its use, consider the basic wage model

$$\begin{aligned} wage = \beta_1 + \beta_2 educ + \delta_1 black + \delta_2 female \\ + \gamma(black \times female) + e \end{aligned}$$

Now, if wages are determined differently in the south, then the slopes and intercept for southerners will be different. The null hypothesis is that the coefficients of the two subsets are equal and the alternative is that they are not. The **gretl** commands to perform the test are:

```
1 list x = const educ black female blk_fem
2 ols wage x
3 chow south --dummy
```

Since the regressors are going to be used again below, I put them into a `list` to simplify things later. Line 2 estimates the model using least squares. Line 3 contains the test command. It is initiated by `chow` followed by the indicator variable that is used to define the subsets, in this case `south`. The option is used to tell **gretl** that `south` is an indicator. When the `--dummy` option is used, `chow` tests the null hypothesis of structural homogeneity with respect to that dummy. Essentially, **gretl** is creating interaction terms between the indicator and each of the regressors and adding them to the model. We will replicate this below in a script.



Observation at which to split the sample: 500
Name of dummy variable to use: south

Figure 7.2: Click `Tests>Chow test` from a model window to reveal the dialog box for the Chow test. Select an indicator variable or a break point for the sample.

The dialog box to perform the Chow test is found in the model window. After estimating the regression via the GUI the **model window** appears. Click `Tests>Chow test` on its menu bar to open the dialog box in Figure 7.2. The results from the test appear below.

```
Augmented regression for Chow test
OLS, using observations 1-1000
Dependent variable: wage

               coefficient   std. error   t-ratio    p-value
  ----------------------------------------------------------
  const         -6.60557      2.33663     -2.827     0.0048    ***
  educ           2.17255      0.166464    13.05      4.89e-036 ***
  black         -5.08936      2.64306     -1.926     0.0544    *
  female        -5.00508      0.899007    -5.567     3.33e-08  ***
  blk_fem        5.30557      3.49727      1.517     0.1296
  south          3.94391      4.04845      0.9742    0.3302
  so_educ       -0.308541     0.285734    -1.080     0.2805
  so_black       1.70440      3.63333      0.4691    0.6391
  so_female      0.901120     1.77266      0.5083    0.6113
  so_blk_fem    -2.93583      4.78765     -0.6132    0.5399

Mean dependent var   20.61566   S.D. dependent var   12.83472
Sum squared resid    129984.4   S.E. of regression   11.45851
R-squared            0.210135   Adjusted R-squared   0.202955
F(9, 990)            29.26437   P-value(F)           2.00e-45
Log-likelihood      -3852.646   Akaike criterion     7725.292
Schwarz criterion    7774.369   Hannan-Quinn         7743.944

Chow test for structural difference with respect to south
  F(5, 990) = 0.320278 with p-value 0.9009
```

Notice that the p-value associated with the test is 0.901, thus providing insufficient evidence to convince us that wages are structurally different in the south.

The other way to do this uses a loop to manually construct the interactions. Though the `chow` command makes this unnecessary, it is a great exercise that demonstrates how to create more general interactions among variables. The variable `south` will be interacted with each variable in a list and then added to a new list. The script is:

```
1 list x = const educ black female blk_fem
2 list dx = null
3 loop foreach i x
4   series south_$i = south * $i
5   list dx = dx south_$i
6 endloop
```

The first line includes each of the variables in the model that are to be interacted with `south`. The statement `list dx = null` creates a new list called `dx` that is empty (i.e., `= null`). In line 3 a `foreach` loop is initiated using the index `i` and it will increment through each element contained in the list, `x`. Line 4 creates a new series named `south_`*`varname`* that is constructed by interacting `south` with each variable in `x`. This is added to the new list, `dx` and the loop is closed.

To make it clear, let's go through a couple of iterations of the loop:

```
i=1
     column 1 of x = const
     series south_const = south * const
     dx = dx south_const
         implies dx = null south_const
         so, dx = south_const
loop ends--increment i
i=2
      column 2 of x = educ
      series south_educ = south * educ
      dx = dx south_educ
          so, dx = south_const south_educ
loop ends--increment i
i=3
      column 3 of x = black
      series south_black = south * black
      dx = dx south_black
           so, dx = south_const south_educ south_black
loop ends--increment i
i=4
and so on ...
```

The interactions are created and a series of regressions are estimated and put into a model table. The remaining script is:

```
1  modeltab clear
2  ols wage x dx
3  scalar sseu=$ess
4  scalar dfu = $df
5  modeltab add
6
7  smpl south=1 --restrict
8  ols wage x
9  modeltab add
10 smpl full
11
12 smpl south=0 --restrict
13 ols wage x
14 modeltab add
15 smpl full
16 modeltab show
```

Notice that the `smpl` command is used to manipulate subsets. It is restricted to observations in the `south` in line 7, restored to `full` in line 10, and then restricted to nonsouth observations in line 12. Also, the sum of squared errors from the unrestricted model is saved. These will be used to manually construct a Chow test below.

The model table appears below

OLS estimates
Dependent variable: wage

	(1)	(2)	(3)
const	-6.606^{**}	-2.662	-6.606^{**}
	(2.337)	(3.420)	(2.302)
educ	2.173^{**}	1.864^{**}	2.173^{**}
	(0.1665)	(0.2403)	(0.1640)
black	-5.089^{*}	-3.385	-5.089^{*}
	(2.643)	(2.579)	(2.604)
female	-5.005^{**}	-4.104^{**}	-5.005^{**}
	(0.8990)	(1.581)	(0.8857)
blk_fem	5.306	2.370	5.306
	(3.497)	(3.383)	(3.446)
south_const	3.944		
	(4.048)		
south_educ	-0.3085		
	(0.2857)		
south_black	1.704		
	(3.633)		

south_female	0.9011		
	(1.773)		
south_blk_fem	−2.936		
	(4.788)		
n	1000	296	704
$\bar{R}^2$	0.2030	0.1730	0.2170
ℓ	−3853	−1149	−2703

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

The first column contains the results from the model with all of the interactions. The second column is for workers in the south, and the third is for workers elsewhere.

The code to perform the Chow test uses the sum-of-squared errors and degrees of freedom that were saved in the unrestricted estimation and computes an F-statistic using that from the restricted regression.

```
smpl full
ols wage x
scalar sser = $ess
scalar fstat = ((sser-sseu)/5)/(sseu/dfu)
pvalue f 5 dfu fstat
```

Be sure to restore the `full` sample before estimating the restricted model. The restricted regression pools observations from the entire country together and estimates them with common coefficients. It is restricted because the parameters are the same in both subsets.

```
F(5, 990): area to the right of 0.320278 = 0.900945
(to the left: 0.0990553)
```

These results match those from the built-in version of the test.

7.2.4 Log-Linear Models with Indicators

In this example an indicator variable is included in a log-linear model. It is based on a wage example used earlier.

$$\ln(wage) = \beta_1 + \beta_2 educ + \delta female + e \tag{7.3}$$

Estimation of this model by least squares allows one to compute percentage differences between the wages of females and males. As discussed in *POE4*, the algebra suggests that the percentage difference is

$$100(e^{\hat{\delta}-1})\% \tag{7.4}$$

The model is estimated and the computation carried out in the following script.

```
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ female
scalar differential = 100*(exp($coeff(female))-1)
```

The natural logarithm of `wage` is taken in line 2. Then the model is estimated an the percentage difference computes.

OLS, using observations 1–1000
Dependent variable: l_wage

	Coefficient	Std. Error	t-ratio	p-value
const	1.6539	0.0844	19.60	1.3e-072
educ	0.0962	0.0060	15.94	3.76e-051
female	−0.2432	0.0327	−7.43	2.31e-013

Sum squared resid	262.2387	S.E. of regression	0.512862
R^2	0.221337	Adjusted R^2	0.219775
$F(2, 997)$	141.7000	P-value(F)	6.88e–55

The computed difference is −21.5896, suggesting that females earn about 21.59% less than males who have comparable levels of education.

7.3 Linear Probability

A linear probability model is a linear regression in which the dependent variable is an indicator variable. The model is estimated by least squares.

Suppose that

$$y_i = \begin{cases} 1 & \text{if alternative is chosen} \\ 0 & \text{if alternative is not chosen} \end{cases} \tag{7.5}$$

Suppose further that the $Pr(y_i = 1) = \pi_i$. For a discrete variable

$$E[y_i] = 1 \times Pr(y_i = 1) + 0 \times Pr(y_i = 0) = \pi_i \tag{7.6}$$

Thus, the mean of a binary random variable can be interpreted as a probability; it is the probability that $y = 1$. When the regression $E[y_i|x_{i2}, x_{i3}, \ldots, x_{iK}]$ is linear then $E[y_i] = \beta_1 + \beta_2 x_{i2} + \ldots + \beta_K x_{iK}$ and the mean (probability) is modeled linearly.

$$E[y_i|x_{i2}, x_{i3}, \ldots, x_{iK}] = \pi_i = \beta_1 + \beta_2 x_{i2} + \ldots + \beta_K x_{iK} \tag{7.7}$$

The variance of a binanry random variable is

$$var[y_i] = \pi_i(1 - \pi_i) \tag{7.8}$$

which means that it will be different for each individual. Replacing the unobserved probability, $E(y_i)$, with the observed indicator variable requires adding an error to the model that we can estimate via least squares. In this following example we have 1140 observations from individuals who purchased Coke or Pepsi. The dependent variable takes the value of 1 if the person buys Coke and 0 if Pepsi. These depend on the ratio of the prices, `pratio`, and two indicator variables, `disp_coke` and `disp_pepsi`. These indicate whether the store selling the drinks had promotional displays of Coke or Pepsi at the time of purchase.

OLS, using observations 1–1140
Dependent variable: coke
Heteroskedasticity-robust standard errors, variant HC3

	Coefficient	Std. Error	t-ratio	p-value
const	0.8902	0.0656	13.56	5.88e-039
pratio	−0.4009	0.0607	−6.60	6.26e-011
disp_coke	0.0772	0.0340	2.27	0.0235
disp_pepsi	−0.1657	0.0345	−4.81	1.74e-006

Sum squared resid	248.0043	S.E. of regression	0.467240
R^2	0.120059	Adjusted R^2	0.117736
$F(3, 1136)$	56.55236	P-value(F)	4.50e–34

The model was estimated using a variance-covariance matrix estimator that is consistent when the error terms of the model have variances that depend on the observation. That is the case here. I'll defer discussion of this issue until the next chapter when it will be discussed at some length.

7.4 Treatment Effects

In order to understand the measurement of treatment effects, consider a simple regression model in which the explanatory variable is a dummy variable, indicating whether a particular individual is in the treatment or control group. Let y be the outcome variable, the measured characteristic the treatment is designed to affect. Define the indicator variable d as

$$d_i = \begin{cases} 1 & \text{if treated} \\ 0 & \text{if not treated} \end{cases} \tag{7.9}$$

The effect of the treatment on the outcome can be modeled as

$$y_i = \beta_1 + \beta_2 d_i + e_i \quad i = 1, 2, \ldots, N \tag{7.10}$$

where e_i represents the collection of other factors affecting the outcome. The regression functions for the treatment and control groups are

$$E(y_i) = \begin{cases} \beta_1 + \beta_2 & \text{if individual is treated} \\ \beta_1 & \text{if not treated} \end{cases} \tag{7.11}$$

The treatment effect that we want to measure is β_2. The least squares estimator of β_2 is

$$b_2 = \frac{\sum_{i=1}^{N}(d_i - \bar{d})(y_i - \bar{y})}{\sum_{i=1}^{N}(d_i - \bar{d})^2} = \bar{y}_1 - \bar{y}_0 \tag{7.12}$$

where $\bar{y}_1$ is the sample mean for the observations on y for the treatment group and $\bar{y}_0$ is the sample mean for the observations on y for the untreated group. In this treatment/control framework the estimator b_2 is called the **difference estimator** because it is the difference between the sample means of the treatment and control groups.

To illustrate, we use the data from project STAR described in *POE4*, chapter 7.5.3.

The first thing to do is to take a look at the descriptive statistics for a subset of the variables. The list `v` is created to hold the variable names of all the variables of interest. Then the `summary` command is issued for the variables in `v` with the `--by` option. This option takes an argument, which is the name of a discrete variable by which the subsets are determined. Here, `small` and `regular` are binary, taking the value of 1 for small classes and 0 otherwise. This will lead to two sets of summary statistics.

```
open "@gretldir\data\poe\star.gdt"
list v = totalscore small tchexper boy freelunch white_asian \
         tchwhite tchmasters schurban schrural
summary v --by=small --simple
summary v --by=regular --simple
```

Here is a partial listing of the output:

```
regular = 1 (n = 2005):

                        Mean         Minimum         Maximum       Std. Dev.
```

```
totalscore          918.04         635.00         1229.0         73.138
small               0.00000        0.00000        0.00000        0.00000
tchexper            9.0683         0.00000        24.000         5.7244
boy                 0.51322        0.00000        1.0000         0.49995
freelunch           0.47382        0.00000        1.0000         0.49944
white_asian         0.68130        0.00000        1.0000         0.46609
tchwhite            0.79800        0.00000        1.0000         0.40159
tchmasters          0.36509        0.00000        1.0000         0.48157
schurban            0.30125        0.00000        1.0000         0.45891
schrural            0.49975        0.00000        1.0000         0.50012

small = 1 (n = 1738):

                       Mean        Minimum        Maximum      Std. Dev.
totalscore          931.94         747.00         1253.0         76.359
small               1.0000         1.0000         1.0000         0.00000
tchexper            8.9954         0.00000        27.000         5.7316
boy                 0.51496        0.00000        1.0000         0.49992
freelunch           0.47181        0.00000        1.0000         0.49935
white_asian         0.68470        0.00000        1.0000         0.46477
tchwhite            0.86249        0.00000        1.0000         0.34449
tchmasters          0.31761        0.00000        1.0000         0.46568
schurban            0.30610        0.00000        1.0000         0.46100
schrural            0.46260        0.00000        1.0000         0.49874
```

The `--simple` option drops the median, C.V., skewness and excess kurtosis from the summary statistics. In this case we don't need those so the option is used.

Next, we want to drop the observations for those classrooms that have a teacher's aide and to construct a set of variable lists to be used in the regressions that follow.

```
smpl aide != 1 --restrict
list x1 = const small
list x2 = x1 tchexper
list x3 = x2 boy freelunch white_asian
list x4 = x3 tchwhite tchmasters schurban schrural
```

In the first line the `smpl` command is used to limit the sample (`--restrict`) to those observations for which the `aide` variable is not equal (`!=`) to one. The `list` commands are interesting. Notice that `x1` is constructed in a conventional way using `list`; to the right of the equality is the name of two variables. Then `x2` is created with the first elements consisting of the list, `x1` followed by the additional variable `tchexper`. Thus, `x2` contains `const`, `small`, and `tchexper`. The lists `x3` and `x4` are constructed similarly. New variables are appended to previously defined lists. It's quite seamless and natural.

Now each of the models is estimated with the `--quiet` option and put into a model table.

OLS estimates
Dependent variable: totalscore

	(1)	(2)	(3)	(4)
const	918.0^{**}	907.6^{**}	927.6^{**}	936.0^{**}
	(1.667)	(2.542)	(3.758)	(5.057)
small	13.90^{**}	13.98^{**}	13.87^{**}	13.36^{**}
	(2.447)	(2.437)	(2.338)	(2.352)
tchexper		1.156^{**}	0.7025^{**}	0.7814^{**}
		(0.2123)	(0.2057)	(0.2129)
boy			-15.34^{**}	-15.29^{**}
			(2.335)	(2.330)
freelunch			-33.79^{**}	-32.05^{**}
			(2.600)	(2.666)
white_asian			11.65^{**}	14.99^{**}
			(2.801)	(3.510)
tchwhite				-2.775
				(3.535)
tchmasters				-8.180^{**}
				(2.562)
schurban				-8.216^{**}
				(3.673)
schrural				-9.133^{**}
				(3.210)
n	3743	3743	3743	3743
$\bar{R}^2$	0.0083	0.0158	0.0945	0.0988
ℓ	−2.145e+004	−2.144e+004	−2.128e+004	−2.127e+004

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

The coefficient on the small indicator variable is not affected by adding or dropping variables from the model. This is indirect evidence that it is not correlated with other regressors. The effect of teacher experience on test scores falls quite a bit when `boy`, `freelunch`, and `white_asian` are added to the equation. This suggests that it is correlated with one or more of these variables and that omitting them from the model leads to biased estimation of the parameters by least squares.

7.4.1 School Fixed Effects

It may be that assignment to treatment groups is related to one or more of the observable characteristics (school size or teacher experience in this case). One way to control for these omitted effects is to used **fixed effects estimation**. This is taken up in more detail later. Here we introduce it to show off a useful **gretl** function called `dummify`.

The `dummify` command creates dummy variables for each distinct value present in a series, `x`. In order for it to work, you must first tell **gretl** that `x` is in fact a discrete variable. We want to create a set of indicator variables, one for each school in the dataset. First declare the `schid` variable to be discrete and then `dummify` it.

Here is the code and another model table that mimics Table 7.7 in *POE4*.

```
1  discrete schid
2  list d = dummify(schid)
3  ols totalscore x1 --quiet
4  scalar sser = $ess
5  scalar r_df = $df
6  modeltab add
7  ols totalscore x2 --quiet
8  modeltab add
9  ols totalscore x1 d --quiet
10 scalar sseu = $ess
11 scalar u_df = $df
12 modeltab add
13 ols totalscore x2 d --quiet
14 modeltab add
15 modeltab show
16 modeltab free
```

The `discrete` function in line 1 makes `schid` into a discrete variable. The next line creates a `list` that includes each of the variables created by `dummify(schid)`. Then, all you have to do is add it to the variable list that includes the fixed effects. **Gretl** smartly avoids the dummy variable trap by dropping one of the indicator variables from the regression.

Here is what you get with the indicator coefficients suppressed:

OLS estimates
Dependent variable: totalscore

	(1)	(2)	(3)	(4)
const	918.0**	907.6**	838.8**	830.8**
	(1.667)	(2.542)	(11.56)	(11.70)

small	13.90**	13.98**	16.00**	16.07**
	(2.447)	(2.437)	(2.223)	(2.218)
tchexper		1.156**		0.9132**
		(0.2123)		(0.2256)
School Effects	no	no	yes	yes
n	3743	3743	3743	3743
$\bar{R}^2$	0.0083	0.0158	0.2213	0.2245
ℓ	−2.145e+004	−2.144e+004	−2.096e+004	−2.095e+004

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

The estimated slopes in columns (3) and (4) match those in *POE4*. The intercepts are different only because a different reference group was used. The substance of the results is unaffected.

Testing the null hypothesis that the fixed effects are zero is very simple. Compare the restricted and unrestricted sum of squared errors using a F-statistic. The restricted sum of squared errors is saved for model (1) and the unrestricted for model (3). The statistic is computed using

```
scalar J = r_df-u_df
scalar fstat = ((sser - sseu)/J)/(sseu/u_df)
pvalue f J u_df fstat
```

and the result is:

```
Generated scalar J = 78
Generated scalar fstat = 14.1177
F(78, 3663): area to the right of 14.1177 = 1.70964e-154
(to the left: 1)
```

Notice how the difference in the number of degrees of freedom reveals how many restrictions are imposed on the model. Given the number of times we've used this computation, it may pay to write a **gretl** function to automate it.

7.4.2 Using Linear Probability to Verify Random Assignment

A number of variables are omitted from the model and it is safe to do so as long as they are not correlated with regressors. This would be evidence of assignments to the control group that are systematic. This can be checked using a regression. Since `small` is an indicator, we use a linear probability regression.

The independent variables include a constant, `boy white_asian`, `tchexper` and `freelunch`. The result is

OLS, using observations 1–3743
Dependent variable: small
Heteroskedasticity-robust standard errors, variant HC3

	Coefficient	Std. Error	t-ratio	p-value
const	0.4665	0.0253	18.46	7.33e-073
boy	0.0014	0.0163	0.09	0.931
white_asian	0.0044	0.0197	0.22	0.823
tchexper	−0.0006	0.0014	−0.42	0.676
freelunch	−0.0009	0.0183	−0.05	0.961

Sum squared resid	930.9297	S.E. of regression	0.499044
R^2	0.000063	Adjusted R^2	-0.001007
$F(4, 3738)$	0.059396	P-value(F)	0.993476

The overall-F statistic is not significant at 10%. None of the individual t-ratios are significant. Finally, a test of the hypothesis that the constant is $\beta_1 = 0.5$ cannot be rejected. A value of 0.5 would be consistent with assigning children to a small or large class by a fair coin flip. I think it is safe to omit these regressors from the model.

7.5 Differences-in-Differences Estimation

If you want to learn about how a change in policy affects outcomes, nothing beats a randomized controlled experiment. Unfortunately, these are rare in economics because they are either very expensive of morally unacceptable. No one want to determines what the return to schooling is by randomly assigning people to a prescribed number of schooling years. That choice should be yours and not someone else's.

But, the evaluation of policy is not hopeless when randomized controlled experiments are impossible. Life provides us with situations that happen to different groups of individuals at different points in time. Such events are not really random, but from a statistical point of view the treatment may appear to be randomly assigned. That is what so-called **natural experiments** are about. You have two groups of similar people. For whatever reason, one group gets treated to the policy and the other does not. Comparative differences are attributed to the policy.

In the example, we will look at the effects of a change in the minimum wage. It is made possible because the minimum wage is raised in one state and not another. The similarity of states is important because the non-treated state is going to be used for comparison.

The data come from Card and Krueger and are found in the file *njmin3.gdt*. We will open it and look at the summary statistics by state.

```
1 open "@gretldir\data\poe\njmin3.gdt"
2 smpl d = 0 --restrict
3 summary fte --by=nj --simple
4 smpl full
5 smpl d = 1 --restrict
6 summary fte --by=nj --simple
7 smpl full
```

Since we want to get a picture of what happened in NJ and PA before and after NJ raised the minimum wage we restrict the sample to before the increase. Then get the summary statistics for `fte` by state in line 3. Restore the full sample and then restrict it to after the policy `d=1`. Repeat the summary statistics for `fte`. The results suggest not much difference at this point.

```
nj = 0 (n = 79) d=0:

                      Mean        Minimum        Maximum      Std. Dev.
fte                 23.331         7.5000         70.500         11.856

nj = 1 (n = 331) d=0:
                      Mean        Minimum        Maximum      Std. Dev.
fte                 20.439         5.0000         85.000         9.1062

nj = 0 (n = 79) d=1:

                      Mean        Minimum        Maximum      Std. Dev.
fte                 21.166        0.00000         43.500         8.2767

nj = 1 (n = 331) d=1:

                      Mean        Minimum        Maximum      Std. Dev.
fte                 21.027        0.00000         60.500         9.2930
```

Now, make some variable list and run a few regressions

```
list x1 = const nj d d_nj
list x2 = x1 kfc roys wendys co_owned
list x3 = x2 southj centralj pa1

```

```
ols fte x1
modeltab add
ols fte x2
modeltab add
ols fte x3
modeltab add
modeltab show
```

The first set of variables include the indicator variables `nj`, `d` and their interaction. The second set adds more indicators for whether the jobs are at `kfc`, `roys`, or `wendys` and if the store is companied owned. The final set add more indicators for location.

The results from the three regressions appear below:

OLS estimates
Dependent variable: fte

	(1)	(2)	(3)
const	23.33^{**}	25.95^{**}	25.32^{**}
	(1.072)	(1.038)	(1.211)
nj	-2.892^{**}	-2.377^{**}	-0.9080
	(1.194)	(1.079)	(1.272)
d	-2.166	-2.224	-2.212
	(1.516)	(1.368)	(1.349)
d_nj	2.754	2.845^{*}	2.815^{*}
	(1.688)	(1.523)	(1.502)
kfc		-10.45^{**}	-10.06^{**}
		(0.8490)	(0.8447)
roys		-1.625^{*}	-1.693^{**}
		(0.8598)	(0.8592)
wendys		-1.064	-1.065
		(0.9292)	(0.9206)
co_owned		-1.169	-0.7163
		(0.7162)	(0.7190)
southj			-3.702^{**}
			(0.7800)
centralj			0.007883
			(0.8975)
pa1			0.9239
			(1.385)
n	794	794	794
$\bar{R}^2$	0.0036	0.1893	0.2115

ℓ	−2904	−2820	−2808

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

The coefficient on `d_nj` is the difference-in-differences estimator of the change in employment due to a change in the minimum wage. It is not significantly different from zero in this case and we can conclude that raising the minimum wage in New Jersey did not adversely affect employment.

In the previous analysis we did not exploit an important feature of Card and Krueger's data. The same restaurants were observed before and after in both states–in 384 of the 410 observations. It seems reasonable to limit the before and after comparison to the same units.

This requires adding an individual fixed effect to the model and dropping observations that have no before or after with which to compare.

```
smpl missing(demp) != 1 --restrict
ols demp const nj
```

Fortunately, the data set includes the ΔFTE where it is called `demp`. Dropping the observations for `demp` that are missing and using least squares to estimate the parameters of the simple regression yield:

$$\widehat{\text{demp}} = \underset{(0.73126)}{-2.28333} + \underset{(0.81519)}{2.75000}\,\text{nj}$$
$$T = 768 \quad \bar{R}^2 = 0.0134 \quad F(1, 766) = 11.380 \quad \hat{\sigma} = 8.9560$$
(standard errors in parentheses)

Again, the coefficient on nj is not significant at the 5% level and we cannot conclude that the increase in minimum wage affects employment.

7.6 Script

```
set echo off
open "@gretldir\data\poe\utown.gdt"
# print first 8 observations
smpl 1 8
print price sqft age utown pool fplace --byobs
```

```
# obtain summary statistics for full sample
smpl full
summary

# create indicator variable for large homes
series ld = (sqft>25)
discrete ld
smpl 1 8
print ld sqft --byobs
smpl full

# create interaction and estimate model
series sqft_utown=sqft*utown
ols price const utown sqft sqft_utown age pool fplace

# generate some marginal effects
scalar premium = $coeff(utown)*1000
scalar sq_u = 10*($coeff(sqft)+$coeff(sqft_utown))
scalar sq_other = 10*$coeff(sqft)
scalar depr = 1000*$coeff(age)
scalar sp = 1000*$coeff(pool)
scalar firep = 1000*$coeff(fplace)
printf "\n University Premium = $%8.7g\n\
Marginal effect of sqft near University = $%7.6g\n\
Marginal effect of sqft elsewhere = $%7.6g\n\
Depreciation Rate = $%7.2f\n\
Pool = $%7.2f\n\
Fireplace = $%7.2f\n",premium,sq_u,sq_other,depr,sp,firep
omit sqft_utown

# testing joint hypotheses
open "@gretldir\data\poe\cps4_small.gdt"
series blk_fem = black*female
ols wage const educ black female blk_fem
restrict
  b[3]=0
  b[4]=0
  b[5]=0
end restrict

ols wage const educ black female blk_fem south midwest west
omit south midwest west
scalar sser = $ess

# creation of interactions using a loop
list x = const educ black female blk_fem
list dx = null
loop foreach i x
  series south_$i = south * $i
  list dx = dx south_$i
endloop
```

```
modeltab clear
ols wage x dx
scalar sseu = $ess
scalar dfu = $df
modeltab add

# estimating subsets
smpl south=1 --restrict
ols wage x
modeltab add
smpl full

smpl south=0 --restrict
ols wage x
modeltab add
modeltab show

# Chow tests
smpl full
ols wage x
scalar sser = $ess
scalar fstat = ((sser-sseu)/5)/(sseu/dfu)
pvalue f 5 dfu fstat

ols wage x
chow south --dummy

# log-linear model--interpretation
open "@gretldir\data\poe\cps4_small.gdt"
logs wage
ols l_wage const educ female
scalar differential = 100*(exp($coeff(female))-1)

# linear probability model with HCCME
open "@gretldir\data\poe\coke.gdt"
ols coke const pratio disp_coke disp_pepsi --robust

# treatment effects
open "@gretldir\data\poe\star.gdt"
list v = totalscore small tchexper boy freelunch \
         white_asian tchwhite tchmasters schurban schrural
summary v --by=small --simple
summary v --by=regular --simple

smpl aide != 1 --restrict
list x1 = const small
list x2 = x1 tchexper
list x3 = x1 boy freelunch white_asian
list x4 = x1 tchwhite tchmasters schurban schrural

ols totalscore x1 --quiet
```

```
modeltab add
ols totalscore x2 --quiet
modeltab add
ols totalscore x3 --quiet
modeltab add
ols totalscore x4 --quiet
modeltab add
modeltab show
modeltab free

# manual creation of multiple indicators for school id
discrete schid
list d = dummify(schid)
ols totalscore x1 --quiet
scalar sser = $ess
scalar r_df = $df
modeltab add
ols totalscore x2 --quiet
modeltab add
ols totalscore x1 d --quiet
scalar sseu = $ess
scalar u_df = $df
modeltab add
ols totalscore x2 d --quiet
modeltab add
modeltab show
modeltab free

scalar J = r_df-u_df
scalar fstat = ((sser - sseu)/J)/(sseu/u_df)
pvalue f J u_df fstat

# testing random assignment of students
ols small const boy white_asian tchexper freelunch
restrict
    b[1]=.5
end restrict

# differences-in-differences
open "@gretldir\data\poe\njmin3.gdt"
smpl d = 0 --restrict
summary fte --by=nj --simple
smpl full
smpl d = 1 --restrict
summary fte --by=nj --simple
smpl full

list x1 = const nj d d_nj
list x2 = x1 kfc roys wendys co_owned
list x3 = x2 southj centralj pa1

```

```
summary x1 fte

ols fte x1
modeltab add
ols fte x2
modeltab add
ols fte x3
modeltab add
modeltab show
modeltab free

smpl missing(demp) != 1 --restrict
ols demp const nj
```

Chapter 8

Heteroskedasticity

The simple linear regression models of chapter 2 and the multiple regression model in Chapter 5 can be generalized in other ways. For instance, there is no guarantee that the random variables of these models (either the y_i or the e_i) have the same inherent variability. That is to say, some observations may have a larger or smaller variance than others. This describes the condition known as **heteroskedasticity**. The general linear regression model is shown in equation (8.1) below.

$$y_i = \beta_1 + \beta_2 x_{i2} + \cdots + \beta_k x_{iK} + e_i \quad i = 1, 2, \ldots, N \tag{8.1}$$

where y_i is the dependent variable, x_{ik} is the i^{th} observation on the k^{th} independent variable, $k = 2, 3, \ldots, K$, e_i is random error, and $\beta_1, \beta_2, \ldots, \beta_K$ are the parameters you want to estimate. Just as in the simple linear regression model, e_i, have an average value of zero for each value of the independent variables and are uncorrelated with one another. The difference in this model is that the variance of e_i now depends on i, i.e., the observation to which it belongs. Indexing the variance with the i subscript is just a way of indicating that observations may have different amounts of variability associated with them. The error assumptions can be summarized as $e_i|x_{i2}, x_{i3}, \ldots x_{iK}\ iid\ N(0, \sigma_i^2)$.

The intercept and slopes, β_1, β_2, …, β_K, are consistently estimated by least squares even if the data are heteroskedastic. Unfortunately, the usual estimators of the least squares standard errors and tests based on them are inconsistent and invalid. In this chapter, several ways to detect heteroskedasticity are considered. Also, statistically valid ways of estimating the parameters of 8.1 and testing hypotheses about the βs when the data are heteroskedastic are explored.

8.1 Food Expenditure Example

First, a simple model of food expenditures is estimated using least squares. The model is

$$food\_exp_i = \beta_1 + \beta_2 income_i + e_i \quad i = 1, 2, \ldots, N \tag{8.2}$$

where *food_exp*$_i$ is food expenditure and *income*$_i$ is income of the i^{th} individual. When the errors of the model are heteroskedastic, then the least squares estimator of the coefficients is consistent. That means that the least squares *point estimates* of the intercept and slope are useful. However, when the errors are heteroskedastic the usual least squares **standard errors** are **inconsistent** and therefore should not be used to form confidence intervals or to test hypotheses.

To use least squares estimates with heteroskedastic data, at a very minimum, you'll need a consistent estimator of their standard errors in order to construct valid tests and intervals. A simple computation proposed by White accomplishes this. Standard errors computed using White's technique are loosely referred to as **robust**, though one has to be careful when using this term; the standard errors are robust to the *presence of heteroskedasticity* in the errors of model (but not necessarily other forms of model misspecification).

Open the *food.gdt* data in **gretl** and estimate the model using least squares.

```
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
gnuplot food_exp income --linear-fit
```

This yields the usual least squares estimates of the parameters, but produces the wrong standard errors when the data are heteroskedastic. To get an initial idea of whether this might be the case a plot of the data is generated and the least squares line is graphed. If the data are heteroskedastic with respect to income then you will see more variation around the regression line for some levels of income. The graph is shown in Figure 8.1 and this appears to be the case. There is significantly more variation in the data for high incomes than for low.

To obtain the heteroskedasticity robust standard errors, simply add the `--robust` option to the regression as shown in the following **gretl** script. After issuing the `--robust` option, the standard errors stored in the accessor `$stderr(income)` are the robust ones.

```
ols food_exp const income --robust
# confidence intervals (Robust)
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
printf "\nThe 95%% confidence interval is (%.3f, %.3f).\n",lb,ub
```

In the script, we have used the `critical(t,$df,0.025)` function to get the desired critical value from the t-distribution. Remember, the degrees of freedom from the preceding regression are stored in `$df`. The first argument in the function indicates the desired distribution, and the last is the desired right-tail probability ($\alpha/2$ in this case).

The script produces

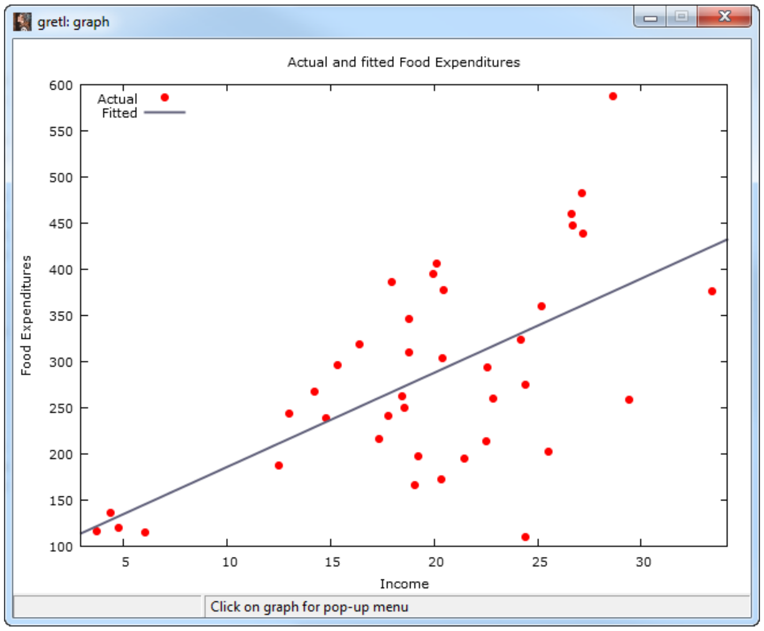


Figure 8.1: Plot of food expenditures against income with least squares fit.

```
The 95% confidence interval is (6.391, 14.028).
```

This can also be done from the pull-down menus. Select `Model>Ordinary Least Squares` (see Figure 2.6) to generate the dialog to specify the model shown in Figure 8.2 below. Note, the check box to generate 'robust standard errors' is circled. You will also notice that there is a button labeled **Configure** just to the right of the 'Robust standard errors' check box. Clicking this button reveals a dialog from which several options can be selected. In this case, we can select the particular method that will be used to compute the robust standard errors and even set robust standard errors to be the default computation for least squares. This dialog box is shown in Figure 8.3 below.

To reproduce the results in Hill et al. (2011), you'll want to select HC1 from the pull-down list. As you can see, other **gretl** options can be selected here that affect the default behavior of the program. The particular variant it uses depends on which dataset structure you have defined for your data. If none is defined, **gretl** assumes you have cross-sectional data.

The model results for the food expenditure example appear in the table below. After estimating the model using the dialog, you can use `Analysis>Confidence intervals for coefficients` to generate 95% confidence intervals. Since you used the `robust` option in the dialog, these will be based on the variant of White's standard errors chosen using the 'configure' button. In this case, I chose HC3, which some suggest performs slightly better in small samples. The result is:

```
VARIABLE         COEFFICIENT      95% CONFIDENCE INTERVAL
     const         83.4160           25.4153      141.417
    income         10.2096           6.39125      14.0280
```

OLS, using observations 1–40
Dependent variable: food_exp
Heteroskedasticity-robust standard errors, variant HC3

	Coefficient	Std. Error	t-ratio	p-value
const	83.4160	28.6509	2.9115	0.0060
income	10.2096	1.88619	5.4128	0.0000

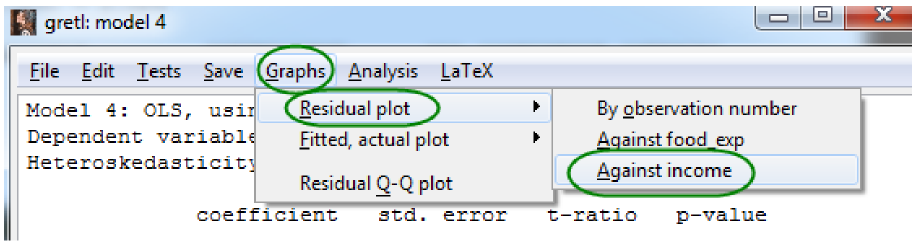
Mean dependent var	283.5735	S.D. dependent var	112.6752
Sum squared resid	304505.2	S.E. of regression	89.51700
R^2	0.385002	Adjusted R^2	0.368818
$F(1, 38)$	29.29889	P-value(F)	3.63e–06

Table 8.1: Least squares estimates with the usual and robust standard errors.

8.2 Detecting Heteroskedasticity

In the discussion above we used a graph of the data and the regression function to give us an initial reading of whether the data are heteroskedastic. Residual plots are equally useful, but some care must be taken in generating and interpreting them. By their very nature, plots allow you to 'see' relationships one variable at a time. If the heteroskedasticity involves more than one variable they may not be very revealing.

In Figure 8.4 is a plot of the least squares residuals against income. It appears that for larger levels of income there is much higher variance in the residuals. The graph was generated from the model window by selecting `Graphs>Residual plot>Against income`. I also right-clicked on the graph, chose **Edit** and altered its appearance a bit. Summoning the dialog looks like



```
Model 4: OLS, usi
Dependent variable
Heteroskedasticity

                    coefficient    std. error    t-ratio    p-value
```

Of course, you can also generate graphs from a script, which in this case is:

```
ols food_exp const income --robust
series res = $uhat
setinfo res -d "Least Squares Residuals" -n "Residual"
gnuplot res income --output=c:\Temp\olsres
```

In this script we continue to expand the use of **gretl** functions. The residuals are saved in line 2. Then in line 3 the `setinfo` command is used to change the description and the graph label using the `-d` and `-n` switches, respectively. Then **gnuplot** is called to plot `res` against `income`. This time the output is directed to a specific file. Notice that no suffix was necessary. To view the file in MS Windows, simply `lauch wgnuplot` and `load 'c:\Temp\olsres'`.

Another graphical method that shows the relationship between the magnitude of the residuals and the independent variable is shown below:

```
series abs_e = abs(res)
setinfo abs_e -d "Absolute value of the LS Residuals"\
      -n "Absolute Value of Residual"
gnuplot abs_e income --loess-fit --output=c:\temp\loessfit.plt
```

The graph appears in Figure 8.5. To generate this graph two things have been done. First, the absolute value of the least squares residuals have been saved to a new variable called `abs_e`. Then these are plotted against `income` as a scatter plot and as a locally weighted, smoothed scatterplot estimated by process called **loess**.

The basic idea behind loess is to create a new variable that, for each value of the dependent variable, y_i, contains the corresponding smoothed value, y_i^s. The smoothed values are obtained by running a regression of y on x by using only the data (x_i, y_i) and a few of the data points near this one. In loess, the regression is weighted so that the central point (x_i, y_i) gets the highest weight and points that are farther away (based on the distance $\mid x_j - x_i \mid$) receive less weight. The estimated regression line is then used to predict the smoothed value y_i^s for y_is only. The procedure is repeated to obtain the remaining smoothed values, which means that a separate weighted regression is performed for every point in the data. Obviously, if your data set is large, this can take a while. Loess is said to be a desirable smoother because of it tends to follow the data. Polynomial smoothing methods, for instance, are global in that what happens on the extreme left of a scatterplot can affect the fitted values on the extreme right.

One can see from the graph in Figure 8.5 that the residuals tend to get larger as income rises, reaching a maximum at 28. The residual for an observation having the largest income is relatively small and the locally smoothed prediction causes the line to start trending downward.

8.3 Lagrange Multiplier Tests

There are many tests of the null hypothesis of homoskedasticity that have been proposed elsewhere. Two of these, based on Lagrange multipliers, are particularly simple to do and useful. The first is sometimes referred to as the Breusch-Pagan (BP) test. The second test is credited to White.

The null and alternative hypotheses for the Breusch-Pagan test are

$$H_0 : \sigma_i^2 = \sigma^2$$
$$H_1 : \sigma_i^2 = h(\alpha_1 + \alpha_2 z_{i2} + \dots \alpha_s z_{iS})$$

The null hypothesis is that the data are homoskedastic. The alternative is that the data are heteroskedastic in a way that depends upon the variables z_{is}, $i = 2, 3, \dots, S$. These variables are exogenous and correlated with the model's variances. The function $h()$, is not specified. It could be anything that depends on its argument, i.e., the linear function of the variables in z. Here are the steps:

1. Estimate the regression model
2. Save the residuals
3. Square the residuals
4. Regress the squared residuals on z_{is}, $i = 2, 3, \dots, S$.
5. Compute NR^2 from this regression and compare it to the α level critical value from the $\chi^2(S-1)$ distribution.

The **gretl** script to perform the test manually is

```
ols food_exp const income
series sq_ehat = $uhat*$uhat
ols sq_ehat const income
scalar NR2 = $trsq
pvalue X 1 NR2
```

The only new item in this script is the use of the accessor, `$trsq`. This is the saved value of NR^2 from the previously estimated model. The output from the script is

```
Replaced scalar NR2 = 7.38442
Chi-square(1): area to the right of 7.38442 = 0.00657911
(to the left: 0.993421)
```

The p-value is less than 5% and we would reject the homoskedasticity null at that level. The heteroskedasticity seen in the residual plots appears to be confirmed.

Gretl has a built-in function that will compute a special case of the BP test that yields the same result in this example. The

```
ols food_exp const income
modtest --breusch-pagan
```

Produces

```
Breusch-Pagan test for heteroskedasticity
OLS, using observations 1-40
Dependent variable: scaled uhat^2

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const      -0.756949     0.633618     -1.195    0.2396
  income      0.0896185    0.0305534     2.933    0.0057  ***

  Explained sum of squares = 14.6879

Test statistic: LM = 7.343935,
with p-value = P(Chi-square(1) > 7.343935) = 0.006729
```

The functionality of `modtest --breusch-pagan` is limited in that it will include every regressor in the model as a `z`. It matches the result we derived manually because the model only includes `income` as the regressor. The `modtest --breusch-pagan` uses it as `z`. This means that you can't test a subset of the regressors with this function, nor can you use it to test for heteroskedasticity of exogenous variables that are not included in the regression function. In either of these cases, use the manual method described above; it is very easy to do.

8.3.1 The White Test

White's test is in fact just a minor variation on the Breusch-Pagan test. The null and alternative hypotheses are

$$\begin{aligned} H_0 &: \sigma_i^2 = \sigma^2 \quad \text{for all } i \\ H_1 &: \sigma_i^2 \neq \sigma_j^2 \quad \text{for at least 1 } i \neq j \end{aligned}$$

This is a composite alternative that captures every possibility other than the one covered by the null. If you know nothing about the nature of heteroskedasticity in your data, then this is a good place to start. The test is very similar to the BP test. In this test, the heteroskedasticity related variables (z_{is}, $i = 2, 3, \ldots, S$) include each non-redundant regressor, its square, and all cross products between regressors. See *POE4* for details. In the food expenditure model there is only one continuous regressor and an intercept. So, the constant squared and the cross product between the constant and income are redundant. This leaves only one unique variable to add to the model, income squared.

In **gretl** generate the squared value of income and regress the squared residuals from the model on income and its square. Compute NR^2 from this regression and compare it to α level critical value from the $\chi^2(S-1)$ distribution. As is the case in all the LM tests considered in this book, N is the number of observations in the second or auxiliary regression.

As with the BP test there is a built-in function that computes White's test. It generates all of the squares and unique cross-products to add to the model. The script to do both manual and built-in tests is found below:

```
ols food_exp const income
series sq_ehat = $uhat*$uhat
series sq_income = income^2
ols sq_ehat const income sq_income
scalar NR2 = $trsq
pvalue X 2 NR2

ols food_exp const income --quiet
modtest --white --quiet
```

The results from the two match perfectly and only that from the built-in procedure is produced below:

```
White's test for heteroskedasticity

Test statistic: TR^2 = 7.555079,
with p-value = P(Chi-square(2) > 7.555079) = 0.022879
```

The homoskedasticity null hypothesis is rejected at the 5% level.

8.3.2 Goldfeld Quandt Test for Heteroskedasticity

Using examples from Hill et al. (2011) a model of grouped heteroskedasticity is estimated and a Goldfeld-Quandt test is performed to determine whether the two sample subsets have the same error variance. The error variance associated with the first subset is σ_1^2 and that for the other subset is σ_2^2.

The null and alternative hypotheses are

$$H_0 : \sigma_1^2 = \sigma_2^2$$
$$H_1 : \sigma_1^2 \neq \sigma_2^2$$

Estimating both subsets separately and obtaining the estimated error variances allow us to construct the following ratio:

$$F = \frac{\hat{\sigma}_1^2/\sigma_1^2}{\hat{\sigma}_2^2/\sigma_2^2} \sim F_{df_1,df_2} \tag{8.3}$$

where $df_1 = N_1 - K_1$ from the first subset and $df_2 = N_2 - K_2$ is from the second subset. Under the null hypothesis that the two variances are equal

$$F = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2} \sim F_{df_1,df_2} \tag{8.4}$$

This is just the ratio of the estimated variances from the two subset regressions.

Wage Example

Below, I have written a **gretl** program to reproduce the wage example from Hill et al. (2011) that appears in chapter 8. The example is relatively straightforward and I'll not explain the script in much detail. It is annotated to help you decipher what each section of the program does.

The example consists of estimating wages as a function of education and experience. In addition, an indicator variable is included that is equal to one if a person lives in a metropolitan area. This is an "intercept" dummy which means that folks living in the metro areas are expected to respond similarly to changes in education and experience (same slopes), but that they earn a premium relative to those in rural areas (different intercept).

Each subset (metro and rural) is estimated separately using least squares and the standard error of the regression is saved for each (`$sigma`). Generally, you should put the group with the larger variance in the numerator. This allows a one-sided test and also allows you to use the standard p-value calculations as done below.

```
open "@gretldir\data\poe\cps2.gdt"
ols wage const educ exper metro
# Use only metro observations
smpl metro=1 --restrict
ols wage const educ exper
scalar stdm = $sigma
scalar df_m = $df
#Restore the full sample
smpl full
# Use only rural observations
smpl metro=0 --restrict
ols wage const educ exper
scalar stdr = $sigma
scalar df_r = $df
# GQ statistic
gq = stdm^2/stdr^2
```

```
scalar pv = pvalue(F, df_m, df_r, gq)
printf "\nThe F(%d, %d) statistic = %.3f. The right\
    side p-value is %.4g.\n",df_m,df_r,gq,pv
```

which produces

```
The F(805, 189) statistic = 2.088. The right side p-value is 1.567e-009.
```

Food Expenditure Example

In this example the data are sorted by income (low to high) and the subsets are created using observation numbers. This is accomplished using the GUI. Click `Data>Sort data` from the main menu bar to reveal the dialog box shown on the right side of Figure 8.6. The large income group is expected to have larger variance so its estimate will be placed in the numerator of the GQ ratio. The script is:

```
open "@gretldir\data\poe\food.gdt"
dataset sortby income
list x = const income
# large variance observations
smpl 21 40 --restrict
ols food_exp x
scalar stdL = $sigma
scalar df_L = $df
#Restore the full sample
smpl full
# small variance observations
smpl 1 20 --restrict
ols food_exp x
scalar stdS = $sigma
scalar df_S = $df
# GQ statistic
gq = stdL^2/stdS^2
scalar pv = pvalue(F, df_m, df_r, gq)
printf "\nThe F(%d, %d) statistic = %.3f. The right\
side p-value is %.4g.\n",df_m,df_r,gq,pv
```

This yields:

```
The F(18, 18) statistic = 3.615. The right side p-value is 0.004596.
```

Notice that in line 3 we have used the `dataset sortby` command in line 2 to sort the data without using the GUI.[1] This allows us to use the `smpl 21 40` command to limit the sample to observations 21-40 for the first subset. The other minor improvement is to use the `list` command in line 3 to specify the list of independent variables. This is useful since the same regression is estimated twice using different subsamples. The homoskedasticity null hypothesis is rejected at the 5% level since the p-value is smaller than 0.05.

8.4 Heteroskedastic-Consistent Standard Errors

The least squares estimator can be used to estimate the linear model even when the errors are heteroskedastic with good results. As mentioned in the first part of this chapter, the problem with using least squares in a heteroskedastic model is that the usual estimator of precision (estimated variance-covariance matrix) is not consistent. The simplest way to tackle this problem is to use least squares to estimate the intercept and slopes and use an estimator of least squares covariance that is consistent whether errors are heteroskedastic or not. This is the so-called heteroskcedasticity robust estimator of covariance that **gretl** uses.

In this example, the food expenditure data is used to estimate the model using least squares with both the usual and the robust sets of standard errors. Start by estimating the food expenditure model using least squares and add the estimates to the model table the estimates (Usual). Reestimate the model using the `--robust` option and store the results (`modeltab add`).

```
ols food_exp const income --quiet
modeltab add
ols food_exp const income --robust --quiet
modeltab add
modeltab show
```

The model table, which I edited a bit, is

OLS estimates
Dependent variable: food_exp

	(Usual)	(HC3 Robust)
const	72.96*	72.96**
	(38.83)	(19.91)
income	11.50**	11.50**
	(2.508)	(2.078)
n	20	20

[1]Replace `sortby income` with `dsortby income` to sort the sample by income in descending order.

R^2	0.5389	0.5389
ℓ	−109.1	−109.1

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

Notice that the coefficient estimates are the same, but that the estimated standard errors are different. Interestingly enough, the robust standard error for the slope is actually smaller than the usual one!

A number of commands behave differently when used after a model that employs the `--robust` option. For instance, the `omit` and restrict commands will use a Wald test instead of the usual one based on the difference in sum of squared errors.

The confidence intervals can be computed manually using saved results from the regression or from the model window of a model estimated through the GUI. Estimate the model using `ols` from the GUI. `Select Analysis > Confidence Intervals for coefficients` in the model window to generate confidence intervals based on the HCCME.

When you estimate the model, check the 'Robust standard errors' option (see Figure 8.2) and choose the 'Configure' button to select one of the options for bias correction using the pull-down menu for cross-sectional data as shown earlier in Figure 8.3.

These robust standard errors are obtained from what is often referred to as the heteroskedasticity-consistent covariance matrix estimator (HCCME) that was proposed by Huber and rediscovered by White. In econometrics, the HCCME standard errors may be referred to as White's standard errors or Huber/White standard errors. This probably accounts for the tab's name in the dialog box.

Since least squares is inefficient in heteroskedastic models, you'd think that there might be another unbiased estimator that is more precise. And, there is. The **generalized least squares** (GLS) estimator is, at least in principle, easy to obtain. Essentially, with the GLS estimator of the heteroskedastic model, the different error variances are used to reweigh the data so that they are all have the same (homoskedastic) variance. If the data are equally variable, then least squares is efficient!

8.5 Weighted Least Squares

If you know something about the structure of the heteroskedasticity, you may be able to get more precise estimates using a generalization of least squares. In heteroskedastic models, observations that are observed with high variance don't contain as much information about the location of the regression line as those observations having low variance. The basic idea of generalized least squares in this context is to reweigh the data so that all the observations contain the same level of information (i.e., same variance) about the location of the regression line. So, observations that contain more **noise** are given small weights and those containing more **signal** a higher weight. Reweighing the data in this way is known in some statistical disciplines as **weighted least squares**. This descriptive term is the one used by **gretl** as well.

Suppose that the errors vary proportionally with x_i according to

$$var(e_i) = \sigma^2 x_i \tag{8.5}$$

The errors are heteroskedastic since each error will have a different variance, the value of which depends on the level of x_i. Weighted least squares reweighs the observations in the model so that each transformed observation has the same variance as all the others. Simple algebra reveals that

$$\frac{1}{\sqrt{x_i}} var(e_i) = \sigma^2 \tag{8.6}$$

So, multiply equation (8.1) by $1/\sqrt{x_i}$ to complete the transformation. The transformed model is homoskedastic and least squares and the least squares standard errors are statistically valid and efficient.

Gretl makes this easy since it contains a function to reweigh all the observations according to a weight you specify. The command is `wls`, which naturally stands for **w**eighted **l**east **s**quares! The only thing you need to be careful of is how **gretl** handles the weights. **Gretl** takes the square root of the value you provide. That is, to reweigh the variables using $1/\sqrt{x_i}$ you need to use its square $1/x_i$ as the weight. **Gretl** takes the square root of `w` for you. To me, this is a bit confusing, so you may want to verify what **gretl** is doing by manually transforming y, x, and the constant and running the regression. The script file shown below does this.

In the example, you first have to create the weight, then call the function `wls`. The script appears below.

```
open "@gretldir\data\poe\food.gdt"

#GLS using built in function
series w = 1/income
wls w food_exp const income
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
printf "\nThe 95%% confidence interval is (%.3f, %.3f).\n",lb,ub

#GLS using OLS on transformed data
series wi = 1/sqrt(income)
series ys = wi*food_exp
series xs = wi*x
series cs = wi
ols ys cs xs
```

The first argument after `wls` is the name of the weight variable. Then, specify the regression to which it is applied. **Gretl** multiplies each variable (including the constant) by the ***square root*** of the given weight and estimates the regression using least squares.

In the next block of the program, $w_i = 1/\sqrt{x_i}$ is created and used to transform the dependent variable, x and the constant. Least squares regression using this manually weighted data yields the same results as you get with **gretl**'s `wls` command. In either case, you interpret the output of weighted least squares in the usual way.

The weighted least squares estimation yields:

Model 6: WLS, using observations 1–40
Dependent variable: food_exp
Variable used as weight: w

	Coefficient	Std. Error	t-ratio	p-value
const	78.6841	23.7887	3.3076	0.0021
income	10.4510	1.38589	7.5410	0.0000

Statistics based on the weighted data:

Sum squared resid	13359.45	S.E. of regression	18.75006
R^2	0.599438	Adjusted R^2	0.588897
$F(1, 38)$	56.86672	P-value(F)	4.61e–09
Log-likelihood	−172.9795	Akaike criterion	349.9591
Schwarz criterion	353.3368	Hannan–Quinn	351.1804

Statistics based on the original data:

Mean dependent var	283.5735	S.D. dependent var	112.6752
Sum squared resid	304611.7	S.E. of regression	89.53266

and the 95% confidence interval for the slope β_2 is (7.645, 13.257).

8.5.1 Grouped Data

In our discussion of the Goldfeld-Quandt test we decided that wages in rural and metropolitan areas showed different amounts of variation. When the heteroskedasticity occurs between groups, it is relatively straightforward to estimate the GLS corrections–this is referred to as **Feasible GLS** (FGLS).

The example consists of estimating wages as a function of education and experience and is based on the *cps2.gdt* used in the Goldfeld-Quandt test example. The strategy for combining these partitions and estimating the parameters using generalized least squares is fairly simple. Each subsample will be used to estimate the model and the standard error of the regression, $\hat{\sigma}$ (using the accessor `$sigma`) will be saved. Then each subsample is weighted by the reciprocal of its estimated variance (which is the squared value of the $1/\hat{\sigma}^2$.

There are a couple of ways to estimate each subsample. The first was used in the Goldfeld-Quandt test example where the metro subsample was chosen using `smpl metro=1 --restrict` and the rural one chosen with `smpl metro=0 --restrict`. Grouped GLS using this method can be found below:

```
1  open "@gretldir\data\poe\cps2.gdt"
2  list x = const educ exper
3  ols wage x metro
4  smpl metro --dummy
5  ols wage x
6  scalar stdm = $sigma
7  smpl full
8  series rural = 1-metro
9  smpl rural --dummy
10 ols wage x
11 scalar stdr = $sigma
12 #Restore the full sample
13 smpl full
14 series wm = metro*stdm
15 series wr = rural*stdr
16 series w = 1/(wm + wr)^2
17 wls w wage x metro
```

The `smpl` command is used in a new way here. In line 3 `smpl metro --dummy` restricts the sample based on the indicator variable `metro`. The sample will be restricted to only those observations for which `metro=1`. The wage equation is estimated in line 4 for the metro dwellers and the standard error of the regression is saved in line 5.

The next lines restore the full sample and create a new indicator variable for rural dwellers. Its value is just `1-metro`. We generate this in order to use the `smpl rural --dummy` syntax. We could have skipped generating the rural and simply used `smpl metro=0 --restrict`. In line 10 the model is estimated for rural dwellers and the standard error of the regression is saved.

The full sample must be restored and two sets of weights are going to be created and combined. In line 14 the statement `series wm = metro*stdm` multiplies the metro S.E. of the regression times the indicator variable. Its values will either be `stdm` for metro dwellers and 0 for rural dwellers. We do the same for rural dwellers in 15. Adding these two series together creates a single variable that contains only two distinct values, $\hat{\sigma}_M$ for metro dwellers and $\hat{\sigma}_R$ for rural ones. Squaring this and taking the reciprocal provides the necessary weights for the weighted least squares regression.

WLS, using observations 1–1000
Dependent variable: wage

	Coefficient	Std. Error	t-ratio	p-value
const	−9.39836	1.01967	−9.2170	0.0000
educ	1.19572	0.0685080	17.4537	0.0000
exper	0.132209	0.0145485	9.0874	0.0000
metro	1.53880	0.346286	4.4437	0.0000

Statistics based on the weighted data:

Sum squared resid	998.4248	S.E. of regression	1.001217
R^2	0.271528	Adjusted R^2	0.269334
$F(3, 996)$	123.7486	P-value(F)	3.99e–68
Log-likelihood	−1418.150	Akaike criterion	2844.301
Schwarz criterion	2863.932	Hannan–Quinn	2851.762

Statistics based on the original data:

Mean dependent var	10.21302	S.D. dependent var	6.246641
Sum squared resid	28585.82	S.E. of regression	5.357296

8.6 A Hetroskedasticity Function

A commonly used model for the error variance is the **multipicative heteroskedasticity** model. It appears below in equation 8.7.

$$\sigma_i^2 = \exp\left(\alpha_1 + \alpha_2 z_i\right) \tag{8.7}$$

The variable z_i is an independent explanatory variable that determines how the error variance changes with each observation. You can add additional zs if you believe that the variance is related to them (e.g., $\sigma_i^2 = \exp\left(\alpha_1 + \alpha_2 z_{i2} + \alpha_3 z_{i3}\right)$). It's best to keep the number of zs relatively small. The idea is to estimate the parameters of (8.7) using least squares and then use predictions as weights to transform the data.

In terms of the food expenditure model, let $z_i = ln(income_i)$. Then, taking the natural logarithms of both sides of (8.7) and adding a random error term, v_i, yields

$$\ln\left(\sigma_i^2\right) = \alpha_1 + \alpha_2 z_i + v_i \tag{8.8}$$

To estimate the αs, first estimate the linear regression (8.2) (or more generally, 8.1) using least squares and save the residuals. Square the residuals, then take the natural log; this forms an estimate of $\ln(\sigma_i^2)$ to use as the dependent variable in a regression. Now, add a constant and the zs to the right-hand side of the model and estimate the αs using least squares.

The regression model to estimate is

$$\ln(\hat{e}_i^2) = \alpha_1 + \alpha_2 z_i + v_i \tag{8.9}$$

where $\hat{e}_i^2$ are the least squares residuals from the estimation of equation (8.1). The predictions from this regression can then be transformed using the exponential function to provide weights for weighted least squares.

For the food expenditure example, the **gretl** code appears below.

```
ols food_exp const income
series lnsighat = log($uhat*$uhat)
series z = ln(income)
ols lnsighat const z
series predsighat = exp($yhat)
series w = 1/predsighat
wls w food_exp const income
```

The first line estimates the linear regression using least squares. Next, a new variable is generated (`lnsighat`) that is the natural log of the squared residuals from the preceding regression. Then, generate z as the natural log of *income*. Estimate the skedasticity function using least squares, take the predicted values (`yhat`) and use these in the exponential function (i.e., $\exp(\widehat{food\_exp_i})$). The reciprocal of these serve as weights for generalized least squares. Remember, **gretl** automatically takes the square roots of `w` for you in the `wls` function.

The output is:

```
WLS, using observations 1-40
Dependent variable: food_exp
Variable used as weight: w

             coefficient   std. error   t-ratio    p-value
  --------------------------------------------------------
  const        76.0538      9.71349      7.830    1.91e-09  ***
  income       10.6335      0.971514    10.95     2.62e-013 ***

Statistics based on the weighted data:

Sum squared resid    90.91135   S.E. of regression   1.546740
R-squared            0.759187   Adjusted R-squared   0.752850
F(1, 38)             119.7991   P-value(F)           2.62e-13
Log-likelihood      -73.17765   Akaike criterion     150.3553
Schwarz criterion    153.7331   Hannan-Quinn         151.5766
```

```
Statistics based on the original data:

Mean dependent var   283.5735   S.D. dependent var   112.6752
Sum squared resid    304869.6   S.E. of regression   89.57055
```

The model was estimated by least squares with the HCCME standard errors in section 8.1. The parameter estimates from FGLS are not much different than those. However, the standard errors are much smaller now. The HC3 standard error for the slope was 1.88 and is now only 0.97. The constant is being estimated more precisely as well. So, there are some potential benefits from using a more precise estimator of the parameters.

8.7 Heteroskedasticity in the Linear Probabilty Model

In chapter 7 we introduced the linear probability model. It was shown that the indicator variable, y_i is heteroskedastic. That is,

$$var(y_i) = \pi_i(1-\pi_i) \tag{8.10}$$

where π_i is the probability that the dependent variable is equal to 1 (the choice is made). The estimated variance is

$$\widehat{var(y_i)} = \hat{\pi}_i(1-\hat{\pi}_i) \tag{8.11}$$

This can be used to perform feasible GLS. The cola marketing data *coke.gdt* is the basis for this example. The independent variable, `coke`, takes the value of 1 if the individual purchases Coca-Cola and is 0 if not. The decision to purchase Coca-Cola depends on the ratio of the price relative to Pepsi, and whether displays for Coca-Cola or Pepsi were present. The variables `disp_coke=1` if a Coca-Cola display was present, otherwise 0; `disp_pepsi=1` if a Pepsi display was present, otherwise it is zero.

```
First, the data are loaded and the summary statistics are provided.
open "@gretldir\data\poe\coke.gdt"
summary --simple
list x = const pratio disp_coke disp_pepsi
```

The `--simple` option is used for the `summary` command. Then a `list` is created that contains the names of the independent variables to be used in the estimated models. The basic summary statistics are:

```
Summary statistics, using the observations 1 - 1140

                        Mean         Minimum         Maximum      Std. Dev.
```

```
coke                0.44737        0.00000        1.0000        0.49744
pr_pepsi             1.2027        0.68000        1.7900        0.30073
pr_coke              1.1901        0.68000        1.7900        0.29992
disp_pepsi          0.36404        0.00000        1.0000        0.48137
disp_coke           0.37895        0.00000        1.0000        0.48534
pratio               1.0272        0.49721        2.3247        0.28661
```

Everything looks good. There are no negative prices, and the indicator variables are all contained between 0 and 1. The magnitudes of the means are reasonable.

Next, least squares is used to estimate the model twice: once with usual standard errors and again with the HCCME standard errors produced by the `--robust` option. Each is added to a model table using `modeltab add`.

```
# OLS
ols coke x
modeltab add
# OLS w/robust
ols coke x --robust
modeltab add
```

Feasible GLS will be estimated in two ways. In the first regression, we will omit any observation that has a negative estimated variance. Remember that one of the problems with linear probability is that predictions are not constrained to lie between 0 and 1. If $\hat{y}_i < 0$ or $\hat{y}_i > 1$, then variance estimates will be negative. In the first line below a new series is created to check this condition. If the variance, `varp`, is greater than zero, `pos` will be equal to 1 and if not, then it is zero. The second line creates a weight for `wls` that is formed by multiplying the indicator variable `pos` times the reciprocal of the variance. In this way, any nonnegative weights become zeros.

Remove observations with negative variance

```
series p = $yhat
series varp = p*(1-p)
series pos = (varp > 0)
series w = pos * 1/varp
# omit regression
wls w coke x
modeltab add
```

The first line uses the accessor for the predicted values from a linear regression, `$yhat`, and therefore it must follow least squares estimation of the linear probability model; in this model, they are interpreted as probabilities. Once again, a trick is being used to eliminate observations from the model. Basically, any observation that has a zero weight in `w` is dropped from the computation. There are equivalent ways to do this in **gretl** as shown below

```
Two other ways to drop observations
 smpl varp>0 --restrict
 setmiss 0 w
```

The restricting the sample is probably the most straightforward method. The second uses the `setmiss` command that changes the missing value code to 0 for elements of `w`; any observation where `w=0` is now considered missing and won't be used to estimate the model.

Finally, another feasible GLS estimation is done. This time, $\hat{p}_1$ is truncated at 0.01 if $\hat{y}_i < 0.01$ and to 0.99 if $\hat{y}_i > 0.99$. The code to do this is

```
WLS with truncated variances for observations out of bounds
series b = (p<.01) || (p>.99)
series pt = b*0.01 + p*(1-b)
series varp_t = pt*(1-pt)
series w_t = 1/varp_t
wls w_t coke x
modeltab add
modeltab show
```

The first line creates another indicator variable that takes the value of 1 if the predicted probability falls outside of the boundary. The `||` is a logical operator that takes the union of the two conditions (="OR"). The second line creates the truncated value of the probability using the indicator variable.

$$p_t = \begin{cases} b(0.01) + p(1-b) = 0.01 & \text{when } b = 1 \\ b(0.01) + p(1-b) = p & \text{when } b = 0 \end{cases} \tag{8.12}$$

There is another, less transparent, way to generate the truncated probabilities: use the ternary conditional assignment operator. This operates like an if statement and can be used to save a line of script. This syntax would create the series as

```
The conditional assignment operator
series pt = ( (p<.01) || (p>.99) ) ?  0.01 : p
```

Basically the bound condition in parentheses $(\texttt{p} < \texttt{.01})||(\texttt{p} > \texttt{.99})$ is checked: that is what the question mark represents. If it is true, `pt` is set to the first value that appears in front of the colon. If false, it is set to the value specified to the right of the colon. It operates very much like a traditional if statement in a spreadsheet program. This method is more efficient computationally as well, which could save some time if used in a loop to perform simulations.

Once the truncated probabilities are created, then the usual weighted least squares estimation can proceed. The model table appears below:

Dependent variable: coke

	(1) OLS	(2) OLS	(3) WLS	(4) WLS
const	0.8902**	0.8902**	0.8795**	0.6505**
	(0.06548)	(0.06563)	(0.05897)	(0.05685)
pratio	−0.4009**	−0.4009**	−0.3859**	−0.1652**
	(0.06135)	(0.06073)	(0.05233)	(0.04437)
disp_coke	0.07717**	0.07717**	0.07599**	0.09399**
	(0.03439)	(0.03402)	(0.03506)	(0.03987)
disp_pepsi	−0.1657**	−0.1657**	−0.1587**	−0.1314**
	(0.03560)	(0.03447)	(0.03578)	(0.03540)
n	1140	1140	1124	1140
$\bar{R}^2$	0.1177	0.1177	0.2073	0.0865
ℓ	−748.1	−748.1	−1617	−1858

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

Columns (1) and (2) are the OLS estimates with usual and robust standard errors, respectively. Column (3) uses WLS with the negative variance observations omitted from the sample. Column (4) is WLS with the negative predictions truncated. These results are quite a bit different from the others. This no doubt occurs because of the large weight being placed on the 16 observations whose weights were constructed by truncation. The $var(e_i) = 0.01(1 - 0.01) = 0.0099$. The square root of the reciprocal is approximately 10, a large weight to be placed on these 16 observations via WLS. Since these extreme observations carry a large weight relative to the others, they exert a considerable influence on the estimated regression.

8.8 Script

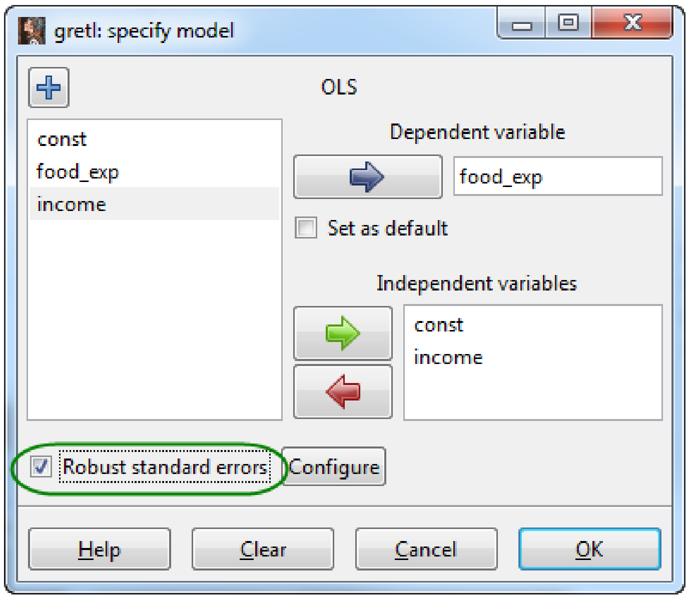
```
open "@gretldir\data\poe\food.gdt"
set echo off
ols food_exp const income
gnuplot food_exp income --linear-fit
# see section 1.4 of this manual for commands to view these plots.

# ols with HCCME standard errors
ols food_exp const income --robust
# confidence intervals (Robust)
```

```
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
printf "\nThe 95%% confidence interval is (%.3f, %.3f).\n",lb,ub

# residual plot
ols food_exp const income --robust
series res = $uhat
setinfo res -d "Least Squares Residuals" -n "Residual"
gnuplot res income --output=c:\Temp\olsres

# lauch gnuplot (Windows only)
launch wgnuplot
# To view graph, type: load 'C:\Temp\olsres' at prompt

# residual magnitude plot with loess fit
series abs_e = abs(res)
setinfo abs_e -d "Absolute value of the LS\
Residuals" -n "Absolute Value of Residual"
gnuplot abs_e income --loess-fit --output=c:\temp\loessfit.plt

# LM test for heteroskdasticity
ols food_exp const income
series sq_ehat = $uhat*$uhat
ols sq_ehat const income
scalar NR2 = $trsq
pvalue X 1 NR2

# built-in LM test
ols food_exp const income
modtest income --breusch-pagan

# White test
ols food_exp const income
series sq_ehat = $uhat*$uhat
series sq_income = income^2
ols sq_ehat const income sq_income
scalar NR2 = $trsq
pvalue X 2 NR2

# built-in White test
ols food_exp const income --quiet
modtest --white --quiet

# grouped data--Goldfeld-Quandt
open "@gretldir\data\poe\cps2.gdt"
ols wage const educ exper metro
# Use only metro observations
smpl metro=1 --restrict
ols wage const educ exper
scalar stdm = $sigma
scalar df_m = $df
```

```
#Restore the full sample
smpl full
# Use only rural observations
smpl metro=0 --restrict
ols wage const educ exper
scalar stdr = $sigma
scalar df_r = $df
# GQ statistic
gq = stdm^2/stdr^2
scalar pv = pvalue(F, df_m, df_r, gq)
printf "\nThe F(%d, %d) statistic = %.3f. The right side\
p-value is %.4g.\n",df_m,df_r,gq,pv

# Goldfeld-Quandt for food expenditure
open "@gretldir\data\poe\food.gdt"
dataset sortby income
list x = const income
ols food_exp x
# large variance observations
smpl 21 40
ols food_exp x
scalar stdL = $sigma
scalar df_L = $df
#Restore the full sample
smpl full
# small variance observations
smpl 1 20
ols food_exp x
scalar stdS = $sigma
scalar df_S = $df
# GQ statistic
gq = stdL^2/stdS^2
scalar pv = pvalue(F, df_L, df_S, gq)
printf "\nThe F(%d, %d) statistic = %.3f. The right\
  side p-value is %.4g.\n",df_L,df_S,gq,pv

# compare ols with and without HCCME
list x = const income
ols food_exp x --quiet
modeltab add
ols food_exp x --robust --quiet
modeltab add
modeltab show

# hypothesis test
ols food_exp x --robust
omit income
ols food_exp x --quiet
restrict
    b[2]=0
end restrict
```

```

ols food_exp x --robust --quiet
restrict
    b[2]=0
end restrict

open "@gretldir\data\poe\food.gdt"

#GLS using built in function
series w = 1/income
wls w food_exp const income
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
printf "\nThe 95%% confidence interval is (%.3f, %.3f).\n",lb,ub

#GLS using OLS on transformed data
series wi = 1/sqrt(income)
series ys = wi*food_exp
series xs = wi*income
series cs = wi
ols ys cs xs

#Wage Example
open "@gretldir\data\poe\cps2.gdt"
ols wage const educ exper metro
# Use only metro observations
smpl metro --dummy
ols wage const educ exper
scalar stdm = $sigma
smpl full
#Create a dummy variable for rural
series rural = 1-metro
#Restrict sample to rural observations
smpl rural --dummy
ols wage const educ exper
scalar stdr = $sigma
#Restore the full sample
smpl full
#Generate standard deviations for each metro and rural obs
series wm = metro*stdm
series wr = rural*stdr
series w = 1/(wm + wr)^2
#Weighted least squares
wls w wage const educ exper metro

# heteroskedastic model
open "@gretldir\data\poe\food.gdt"
ols food_exp const income
series lnsighat = log($uhat*$uhat)
series z = log(income)
ols lnsighat const z
```

```
series predsighat = exp($yhat)
series w = 1/predsighat
wls w food_exp const income

# linear probability model
open "@gretldir\data\poe\coke.gdt"
summary --simple
list x = const pratio disp_coke disp_pepsi
# OLS
ols coke x
modeltab add
# OLS w/robust
ols coke x --robust
modeltab add
series p = $yhat
series varp = p*(1-p)
series pos = (varp > 0)
series w = pos * 1/varp

# omit regression
wls w coke x
modeltab add

# smpl varp>0 --restrict
# setmiss 0 w
series b = (p<.01) || (p>.99)
series pt = b*0.01 + p*(1-b)
series varp_t = pt*(1-pt)
series w_t = 1/varp_t
# trunc regression
wls w_t coke x
modeltab add
modeltab show
```



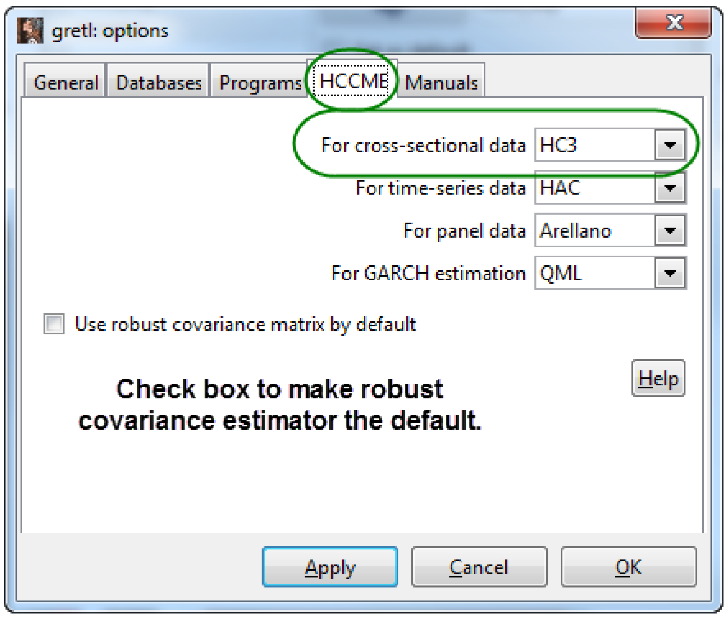
Dependent variable: food_exp

[ ] Set as default

Independent variables: const, income

[✓] Robust standard errors

Figure 8.2: Check the box for heteroskedasticity robust standard errors.



For cross-sectional data: HC3
For time-series data: HAC
For panel data: Arellano
For GARCH estimation: QML

[ ] Use robust covariance matrix by default

Check box to make robust covariance estimator the default.

Figure 8.3: Set the method for computing robust standard errors. These are located under the HCCME tab. From the pull-down list for cross-sectional data choose an appropriate option–HC3 in this case.

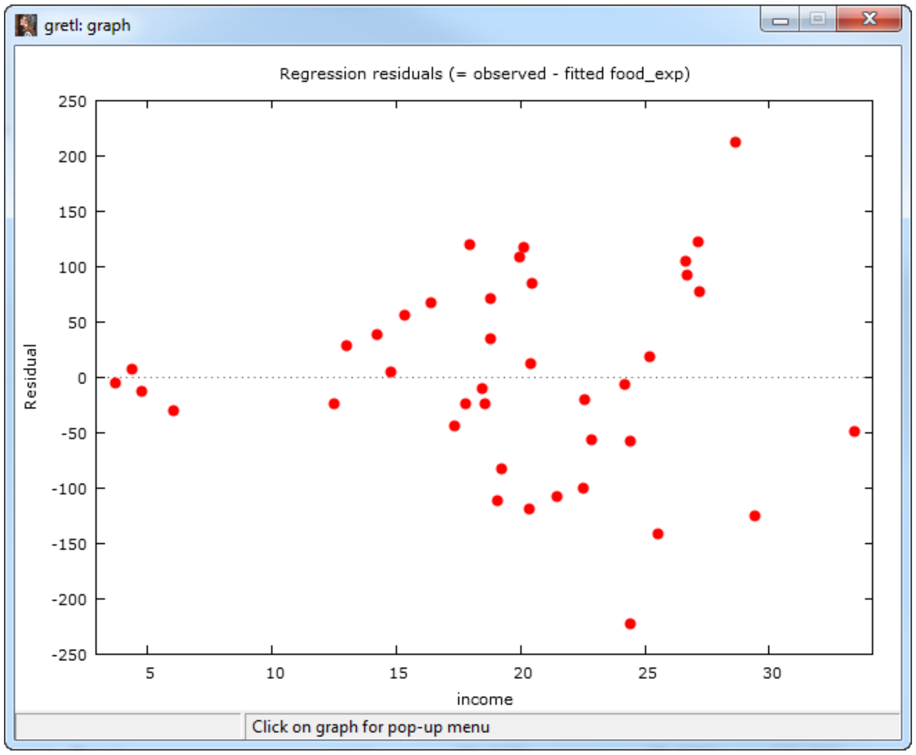


Figure 8.4: Plot of food expenditures against income with least squares fit.

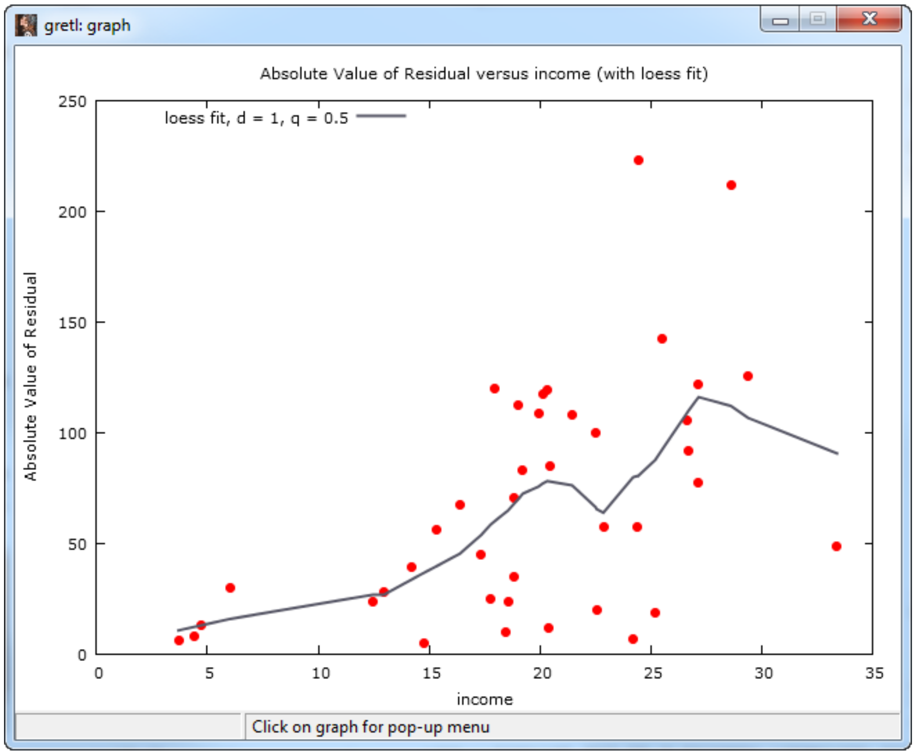


Figure 8.5: Plot of the absolute value of the food expenditures model residuals against income with loess fit.

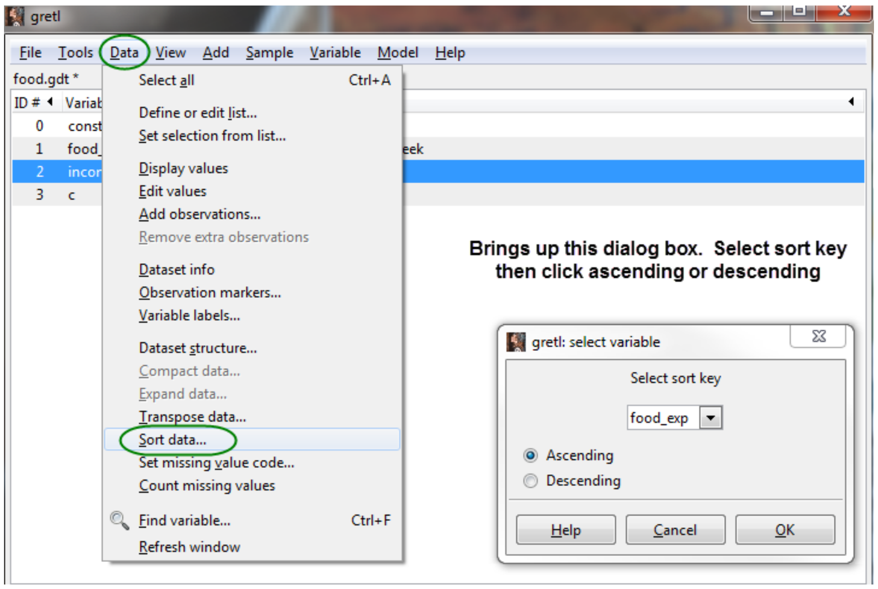


Figure 8.6: Select `Data>Sort` data from the main menu bar to reveal the dialog box shown on the right side of of this figure. Choose the desired sort key and indicate whether you want to sort in ascending or descending order.

Chapter 9

Regression with Time-Series Data: Stationary Variables

As in chapter 9 of Principles of Econometrics, 4th edition, three ways in which dynamics can enter a regression relationship are considered–through lagged values of the explanatory variable, lagged values of the dependent variable, and lagged values of the error term.

In time-series regressions the data need to be stationary in order for the usual econometric procedures to have the proper statistical properties. Basically this requires that the means, variances and covariances of the time-series data cannot depend on the time period in which they are observed. For instance, the mean and variance of GDP in the third quarter of 1973 cannot be different from those of the 4th quarter of 2006. Methods to deal with this problem have provided a rich field of research for econometricians in recent years and several of these techniques are explored later in chapter 12.

One of the first diagnostic tools used is a simple time-series plot of the data. A time-series plot will reveal potential problems with the data and suggest ways to proceed statistically. As seen in earlier chapters, time-series plots are simple to generate in **gretl** and a few new tricks will be explored below.

Finally, since this chapter deals with time-series observations the usual number of observations, N, is replaced by the more commonly used T. In later chapters, where both time-series and cross sectional data are used, both N and T are used.

9.1 Data Structures: Time Series

In order to take advantage of **gretl**'s many built-in functions for analyzing time-series data, one has to declare the data in the set to be a time-series. Since time-series are ordered in time their position relative to the other observations must be maintained. It is, after all, their temporal relationships that make analysis of this kind of data different from cross-sectional analysis.

If the data you have do not already have a proper date to identify the time period in which the observation was collected, then adding one is a good idea. This makes identification of historical periods easier and enhances the information content of graphs considerably. Most of the data sets distributed with your book have been declared to be time-series and contain the relevant dates in the set of variables. However, it is a good idea to know how to add this information yourself and we show how to do so here. Basically you need to identify to **gretl** that the data are time-series, you need to specify their frequency of observation, and then identify the starting date. As long as there are no 'holes' in the data, this should get you the relevant set of dates matched to the periods they are observed.

Before getting to the specific examples from the text, something should be said about how **gretl** handles dates and times.

Gretl is able to recognize dates as such in imported data if the date strings conform to the following rules. For annual data, you must use 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.

Gretl allows you to declare time-series annually, monthly, weekly, daily (5, 6, or 7 per week), hourly, decennially, and has a special command for other irregular dates. Its date handling features are reasonably good, but it is not nearly as sophisticated as those found in other software like Stata. On the other hand, for what it does it is much easier to use. It works beautifully with most datasets.

There are two methods of getting your dataset structured as a time-series. The first uses the GUI. `Click Data>Dataset` structure from the pull-down menu to initiate the data structure wizard. The wizard serves up a series of dialog boxes that help you to define when the observations occur. The first dialog defines the structure: the choices are cross-sectional, time-series, and panel. Choosing time-series brings up a dialog to set the frequency. Choices include: annual, quarterly, monthly, weekly, daily (5, 6, or 7 per week), hourly, decennial, a special command for other irregular dates. Choosing one of these brings up the next dialog that sets the start point. For instance, quarterly data might start at 3rd quarter of 1972. You would enter, `1972:3` in the box. Then the confirmation dialog opens. It reveals how **gretl** interpreted your choices. You check to see whether the data start and stop when expected. If so, then your data structure is almost certainly correct. If the end date is something other than you expect, then go back and try again. You may have some gaps in the data series that need to be filled in order for the dates and the number of observations to match up. Sometimes things need manual editing due to holidays and such. Be patient and get

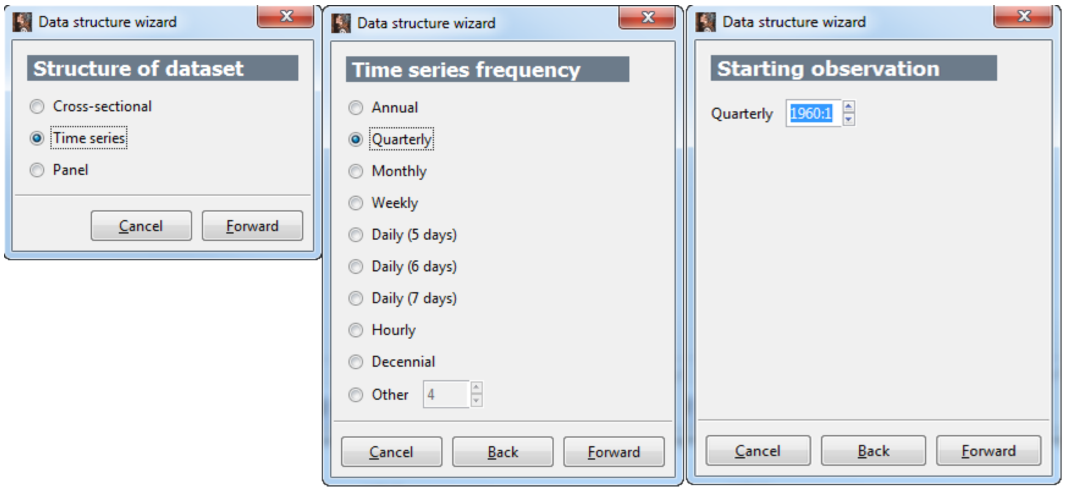


Figure 9.1: Choose `Data>Dataset structure` from the main window. This starts the **Dataset** wizard, a series of dialogs that allow you to specify the periodicity and dates associated with your data.

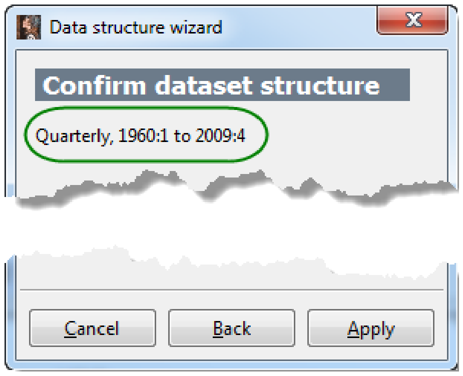
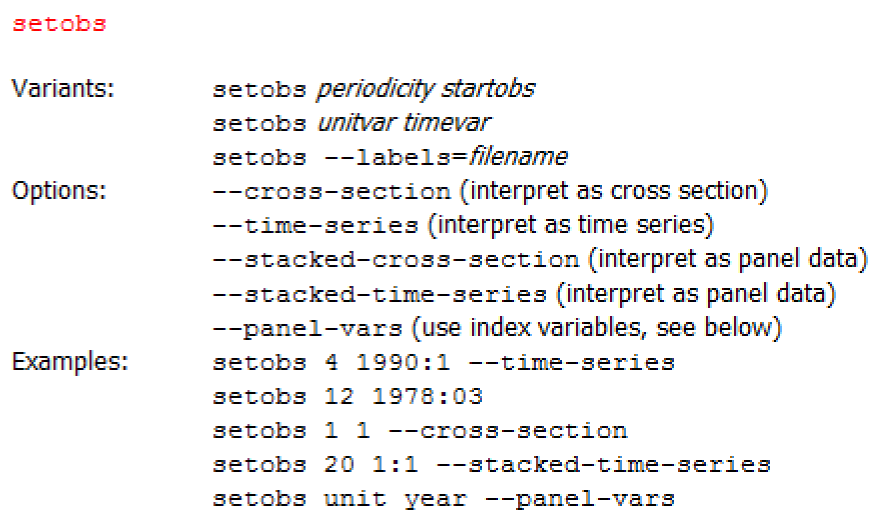


Figure 9.2: Check the confirmation box to be sure the expected time periods are given.

this right, otherwise you may end up having to redo you analysis. Figure 9.1 shows the first three dialog boxes for defining time-series structure. The last box (Figure 9.2) confirms that the series starts in 1960:1 and ends in 2009:4.

The `setobs` command is used to accomplish the same thing from the console or in a script. The syntax is summarized


setobs

Variants:	`setobs` *periodicity startobs*
	`setobs` *unitvar timevar*
	`setobs --labels=`*filename*
Options:	`--cross-section` (interpret as cross section)
	`--time-series` (interpret as time series)
	`--stacked-cross-section` (interpret as panel data)
	`--stacked-time-series` (interpret as panel data)
	`--panel-vars` (use index variables, see below)
Examples:	`setobs 4 1990:1 --time-series`
	`setobs 12 1978:03`
	`setobs 1 1 --cross-section`
	`setobs 20 1:1 --stacked-time-series`
	`setobs unit year --panel-vars`

Basically you define the periodicity and when the series starts. Then the options are used to indicate what the actual structure is (e.g., time-series). Some examples are found in Table 9.1.

9.2 Time-Series Plots

Gnuplot handles all the plotting in **gretl**. **Gretl** includes some functions that help to communicate with **gnuplot**, which makes things much easier to do. On the other hand, if you have something really fancy to plot, you may have to use **gnuplot** directly to get the desired result. All-in-all, **gretl**'s graphical interface that works with **gnuplot** is quite easy to use and powerful.

Gretl's time-series plot is really just an XY scatter plot against time with the `--lines` option used to connect the data point. It's relatively primitive. Clicking on a graph brings up a list of things you can do, including edit the graph. Clicking the *edit* button brings up the plot control dialog box (Figure 4.16) where substantial customization can be done.

Gretl also has a facility to plot multiple series in separate graphs that appear on the same page. This is accomplished using the `scatters` command or `View>Multiple graphs>Time-series` from the main menu bar. There is no built-in facility for further editing these graphs, but you can save them in several formats. Examples of this are found below.

Syntax	**Results**
`setobs 4 1990:1 --time-series`	Quarterly data that start in 1990:1
`setobs 1 1952 --time-series`	Annual data starting in 1952
`setobs 12 1990:03 --time-series`	Monthly data starting in March, 1990
`setobs 5 1950/01/06 --time-series`	Daily data (5 day weeks) starting Jan. 6, 1950

Table 9.1: Data structure using `setobs`: Some examples for time-series

In this example time-series graphs are plotted for the U.S. unemployment rate and GDP growth from 1985 to 2009. The data are found in the *okun.gdt* data file.

```
open "@gretldir\data\poe\okun.gdt"
setinfo g -d "percentage change in U.S. Gross Domestic Product, seasonally \
  adjusted" -n "Real GDP growth"
setinfo u -d "U.S. Civilian Unemployment Rate  (Seasonally adjusted)" -n \
  "Unemployment Rate"
gnuplot g --with-lines --time-series --output=c:\temp\okun_g.plt
gnuplot u --with-lines --time-series --output=c:\temp\okun_u.plt
```

The two plots are shown in Figure 9.3. The graphs can be combined using the GUI by choosing `View>Multiple graphs>Time-series`. The result appears in Figure 9.4. The **gretl** command to generate multiple series in multiple graphs is

```
scatters g u
```

9.3 Finite Distributed Lags

Finite distributed lag models contain independent variables and their lags as regressors.

$$y_t = \alpha + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \ldots \beta_q x_{t-q} + e_t \tag{9.1}$$

for $t = q+1, \ldots, T$. The particular example considered here is an examination of Okun's Law. In this model the change in the unemployment rate from one period to the next depends on the rate of growth of output in the economy.

$$u_t - u_{t-1} = -\gamma(g_t - g_N) \tag{9.2}$$

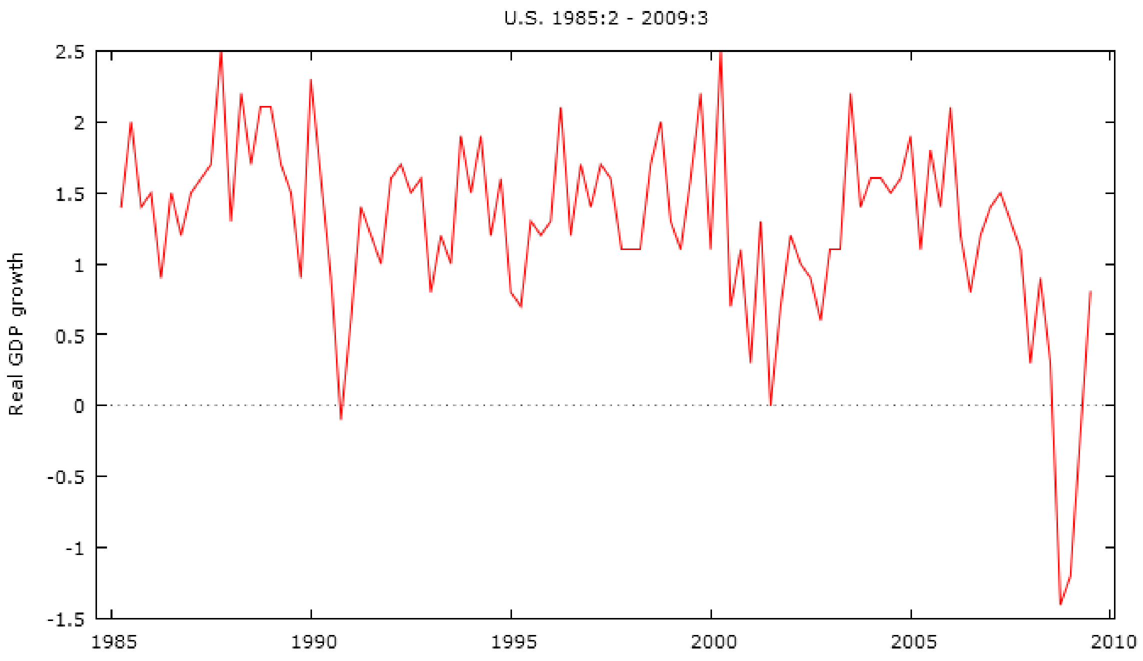
where u_t is the unemployment rate, g_t is GDP growth, and g_N is the normal rate of GDP growth. The regression model is

$$\Delta u_t = \alpha + \beta_0 g_t + e_t \tag{9.3}$$

where Δ is the difference operator, $\alpha = \gamma G_N$, and $\beta_0 = -\gamma$. An error term has been added to the model. The difference operator, $\Delta u = u_t - u_{t-1}$ for all $= 2, 3, \ldots, T$. Notice that when you take the difference of a series, you will lose an observation.

Recognizing that changes in output are likely to have a distributed-lag effect on unemployment–not all of the effect will take place instantaneously–lags are added to the model to produce:

$$\Delta u_t = \alpha + \beta_0 g_t + \beta_1 g_{t-1} + \beta_2 g_{t-2} + \cdots + \beta_q g_{t-q} + e_t \tag{9.4}$$



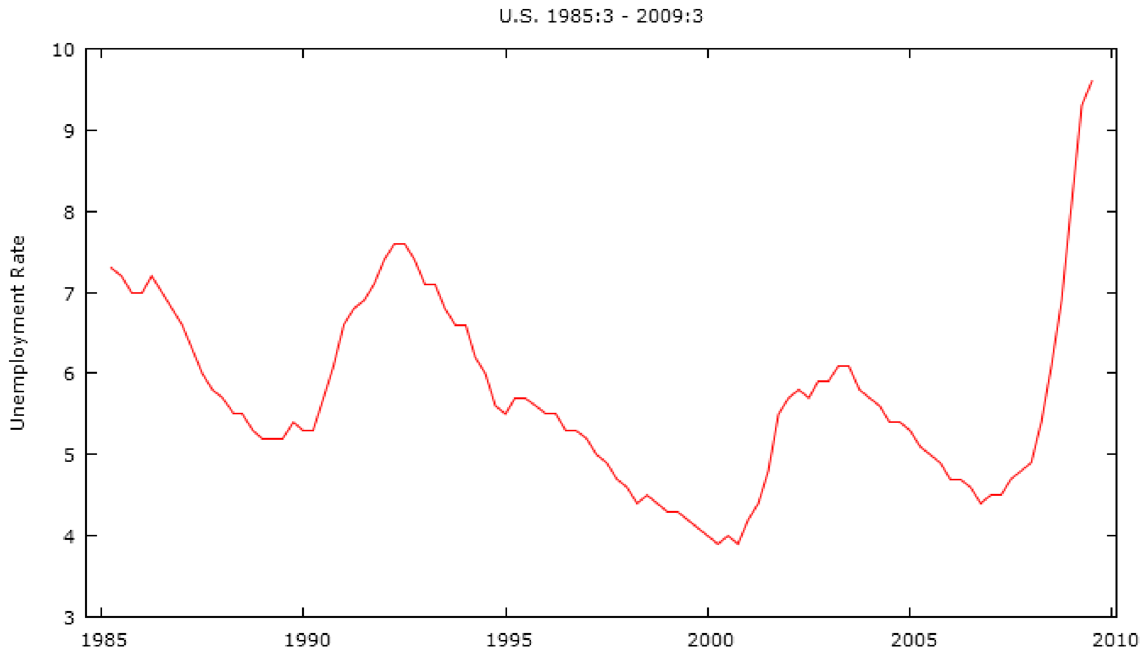


Figure 9.3: Time-Series graphs of Okun data

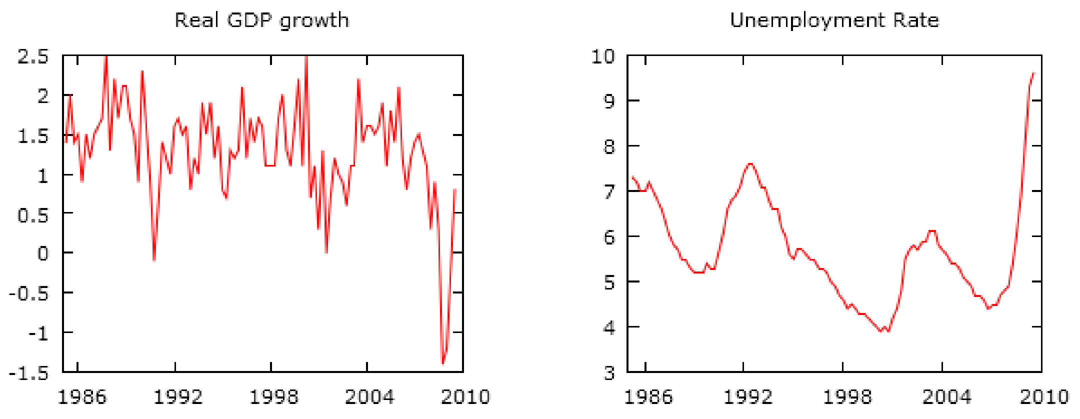


Figure 9.4: Multiple time-series graphs of Okun data produced using `View>Multiple graphs>Time-series`. This uses the `scatters` command.

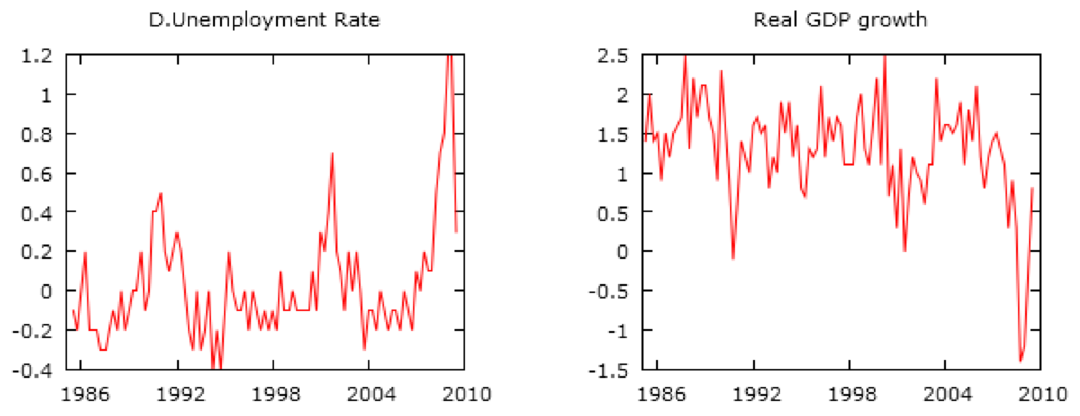


Figure 9.5: Change in unemployment and real GDP growth. This uses the `scatters` command.

The differences of the unemployment rate are taken and the series plotted in Figure 9.5 below. and this will produce a single graph that looks like those in Figure 9.4 of *POE4*. To estimate a finite distributed lag model in **gretl** is quite simple using the lag operators. Letting $q = 3$ and

```
diff u
ols d_u const g(0 to -3)
```

This syntax is particularly pleasing. First, the `diff` *varname* function is used to add the first difference of any series that follow; the new series is called `d_`*varname*. Next, the contemporaneous and lagged values of `g` can be succinctly written `g(0 to -3)`. That tells **gretl** to use the variable named `g` and to include `g`, $\mathtt{g}_{t-1}$, $\mathtt{g}_{t-2}$, and $\mathtt{g}_{t-3}$. When the lagged values of `g` are used in the regression, they are actually being created and added to the dataset. The names are `g_`***`number`***. The number after the underline tells you the lag position. For instance, `g_2` is `g` lagged two time periods. The new variables are given ID numbers and added to the variable list in the main **gretl** window as shown in Figure 9.6.

The regression output that uses the new variables is:

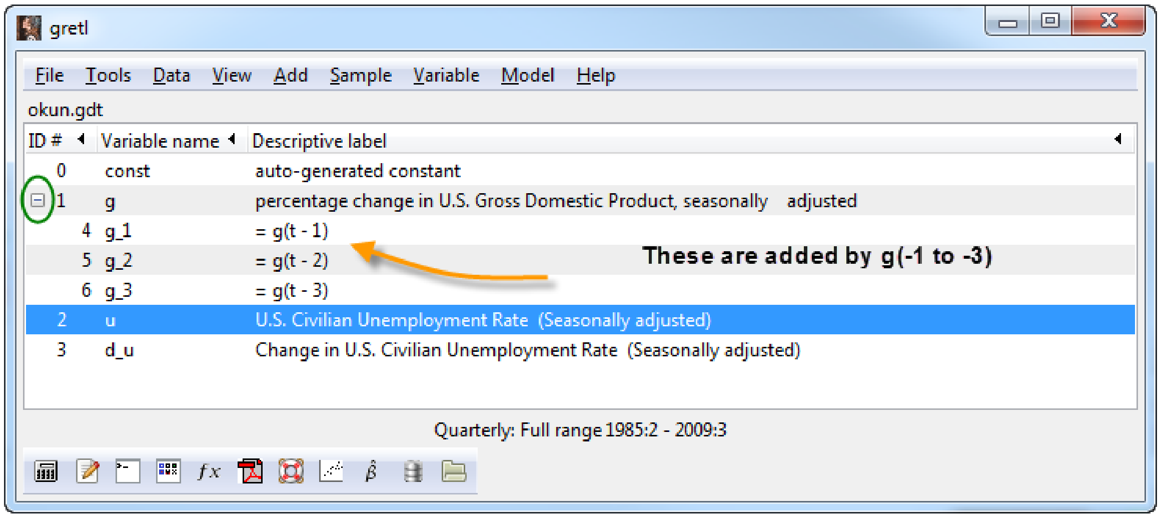
OLS, using observations 1986:1–2009:3 (T = 95)
Dependent variable: d_u

	Coefficient	Std. Error	t-ratio	p-value
const	0.580975	0.0538893	10.7809	0.0000
g	−0.202053	0.0330131	−6.1204	0.0000
g_1	−0.164535	0.0358175	−4.5937	0.0000
g_2	−0.0715560	0.0353043	−2.0268	0.0456
g_3	0.00330302	0.0362603	0.0911	0.9276

Mean dependent var	0.027368	S.D. dependent var	0.289329
Sum squared resid	2.735164	S.E. of regression	0.174329
R^2	0.652406	Adjusted R^2	0.636957
$F(4, 90)$	42.23065	P-value(F)	6.77e–20
Log-likelihood	33.71590	Akaike criterion	−57.43179
Schwarz criterion	−44.66241	Hannan–Quinn	−52.27200
$\hat{\rho}$	0.358631	Durbin–Watson	1.274079

Notice that the t-ratio on `g_3` is not significantly different from zero at 10%. We drop it and reestimate the model with only 2 lagged values of `g`. For comparison, the sample is held constant.

```
smpl 1986:1 2009:3
ols d_u const g(0 to -2)
```


gretl
File Tools Data View Add Sample Variable Model Help
okun.gdt

ID #	Variable name	Descriptive label
0	const	auto-generated constant
1	g	percentage change in U.S. Gross Domestic Product, seasonally adjusted
4	g_1	= g(t - 1)
5	g_2	= g(t - 2)
6	g_3	= g(t - 3)
2	u	U.S. Civilian Unemployment Rate (Seasonally adjusted)
3	d_u	Change in U.S. Civilian Unemployment Rate (Seasonally adjusted)

These are added by g(-1 to -3)
Quarterly: Full range 1985:2 - 2009:3

Figure 9.6: Notice that the lagged variables used in the model are added to the list of available series. They also receive ID numbers.

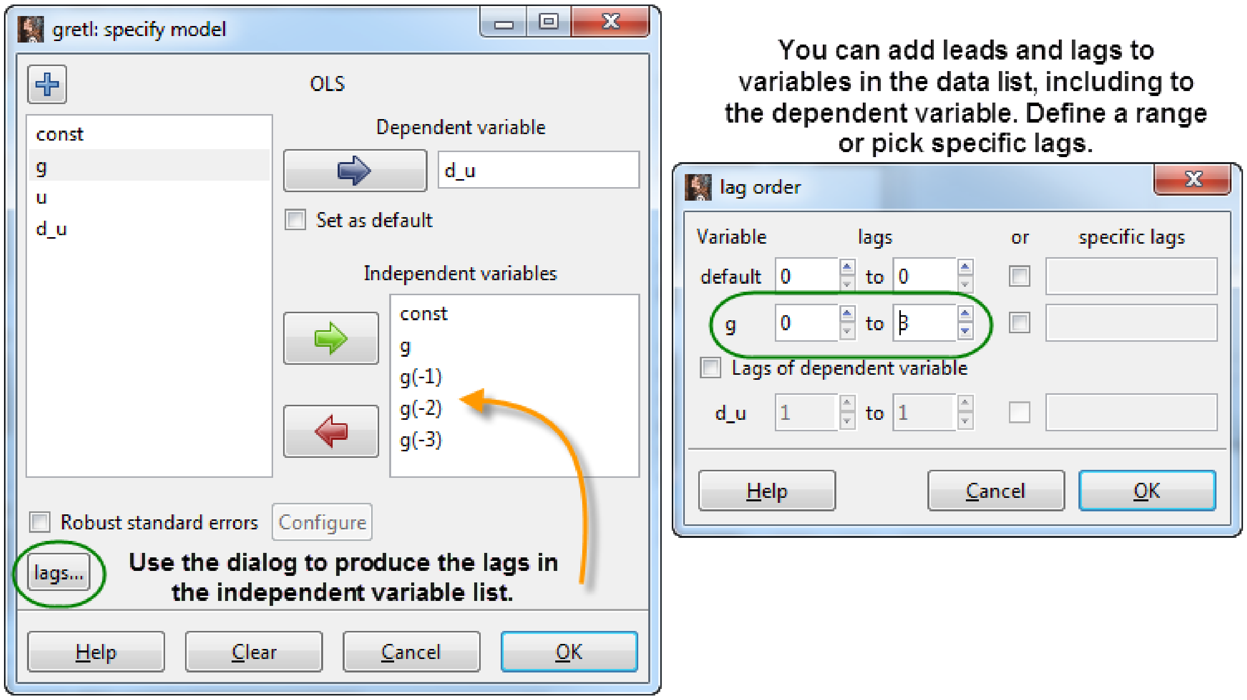
The *AIC* reported by **gretl** has fallen to -59.42303, indicating a marginal improvement in the model.

If you are using the GUI rather than a **gretl** script to estimate the model, you have the opportunity to create the lagged variables through a dialog box. The specify model dialog and the lag order dialog are shown in Figure 9.7 below.

9.4 Serial Correlation

The multiple linear regression model of equation (5.1) assumes that the observations are not correlated with one another. While this is certainly believable if one has drawn a random sample, it's less likely if one has drawn observations sequentially in time. Time series observations, which are drawn at regular intervals, usually embody a structure where time is an important component. If you are unable to completely model this structure in the regression function itself, then the remainder spills over into the unobserved component of the statistical model (its error) and this causes the errors be correlated with one another.

One way to think about it is that the errors will be **serially correlated** when omitted effects last more than one time period. This means that when the effects of an economic 'shock' last more than a single time period, the unmodeled components (errors) will be correlated with one another. A natural consequence of this is that the more frequently a process is sampled (other things being equal), the more likely it is to be autocorrelated. From a practical standpoint, monthly observations are more likely to be autocorrelated than quarterly observations, and quarterly more likely than yearly ones. Once again, ignoring this correlation makes least squares inefficient at best and the usual measures of precision (standard errors) inconsistent.



Dependent variable: d_u

Independent variables: const, g, g(-1), g(-2), g(-3)

lag order: default 0 to 0; g 0 to 3; d_u 1 to 1

You can add leads and lags to variables in the data list, including to the dependent variable. Define a range or pick specific lags.

Use the dialog to produce the lags in the independent variable list.

Figure 9.7: The OLS specify model dialog box has a button that brings up a dialog to specify lag order. Once entered the new lagged variables show up in the list of independent variables.

9.4.1 Serial Correlation in a Time-Series

To gain some visual evidence of autocorrelation you can plot the series against its lagged values. If there is serial correlation, you should see some sort of positive or negative relationship between the series. Below (Figure 9.8) is the plot of Real GDP growth against its lagged value. A least squares fit is plotted to show the general orientation of the linear relationship. The series itself certainly appears to be serially correlated.

Other evidence can be obtained by looking at the correlogram. A **correlogram** is simply a plot of a series' sample autocorrelations. The kth order sample autocorrelation for a series y is the correlation between observations that are k periods apart. The formula is

$$r_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2} \tag{9.5}$$

In **gretl** the correlogram plots a number of these against lags. The syntax to plot 12 autocorrelations of the series g is

```
corrgm g 12
```

which yields the plot in Figure 9.9. The correlogram is the plot at the top and the partial auto-

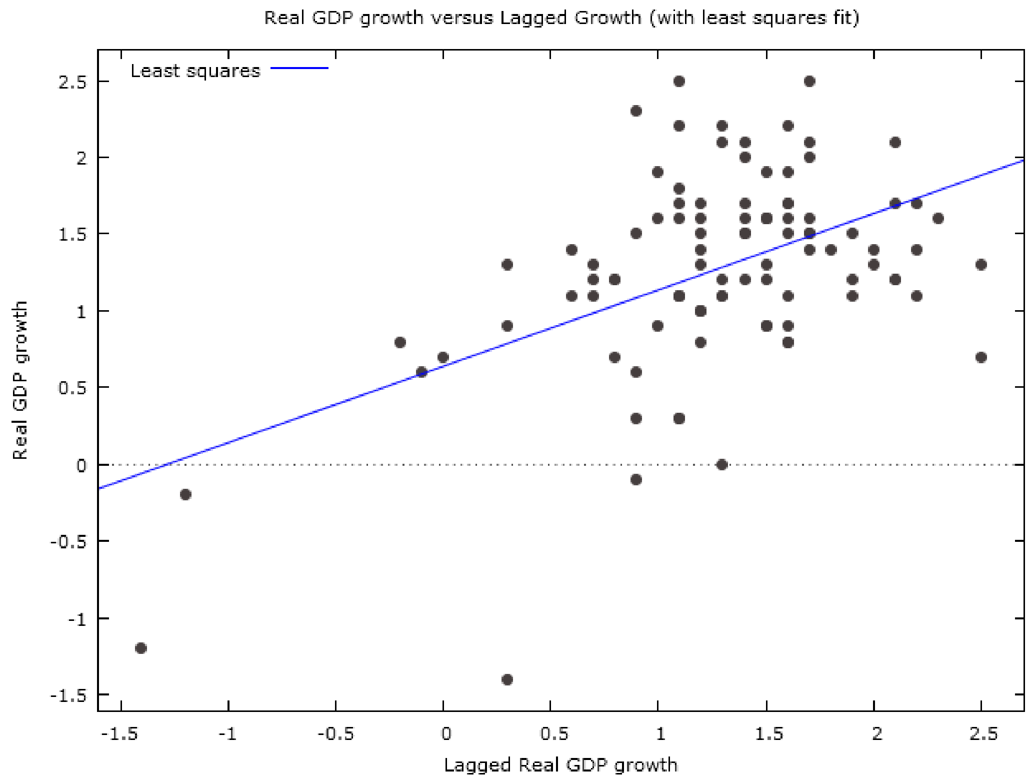


Figure 9.8: This plot shows the relationship between GDP growth vs. lagged growth.

correlations are printed in the bottom panel. Approximate 95% confidence intervals are plotted to indicate which are statistically significant at 5%.

Approximate 95% confidence bands are computed using the fact that $\sqrt{T}r_k \sim N(0,1)$. These can be computed manually using the fact that the `corrgm` function actually generates a matrix return. The script to generate the intervals is

```
matrix ac = corrgm(g, 12)
matrix lb = ac[,1]-1.96/sqrt($nobs)
matrix ub = ac[,1]+1.96/sqrt($nobs)
matrix all = lb~ac[,1]~ub
colnames(all, "Lower  AC Upper ")
printf "\nAutocorrelations and 95%% confidence intervals\n %9.4f\n", all

```

The intervals so generated are:

```
Autocorrelations and 95% confidence intervals
     Lower       AC    Upper
```

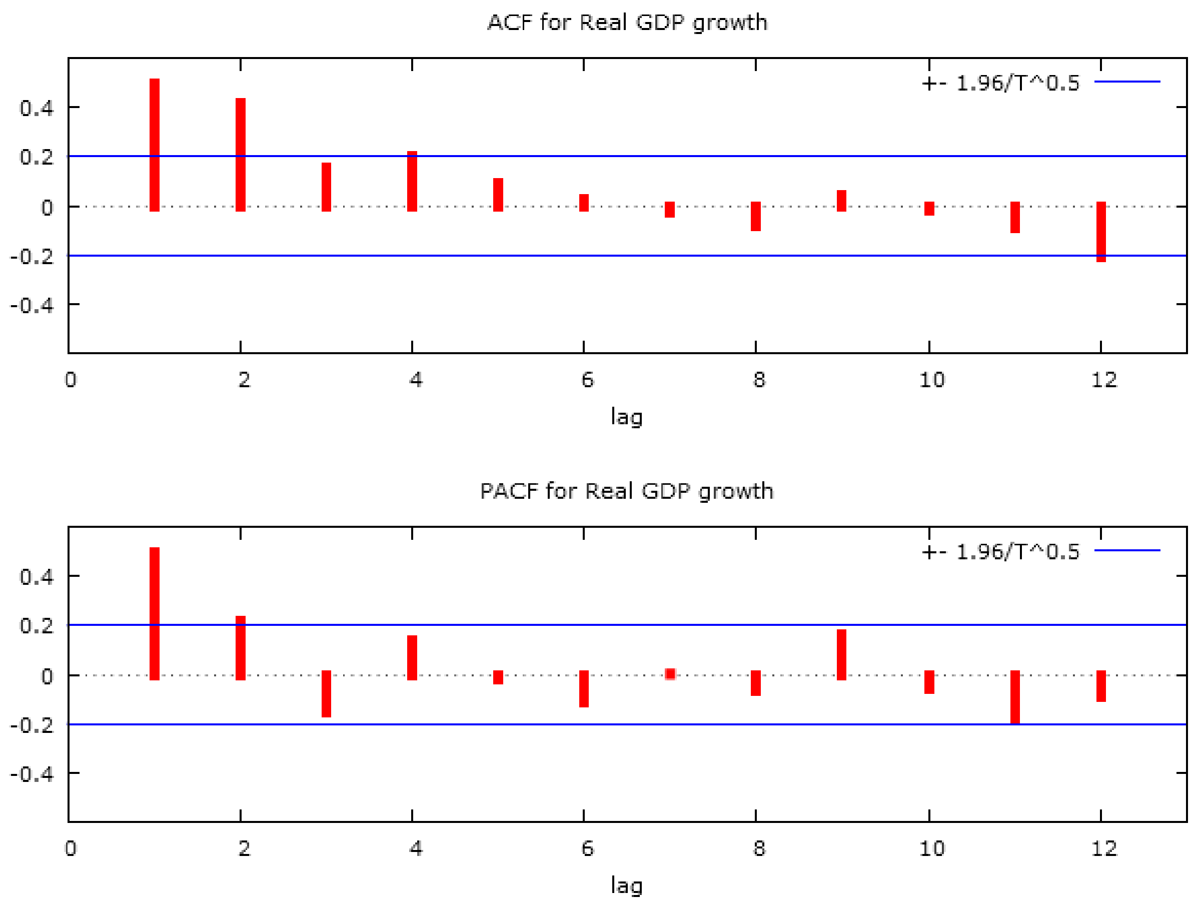


Figure 9.9: The 12 period correlogram for U.S. GDP growth.

```
 0.296     0.494     0.692
 0.213     0.411     0.609
-0.044     0.154     0.352
 0.002     0.200     0.398
-0.108     0.090     0.288
-0.174     0.024     0.222
-0.228    -0.030     0.168
-0.280    -0.082     0.116
-0.154     0.044     0.242
-0.219    -0.021     0.177
-0.285    -0.087     0.111
-0.402    -0.204    -0.006
```

The matrix `ac` holds the autocorrelations in the first column and the partial autocorrelations in the second. The matrices `lb`, `ub`, and `all` use indexing to use all rows of the first column of ac, i.e., `ac[,1]`. This was be dressed up a bit by adding colnames function to add the column names to the matrix.

You can see that zero is not included in the 1st, 2nd, 4th, and last interval. Those are significantly different from zero at 5% level.

The correlogram can be useful for detecting the order of autocorrelation. A long series of declining autocorrelations with a single significant pacf is often an indication of a short order autocorrelation process. See *POE4* for more guidance.

9.4.2 Serial Correlation in Residuals

The correlogram can also be used to check whether the assumption that model errors have zero covariance–an important assumption in the proof of the Gauss-Markov theorem. The example that illustrates this is based on the Phillips curve that relates inflation and unemployment. The data used are from Australia and reside in the *phillips_aus.gdt* dataset.

The model to be estimated is

$$inf_t = \beta_1 + \beta_2 \Delta u_t + e_t \tag{9.6}$$

The data are quarterly and begin in 1987:1. A time-series plot of both series is shown below in Figure 9.10. The graphs show some evidence of serial correlation in both series.

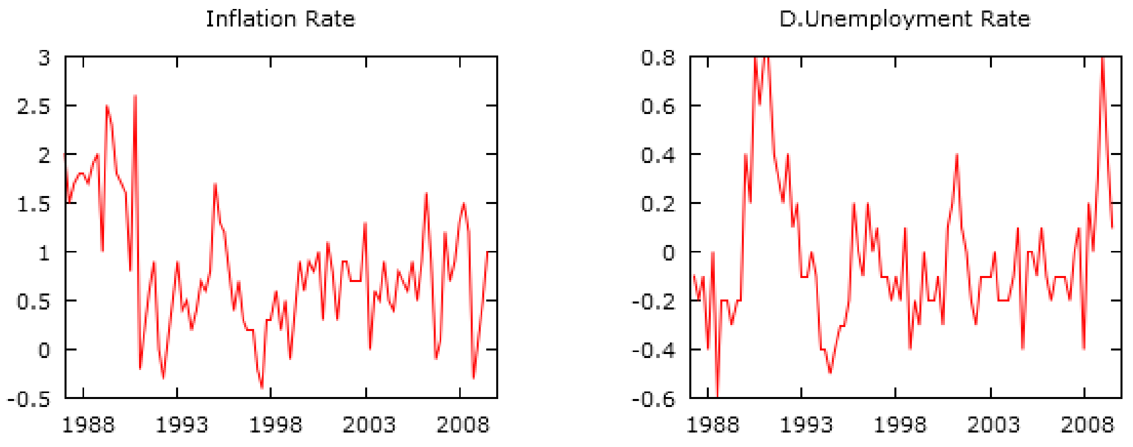


Figure 9.10: This plot shows the relationship between inflation and the change in unemployment in Australia, 1987:1 - 2009:3.

The model is estimated by least squares and the residuals are plotted against time. These appear in Figure 9.11. A correlogram of the residuals that appears below seems to confirm this. To generate the regression and graphs is simple. The script to do so is:

```
ols inf const d_u
series ehat = $uhat
gnuplot ehat --time-series
corrgm ehat
```

Unfortuantely, **gretl** will not accept the accessor, `$uhat`, as an input into either `gnuplot` or `corrgm`. That means you have to create a series, `ehat`, first. Once this is created, both functions work as expected.

The GUI is even easier in this instance once the model is estimated. The model window offers a way to produce both sets of graphs. Simply choose `Graphs>Residual plot>Against time` to produce the first. The second is `Graphs>Residual correlogram`. The latter opens a dialog box allowing you to specify how many autocorrelations to compute. In this example, I set it to 12.

9.5 Another Test for Autocorrelation

Another way to determine whether or not your residuals are autocorrelated is to use an *LM* (Lagrange multiplier) test. For autocorrelation, this test is based on an auxiliary regression where lagged least squares residuals are added to the original regression equation. If the coefficient on the lagged residual is significant then you conclude that the model is autocorrelated. So, for a regression model $y_t = \beta_1 + \beta_2 x_t + e_t$ the first step is to estimate the parameters using least squares and save the residuals, $\hat{e}_t$. An auxiliary regression model is formed using $\hat{e}_t$ as the dependent variable and original regressors and the lagged value $\hat{e}_{t-1}$ as an independent variables. The resulting auxiliary regression is

$$\hat{e}_t = \beta_1 + \beta_2 x_t + \rho \hat{e}_{t-1} + v_t \tag{9.7}$$

Now, test the hypothesis $\rho = 0$ against the alternative that $\rho \neq 0$ and you are done. The test statistic is NR^2 from this regression which will have a χ^2_1 if H_0 : is true. The script to accomplish this is:

```
ols ehat const d_u ehat(-1)
scalar TR2 = $trsq
pvalue X 1 TR2
```

Estimating the statistic in this way causes the first observation to be dropped (since $\hat{e}_0$ is not observed. The result for the *phillips_aus* data reveal

```
Chi-square(1): area to the right of 27.6088 = 1.48501e-007
(to the left: 1)
```

The no autocorrelation null hypothesis is clearly rejected at any reasonable level of significance.

Gretl also includes a model test function that does the same thing. To use it, estimate the model of interest and then use `modtest 1 --autocorr` as shown here:

```
ols inf const d_u --quiet
modtest 1 --autocorr
```

The print out from the `modtest` is fairly extensive as shown here:

```
Breusch-Godfrey test for first-order autocorrelation
OLS, using observations 1987:2-2009:3 (T = 90)
Dependent variable: uhat

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const      -0.00216310   0.0551288    -0.03924   0.9688
  d_u        -0.151494     0.193671     -0.7822    0.4362
  uhat_1      0.558784     0.0900967     6.202     1.82e-08 ***

  Unadjusted R-squared = 0.306582

Test statistic: LMF = 38.465381,
with p-value = P(F(1,87) > 38.4654) = 1.82e-008

Alternative statistic: TR^2 = 27.592347,
with p-value = P(Chi-square(1) > 27.5923) = 1.5e-007

Ljung-Box Q' = 28.0056,
with p-value = P(Chi-square(1) > 28.0056) = 1.21e-007
```

Before explaining what is reported here, an important difference between the manual method and `modtest` needs to be pointed out. When `modtest` is used to perform this test, it sets $\hat{e}_0 = 0$, which is its expected value. By doing so, it is able to use the complete set of 90 observations in the data. The manual method used only 89. Hence, you'll get slightly different results depending on the size of your sample and the number of lags tested.

The results themselves are relevant to those found in *POE4*. The first thing to notice is the t-ratio on `uhat_1` is equal to 6.202, significantly different from zero at 5%. Next, the statistic named `LMF` actually performs an F-test of the no autocorrelation hypothesis based upon the regression. With only one autocorrelation parameter this is equivalent to the square of the t-ratio. The next test is the LM test, i.e., TR^2 from the auxiliary regression. **Gretl** also computes a Ljung-Box Q statistic whose null hypothesis is no autocorrelation. It is also insignificant at the 5% level. These results match those in *POE4* exactly.

If you prefer to use the dialogs, then estimate the model using least squares in the usual way (`Model>Ordinary least squares`). This generates a model window containing the regression results. From this select `Tests>Autocorrelation` to reveal a dialog box that allows you to choose the number of lagged values of $\hat{e}_t$ to include as regressors in the auxiliary regression. Choose the number of lagged values of $\hat{e}_t$ you want to include (in our case 4) and click **OK**. This will give you the same result as the script. The result appears in Figure 9.12. Note, the first statistic reported is simply the joint test that all the lagged values of $\hat{e}$ you included in auxiliary are jointly zeros. The second one is the TR^2 version of the test done in the script. This example shows the relative strength of the *LM* test. One can use it to test for any order of autocorrelation. Other tests, like that of Durbin and Watson discussed later, are more difficult to do in higher orders. The LM test is also robust to having lag(s) of the dependent variable as a regressor.

9.6 Estimation with Serially Correlated Errors

In this section, several methods of estimating models with serially correlated errors will be explored. We will use least squares with robust standard errors to estimate regression models with serial correlation in the errors. We also consider the nonlinear least squares estimator of the model and a more general strategy for estimating models with serially correlation. In the appendix to this chapter, you will find some traditional estimators of this model as well.

9.6.1 Least Squares and HAC Standard Errors

As is the case with heteroskedastic errors, there is a statistically valid way to use least squares when your data are autocorrelated. In this case you can use an estimator of standard errors that is robust to both heteroskedasticity and autocorrelation. This estimator is sometimes called **HAC**, which stands for **heteroskedasticity autocorrelated consistent**. This and some issues that surround its use are discussed in the next few sections.

9.6.2 Bandwidth and Kernel

HAC is not quite as automatic as the heteroskedasticity consistent (HCCME) estimator in chapter 8. To be robust with respect to autocorrelation you have to specify how far away in time the autocorrelation is likely to be significant. Essentially, the autocorrelated errors over the chosen time window are averaged in the computation of the HAC standard errors; you have to specify how many periods over which to average and how much weight to assign each residual in that average. The language of time-series analysis can often be opaque. This is the case here. The weighted average is called a **kernel** and the number of errors to average in this respect is called **bandwidth**. Just think of the kernel as another name for weighted average and bandwidth as the term for number of terms to average.

Now, what this has to do with **gretl** is fairly simple. You get to pick a method of averaging (Bartlett kernel or Parzen kernel) and a bandwidth (`nw1`, `nw2` or some integer). **Gretl** defaults to the Bartlett kernel and the bandwidth $nw1 = 0.75 \times N^{1/3}$. As you can see, the bandwidth `nw1` is computed based on the sample size, N. The `nw2` bandwidth is $nw2 = 4 \times (N/100)^{2/9}$. This one appears to be the default in other programs like EViews.

Implicity there is a trade-off to consider. Larger bandwidths reduce bias (good) as well as precision (bad). Smaller bandwidths exclude more relevant autocorrelations (and hence have more bias), but use more observations to compute the overall covariance and hence increase precision (smaller variance). The general principle is to choose a bandwidth that is large enough to contain the largest autocorrelations. The choice will ultimately depend on the frequency of observation and the length of time it takes for your system to adjust to shocks.

The bandwidth or kernel can be changed using the `set` command from the console or in a script. The `set` command is used to change various defaults in **gretl** and the relevant switches for our use are `hac_lag` and `hac_kernel`. The use of these is demonstrated below. The following script changes the kernel to `bartlett` and the bandwidth to `nw2`. Then the differences of the unemployment rate are generated. The Phillips curve is estimated by OLS using the ordinary covariance estimator and then by the HAC estimator. The results are collected in a model table.

```
open "@gretldir\data\poe\phillips_aus.gdt"
set hac_kernel bartlett
set hac_lag nw2
diff u
ols inf const d_u
modeltab add
ols inf const d_u --robust
modeltab add
modeltab show
```

The results from the model table are

OLS estimates
Dependent variable: inf

	(OLS)	(OLS w/HAC)
const	0.7776^{**}	0.7776^{**}
	(0.06582)	(0.1018)
d_u	-0.5279^{**}	-0.5279^{*}
	(0.2294)	(0.3092)
n	90	90
R^2	0.0568	0.0568
ℓ	-83.96	-83.96

Standard errors in parentheses
* indicates significance at the 10 percent level
** indicates significance at the 5 percent level
HAC: bandwidth 3 – Bartlett kernel

You can see that the HAC standard errors are quite a bit larger than the usual (and inconsistent) ones. Once **gretl** recognizes that your data are time-series, then the `--robust` option will automatically apply the HAC estimator of standard errors with the default values of the kernel and bandwidth (or the ones you have set with the `set` command).

Notice that the standard errors computed using HAC are a little different from those in Hill et al. (2011). No worries, though. They are statistically valid and suggest that EViews and **gretl** are doing the computations a bit differently.

9.6.3 Nonlinear Least Squares

Perhaps the best way to estimate a linear model that is autocorrelated is using **nonlinear least squares**. As it turns out, the nonlinear least squares estimator only requires that the errors be stable (not necessarily stationary). The other methods commonly used make stronger demands on the data, namely that the errors be covariance stationary. Furthermore, the nonlinear least squares estimator gives you an unconditional estimate of the autocorrelation parameter, ρ, and yields a simple t-test of the hypothesis of no serial correlation. Monte Carlo studies show that it performs well in small samples as well. So with all this going for it, why not use it?

The biggest reason is that nonlinear least squares requires more computational power than linear estimation, though this is not much of a constraint these days. Also, in **gretl** it requires an extra step on your part. You have to type in an equation for **gretl** to estimate. This is the way one works in EViews and other software by default, so the burden here is relatively low.

Nonlinear least squares (and other nonlinear estimators) use numerical methods rather than analytical ones to find the minimum of your sum of squared errors objective function. The routines that do this are iterative. You give the program a good first guess as to the value of the parameters and it evaluates the sum of squares function at this guess. The program looks at the slope of your sum of squares function at the guess, points you in a direction that leads closer to smaller values of the objective function, and computes a *step* in the parameter space that takes you some distance toward the minimum (further down the hill). If an improvement in the sum of squared errors function is found, the new parameter values are used as the basis for another step. Iterations continue until no further significant reduction in the sum of squared errors function can be found.

In the context of the area response equation the AR(1) model is

$$inf_t = \beta_1(1-\rho) + \beta_2(\Delta u_t - \rho\,\Delta u_{t-1}) + \rho\, inf_{t-1} + v_t \tag{9.8}$$

The errors, v_t, are random and the goal is to find β_1, β_2, and ρ that minimize $\sum v_t^2$. Ordinary least squares is a good place to start in this case. The OLS estimates are consistent so we'll start our numerical routine there, setting ρ equal to zero. The **gretl** script to do this follows:

```
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
ols inf const d_u --quiet

scalar beta1 = $coeff(const)
scalar beta2 = $coeff(d_u)
scalar rho = 0

nls inf = beta1*(1-rho) + rho*inf(-1) + beta2*(d_u-rho*d_u(-1))
    params rho beta1 beta2
end nls
```

Magically, this yields the same result from your text!

The `nls` command is initiated with `nls` followed by the equation representing the systematic portion of your model. The command is closed by the statement `end nls`. If possible, it is always a good idea to supply analytical derivatives for nonlinear maximization. In this case I did not, opting to let **gretl** take numerical derivatives. When using numerical derivatives, the `params` statement is required in order for **gretl** to figure out what to take the derivatives with respect to. In the script, I used **gretl**'s built in functions to take differences and lags. Hence, `inf(-1)` is the variable `inf` lagged by one period (-1). In this way you can create lags or leads of various lengths in your **gretl** programs without explicitly having to create new variables via the `generate` or `series` command. The results of nonlinear least squares appear below in Figure 9.13.

9.6.4 A More General Model

Equation 9.8 can be expanded and rewritten in the following way:

$$\mathit{inf}_t = \beta_1(1-\rho) + \beta_2\Delta u_t - \beta_2\rho\,\Delta u_{t-1} + \rho\,\mathit{inf}_{t-1} + v_t \tag{9.9}$$

$$\mathit{inf}_t = \delta + \delta_0\Delta u_t - \delta_1\Delta u_{t-1} + \theta\,\mathit{inf}_{t-1} + v_t \tag{9.10}$$

Both equations contain the same variables, but Equation (9.8) contains only 3 parameters while (9.10) has 4. This means that (9.8) is *nested* within (9.10) and a formal hypothesis test can be performed to determine whether the implied restriction holds. The restriction is $\delta_1 = -\theta_1\delta_0$.[1] To test this hypothesis using **gretl** you can use a variant of the statistic (6.2) discussed in section 6.1. You'll need the restricted and unrestricted sum of squared errors from the models. The statistic is

$$J \times F = \frac{(SSE_r - SSE_u)}{SSE_u/(N-K)} \dot{\sim} \chi^2_J \qquad \text{if } H_0:\ \delta_1 = -\theta_1\delta_0 \text{ is true} \tag{9.11}$$

[1] $\delta = \beta_1(1-\rho), \delta_0 = \beta_2, \delta_1 = -\rho\beta_2, \theta_1 = \rho$

Since $J = 1$ this statistic has an approximate χ^2_1 distribution and it is equivalent to an F test. Note, you will get a slightly different answer than the one listed in your text. However, rest assured that the statistic is asymptotically valid.

For the example, we've generated the output:

```
Chi-square(1): area to the right of 0.112231 = 0.737618
(to the left: 0.262382)

F(1, 85): area to the right of 0.112231 = 0.738443
(to the left: 0.261557)
```

Because the sample is relatively large the p-values from the F(1,85) and the χ^2_1 are very close to one another. Neither is significant at the 5% level.

The estimated model is:

OLS, using observations 1987:3–2009:3 ($T = 89$)
Dependent variable: inf

	Coefficient	Std. Error	t-ratio	p-value
const	0.333633	0.0899028	3.7110	0.0004
d_u	−0.688185	0.249870	−2.7542	0.0072
d_u_1	0.319953	0.257504	1.2425	0.2175
inf_1	0.559268	0.0907962	6.1596	0.0000

Mean dependent var	0.783146	S.D. dependent var	0.635902
Sum squared resid	23.16809	S.E. of regression	0.522078
R^2	0.348932	Adjusted R^2	0.325953
$F(3, 85)$	15.18488	P-value(F)	5.37e–08
Log-likelihood	−66.39473	Akaike criterion	140.7895
Schwarz criterion	150.7440	Hannan–Quinn	144.8019
$\hat{\rho}$	−0.149981	Durbin's h	−2.685227

Notice how **gretl** refers to the parameters–by their variable names. This is possible because the model is linear and there is no ambiguity. Also, Δu_{t-1} is referred to as `d_u_1`. It can get a little confusing, but `d_u` is the difference and the lag has the usual `_1` suffix.

The lagged unemployment rate has a t-ratio of 1.243. It is not significant and it may be worth considering removing it from the model using the `omit d_u(-1)` statement.

You can also compare nonlinear combinations of parameters from the equations (9.8) and (9.10). To do so you can use **gretl** to compute the relevant scalars and print them to the screen as shown below in the script:

```
nls inf = beta1*(1-rho) + rho*inf(-1) + beta2*(d_u-rho*d_u(-1))
    params rho beta1 beta2
end nls
scalar delta = $coeff(beta1)*(1-$coeff(rho))
scalar delta1 = -$coeff(rho)*$coeff(beta2)
printf "\nThe estimated delta is %.3f and the \
estimated delta1 is %.3f.\n",delta,delta1
```

In lines 4 and 5 δ and δ_1 are approximated from the NLS estimated AR(1) regression. the result is

```
The estimated delta is 0.337 and the estimated delta1 is 0.387.
```

You can see that these values are actually fairly close to the ones estimated in the unrestricted model, which were 0.334 and 0.320, respectively. Also, $\hat{\beta}_2$ is similar to $\hat{\delta}_1$ and $\hat{\rho}$ is similar to $\hat{\theta}$. It is no wonder that the hypothesis restrictions are not rejected statistically.

9.7 Autoregressive Distributed Lag Models

A model that combines finite distributed lags and is autoregressive is considered. This is the so-called autoregressive distributed lag model (ARDL). The ARDL(p,q) model has the general form

$$y_t = \delta + \theta_1 y_{t-1} + \cdots + \theta_p y_{t-p} + \delta_0 x_t + \delta_1 x_{t-1} + \cdots + \delta_q x_{t-q} + v_t \tag{9.12}$$

As regressors, it has p lags of the dependent variable, y_t, and q lags of the independent variable, x_t.

9.7.1 Phillips Curve

The ARDL(1,1) and ARDL(1,0) models of inflation can be estimated using least squares. The two models of the Phillips curve

OLS estimates
Dependent variable: inf

	(1)	(2)
const	0.3336**	0.3548**
	(0.08990)	(0.08760)
inf_1	0.5593**	0.5282**
	(0.09080)	(0.08508)
d_u	−0.6882**	−0.4909**

	(0.2499)	(0.1921)
d_u_1	0.3200	
	(0.2575)	
n	89	90
$\bar{R}^2$	0.3260	0.3314
ℓ	-66.39	-67.45

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

Choosing between these models can be done in several ways. First, if the t-ratio on Δu_{t-1} is insignificant, then the evidence suggests that omitting it may not adversely impact the properties of the least squares estimator of the restricted model. it is not significant in this case and you may consider dropping it from the model.

Another possibility is to use one of the model selection rules discussed in chapter 6. Recall that we wrote a function called `modelsel` that computes the *AIC* and *SC* model selection rules. Here, the program is modified slightly by omitting the display of the adjusted R^2. Refer to chapter 6 for more details on the program structure in **gretl**.

To choose between the ARDL(1,1) and ARDL(1,0) using the *AIC* or *SC* create and run the following function called `modelsel`.

```
function matrix modelsel (series y, list xvars)
    ols y xvars --quiet
    scalar sse = $ess
    scalar N = $nobs
    scalar K = nelem(xvars)
    scalar aic = ln(sse/N)+2*K/N
    scalar bic = ln(sse/N)+K*ln(N)/N
    matrix A = { K, N, aic, bic}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "K = %d, N = %d, AIC = %.4f SC = %.4f.\n",K,N,aic,bic
    return A
end function
```

Then, we can form variable lists and use the function to compare two models:

```
list x = const inf(-1) d_u(0 to -1)
matrix a = modelsel(inf,x)
```

```
list x = const inf(-1) d_u(0)
matrix b = modelsel(inf,x)
```

This yields

```
Regressors: const,inf_1,d_u,d_u_1
K = 4, N = 91, AIC = -1.2802 SC = -1.1698.

Regressors: const,inf_1,d_u
K = 3, N = 91, AIC = -1.2841 SC = -1.2013.
```

The smaller model ($K = 3$) has a smaller AIC and SC and it is preferred.

We could also search over a wider range of models using loops. Searching over $p = 1, 2, \ldots 6$ and $q = 0, 1$ is done in the next section of code. Admittedly, this is a little clumsy in that formulating a set of nested loops for this setup is not straightforward in **gretl** due to the fact that it cannot recognize variables like `inf(0 to 0)`. This causes one to have to hard code certain parts and use a series of if statements to control the construction of the variable lists.[2] The code to search over this set is:

```
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
smpl 1988:3 2009:3
matrix A = {}
scalar q = 0
loop p = 1..6 --quiet
        if p = 1
            list x = const inf(-1) d_u
        else
            list x = const inf(-1 to -p) d_u
        endif
        matrix a = p~q~modelsel(inf,x)
        matrix A = A | a
        modelsel(inf,x)
    endloop
scalar q = 1
loop p = 1..6 --quiet
        if p = 1
            list x = const inf(-1) d_u(0 to -1)
        else
            list x = const inf(-1 to -p) d_u(0 to -1)
        endif
        matrix a = p~q~modelsel(inf,x)
        matrix A = A | a
```

[2]I'm still working on a more elegant solution. Stay tuned for future editions to see if I succeed.

```
25     endloop
26 colnames(A,"p q K N AIC SC ")
27 print A
```

The data are loaded and the differences of unemployment are generated using the `diff` command. Then, the sample is limited to 1988:3 - 2009:3 in order to get the same results as found in Table 9.4 of *POE4*. An empty matrix `A` is created. This matrix will be used to collect results of the `modelsel` command. To do this, the row vectors created by `modelsel` will be vertically concatenated. That means as a new row will be appended below existing rows. If the matrix starts out empty the first row appended becomes the first row!

As I mentioned above, some of the variables are hard coded into the loop. In this example the distributed lag parameter, q, only takes two values, 0 and 1. In the first loop q is hard coded to be equal to zero. So, the loop is executed with the variable Δu_t permanently in the variable list named `x`.

The loop itself loops over the parameter `p`, which starts at 1 increments to 6. When `p=1`, the syntax `inf(-1 to -1)` fails so we must tell **gretl** to construct the variable list x with inf(-1) when `p=1`. Otherwise we can construct the variable list using `inf(-1 to -$p)`.

In line 12 a row vector is created that includes `p`, `q`, and the results from `modelsel`. This uses horizontal concatenation via the symbol, `~`. In the next line vertical concatenation is used to stack the new vector of results underneath the existing ones. The loop ends and column names are added to the matrix and printed.

The next loop is nearly identical. The only difference is that `q=1` is hard coded into the script. Notice that `q=1` is fixed as a scalar in line and that `d_u(0 to -1)` replaces `d_u` in the previous loop. So, the code looks complicated, but it can effectively be replicated by a cut and paste with minor editing. In this particular script, `p` and `q` are actually numbers that work in this loop construct. Hence, there is no need to use the string prefix, $ (although if used in lines 10 and 12 this will work as well).

That is a lot of code, but the output is nice:

```
            p            q            K            N          AIC           SC
       1.0000       0.0000       3.0000       85.000      -1.2466      -1.1604
       2.0000       0.0000       4.0000       85.000      -1.2905      -1.1755
       3.0000       0.0000       5.0000       85.000      -1.3352      -1.1915
       4.0000       0.0000       6.0000       85.000      -1.4020      -1.2296
       5.0000       0.0000       7.0000       85.000      -1.3964      -1.1952
       6.0000       0.0000       8.0000       85.000      -1.3779      -1.1480
       1.0000       1.0000       4.0000       85.000      -1.2425      -1.1275
       2.0000       1.0000       5.0000       85.000      -1.2860      -1.1423
       3.0000       1.0000       6.0000       85.000      -1.3233      -1.1509
       4.0000       1.0000       7.0000       85.000      -1.3795      -1.1784
       5.0000       1.0000       8.0000       85.000      -1.3729      -1.1430
```

```
        6.0000      1.0000      9.0000      85.000     -1.3544    -1.0958
```

From this you can see that the ARDL(4,0) minimizes both *AIC* and *SC*. Estimating this model yields,

OLS, using observations 1988:1–2009:3 (T = 87)
Dependent variable: inf

	Coefficient	Std. Error	t-ratio	p-value
const	0.100100	0.0982599	1.0187	0.3114
d_u	−0.790172	0.188533	−4.1912	0.0001
inf_1	0.235440	0.101556	2.3183	0.0230
inf_2	0.121328	0.103757	1.1693	0.2457
inf_3	0.167690	0.104960	1.5977	0.1140
inf_4	0.281916	0.101380	2.7808	0.0067

Sum squared resid	18.23336	S.E. of regression	0.474450
R^2	0.458422	Adjusted R^2	0.424992
$F(5,81)$	13.71262	P-value(F)	1.07e–09
Log-likelihood	−55.47215	Akaike criterion	122.9443
Schwarz criterion	137.7397	Hannan–Quinn	128.9020
$\hat{\rho}$	−0.032772	Durbin's h	−0.903935

Finally, you can check the residuals for autocorrelation using the *LM* test. Here we want to check the model for autocorrelation for up to 5 lags. The easiest way is to put `modtest` into a loop. The underlying regression is an ARDL(1,0). This one wins the model selection derby because the coefficient on Δu_{t-1} was not significant in and ARDL(1,1).

```
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
ols inf inf(-1) d_u const
loop i=1..4
    modtest $i --autocorr --quiet
endloop
```

This is an example of an **index loop**. The index is called `i` and it loops in increments of 1 from 1 to 4. The `modtest` command takes the string argument `$i` at each iteration. The `--quiet` option is used to reduce the copious amount of output this loop will produce. The p-values for the *LM* test, which I've chosen not to include, match the ones Table 9.3 of *POE4*.

9.7.2 Okun's Law

Okun's Law provides another opportunity to search for an adequate specification of the time-series model. Load the *okun.gdt* data. These quarterly data begin at 1985:2. Set the data structure to time-series if needed. In this example, the model search is over $p = 0, 1, 2$ and $q = 1, 2, 3$. There are 12 possible models to consider and loops will again be used to search for the preferred one.

To make the loop simpler, the `modelsel` function has been modified slightly. It now accepts a single variable list as its input. This allows us to place the dependent variable, `x`, and its first lag into the model as `x(0 to -1)`. **Gretl** reads this as `x x(-1)`. Thus, these two regressions would yield the same result

```
ols x const x(-1)
ols x(0 to -1) const
```

Placing the constant at the end of the list only moves its position in the output, it does not change the substance of the results.

The new and improved `modelsel2` appears below:

modelsel2 function useful for ARDL models

```
1  function matrix modelsel2 (list xvars)
2      ols xvars --quiet
3      scalar sse = $ess
4      scalar N = $nobs
5      scalar K = nelem(xvars)-1
6      scalar aic = ln(sse/N)+2*K/N
7      scalar bic = ln(sse/N)+K*ln(N)/N
8      matrix A = { K, N, aic, bic}
9  #   printf "\nDependent variable and Regressors: %s\n",varname(xvars)
10 #   printf "K = %d, N = %d, AIC = %.4f SC = %.4f.\n",K,N,aic,bic
11     return A
12 end function
```

Notice that the input on line one is now just a single variable list. Line 6 is modified by subtracting one from the number of elements in the variable list, since the list now includes the dependent variable. Also, the `printf` statements are commented out to reduce the amount of output sent to the screen. You can remove the # from lines 9 and 10 if you want to see what's in the model and the results at each iteration. Since we are dealing with an ARDL(p, q), p and q tell us exactly which regressors are in the model so these are really not needed in the current context.

The new and improved loop to compute the model selection rules is:

Loop for the Okun Example

```
 1 open "@gretldir\data\poe\okun.gdt"
 2 diff u
 3 smpl 1986:1 2009:3
 4 matrix A = {}
 5 loop p = 0..2  --quiet
 6     loop q = 1..3 --quiet
 7         if p=0
 8             list vars = d_u g(0 to -q) const
 9         else
10             list vars = d_u(0 to -p) g(0 to -q) const
11         endif
12         matrix a = p~q~modelsel2(vars)
13         matrix A = A | a
14     endloop
15 endloop
16 colnames(A,"p q K N AIC SC ")
17 print A
18 function modelsel2 clear
```

This loop improves upon the last in at least one way. It now contains a nest that should function properly for any `p>1` and `q>0`. The first three lines load the data, create the difference of unemployment, and set the sample to match the one used in *POE4*. This script contains two loops, one for `p` and one for `q` that are nested. When loops are nested this way, the `p` loop starts at zero and then the `q` loop iterates from 1 to 3. Once the `q` loop is finished, the `p` loop increments by 1 and the `q` loop starts over again.

The conditional `if` statement is necessary because when `p=0` the statement `d_u(0 to -p)` in line 10 cannot be computed. The last line clears the `modelsel2` function from memory. If you need to modify the function, perform your changes, and re-run it to load it into memory. Once loaded into memory, there is no need to run it again.

The results from this script are

```
          p            q            K            N          AIC           SC
     0.0000       1.0000       3.0000       95.000      -3.4362      -3.3556
     0.0000       2.0000       4.0000       95.000      -3.4634      -3.3559
     0.0000       3.0000       5.0000       95.000      -3.4424      -3.3080
     1.0000       1.0000       4.0000       95.000      -3.5880      -3.4805
     1.0000       2.0000       5.0000       95.000      -3.5675      -3.4331
     1.0000       3.0000       6.0000       95.000      -3.5612      -3.3999
     2.0000       1.0000       5.0000       95.000      -3.5693      -3.4349
     2.0000       2.0000       6.0000       95.000      -3.5483      -3.3870
     2.0000       3.0000       7.0000       95.000      -3.5491      -3.3609
```

The ARDL(1,1) minimizes both *AIC* and *SC*. The estimates for this model are:

OLS, using observations 1985:4–2009:3 ($T = 96$)

Dependent variable: d_u
HAC standard errors, bandwidth 3 (Bartlett kernel)

	Coefficient	Std. Error	t-ratio	p-value
const	0.378010	0.0671726	5.6275	0.0000
g	−0.184084	0.0268375	−6.8592	0.0000
g_1	−0.0991552	0.0388520	−2.5521	0.0124
d_u_1	0.350116	0.0861251	4.0652	0.0001

Mean dependent var	0.025000	S.D. dependent var	0.288736
Sum squared resid	2.422724	S.E. of regression	0.162277
R^2	0.694101	Adjusted R^2	0.684126
$F(3, 92)$	70.24213	P-value(F)	1.04e–23
Log-likelihood	40.39577	Akaike criterion	−72.79155
Schwarz criterion	−62.53415	Hannan–Quinn	−68.64534
$\hat{\rho}$	−0.024372	Durbin's h	−0.437108

9.7.3 Autoregressive Models

An autoregressive model is just a special case of an ARDL(p,q) where $q = 0$. The model only includes lags of the dependent variable.

$$y_t = \delta + \theta_1 y_{t-1} + \theta_2 y_{t-2} \cdots + \theta_p y_{t-p} + v_t \tag{9.13}$$

The example is based on the *okun.gdt* data. An initial AR(2) model is estimated using GDP growth. The possibility of residual autocorrelation is explored using *LM* tests and by looking at the correlogram.

```
open "@gretldir\data\poe\okun.gdt"
ols g(0 to -2) const
series res = $uhat
corrgm res
loop i = 1..4
    modtest $i --autocorr --quiet
endloop
```

The correlogram appears in Figure 9.14 below. Only the autocorrelation at the 12th lag is significant, probably by chance. None of the *LM* statistics computed by the `modtest` loop have p-values smaller than 10%, therefore this model may be properly specified. To see how this compares with others via the model selection rules, we use another loop and the `modesel2` function.

```
open "@gretldir\data\poe\okun.gdt"
smpl 1986:3 2009:3
matrix A = {}
scalar q=0
loop p = 1..5  --quiet
       list vars = g(0 to -p) const
       matrix a = p~q~modelsel2(vars)
       matrix A = A | a
    endloop
colnames(A,"p q K N AIC SC ")
print A
```

The sample was shortened again and the nested loop is removed. Otherwise, this is the same as used to model select in the ARDL(p,q) example. The results

```
p           q          K           N          AIC          SC
1.0000      0.0000     2.0000      93.000    -1.0935     -1.0391
2.0000      0.0000     3.0000      93.000    -1.1306     -1.0489
3.0000      0.0000     4.0000      93.000    -1.1242     -1.0153
4.0000      0.0000     5.0000      93.000    -1.1332    -0.99700
5.0000      0.0000     6.0000      93.000    -1.1117    -0.94827
```

match those in *POE4*. The AR(2) model is supported by the *SC* while the *AIC* chooses one with 4 lags. As mentioned previously, the *SC* criterion imposes a slightly larger penalty for adding regressors and may sometimes leas to smaller models.

9.8 Forecasting

In this section we consider forecasting using 3 different models, an AR model, an ARDL model, and an exponential smoothing model. The examples focus on short-term forecasting, typically up to 3 periods into the future.

9.8.1 Forecasting with an AR model

Suppose that it is the 3rd quarter in 2009 and have estimated the AR(2) model of GDP growth using data up to and including 2009:3. In this section the use of an AR(2) model to forecast the next three periods is discussed and forecast confidence intervals are generated.

The AR(2) model in terms of its unknown coefficients

$$g_t = \delta + \theta_1 g_{t-1} + \theta_2 g_{t-2} + v_t \tag{9.14}$$

Denoting the last sample observation as g_T, the task is to forecast g_{T+1}, g_{T+2}, and g_{T+3}. The value of the next observation beyond the available sample is

$$g_{T+1} = \delta + \theta_1 g_T + \theta_2 g_{T-1} + v_{T+1} \tag{9.15}$$

Growth rates for the 2 most recent quarters are $G_T = G_{2009:3} = 0.8$, and $g_{T-1} = g_{2009:2} = -0.2$, which with the estimated values of the parameters is used to make a forecast of $g_{T+1} = g_{2009:4}$.

$$\begin{aligned}\hat{g}_{T+1} =&\hat{\delta} + \hat{\theta}_1 g_T + \hat{\theta}_2 g_{T-1}\\ =&0.46573 + 0.37700\ \times 0.8 + 0.24624 \times (-0.2)\\ =&0.7181\end{aligned}$$

Once the model is estimated it is easy to compute this forecast.

```
open "@gretldir\data\poe\okun.gdt"
ols g(0 to -2) const --robust --quiet
```

Using this model to forecast in **gretl** is very simple. The main decision you have to make at this point is how many periods into the future you want to forecast. In **gretl** you have to extend the sample to include future periods under study.

9.8.2 Using the Dialogs

Return to the main **gretl** window and choose `Model>Ordinary least squares`. This will bring up the 'specify model' dialog box. Choose `g` as the dependent variable as shown.

Since your data are defined as time-series (recall, you did this through `Data>Dataset structure`) an extra button, labeled 'lags...', appears at the bottom of the dialog. Click the 'lags...' button in the specify model dialog box and the 'lag order' dialog box shown on the right-hand side in Figure 9.7 opens.

Click **OK** and the 3 lagged values of GDP growth are added to the model. Now, click **OK** in the specify model dialog and the model is estimated.

Now, we'll use the dialogs to extend the sample and generate the forecasts. From the model window choose `Analysis>Forecasts`. This opens the 'Add observations' dialog box shown in Figure 9.15. To add three observations change the number in the box to 3. Click **OK** to open the forecast dialog box shown below in Figure 9.16.

By choosing to add 3 observations to the sample, the forecast range is automatically set to 2009:4 to 2010.2. Notice that we've chosen 'automatic forecast (dynamic out of sample).' Click **OK** and the forecast results appear.

A script is actually much simpler. Here is the example in a script.

```
open "@gretldir\data\poe\okun.gdt"
ols g(0 to -2) const
dataset addobs 3
fcast 2009:4 2010:2 --plot=c:\temp\ar2plot.plt
```

In line 3 the `dataset addobs` tells **gretl** to add 3 observations to the dataset. Then, the `fcast` command with the desired dates to forecast are given. The results are:

```
For 95% confidence intervals, t(93, 0.025) = 1.986

     Obs            g    prediction    std. error        95% interval

   2009:4                  0.718079     0.552688    -0.379448 -    1.81561
   2010:1                  0.933435     0.590660    -0.239499 -    2.10637
   2010:2                  0.994452     0.628452    -0.253530 -    2.24243
```

Miraculously, these match those in *POE4*! **Gretl** can optionally use **gnuplot** to plot the time-series and the forecasts (with intervals). The plot is shown in Figure 9.17.[3] The last three observations are forecasts (in blue) and include the 95% confidence intervals shown in green. Actual inflation appears in red. From an economics standpoint, the forecast is depressing, mainly because the intervals are very wide. The 95% interval includes a possible recession.

9.8.3 Exponential Smoothing

Another popular model used for predicting the future value of a variable based on its history is **exponential smoothing**. Like forecasting with an AR model, forecasting using exponential smoothing does not use information from any other variable.

The basic idea is that the forecast for next period is a weighted average of the forecast for the current period and the actual realized value in the current period.

$$\hat{y}_{T+1} = \alpha y_T + (1-\alpha)\hat{y}_T \tag{9.16}$$

The exponential smoothing method is a versatile forecasting tool, but one needs a value for the smoothing parameter α and a value for $\hat{y}_T$ to generate the forecast $\hat{y}_{T-1}$. The value of α can reflect one's judgment about the relative weight of current information; alternatively, it can be estimated from historical information by obtaining **within-sample forecasts**

$$\hat{y}_t = \alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1} \tag{9.17}$$

[3]This graph was generated from the GUI. The `plot` command as shown in the script actually yields only the plots of the forecasted values and their intervals.

and choosing that value of α that minimizes the sum of squares of the **one-step forecast errors**

$$v_t = y_t - \hat{y}_t = y_t - (\alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1}) \tag{9.18}$$

Smaller values of α result in more smoothing of the forecast. **Gretl** does not contain a routine that performs exponential smoothing, though it can perform other types.

Below, the *okun.gdt* data are used to obtain the exponentially smoothed forecast values of GDP growth. First the data are opened. Then the series to be smoothed is placed in a matrix called `y`. The number of observations is counted and an another matrix called `sm1` is created; it is na $T \times 1$ vector of zeros. We will populate this vector with the smoothed values of y. In line 5 the smoothing parameter is set to 0.38.

There are several ways to populate the first forecast value. A popular way is the take the average of the first $(T+1)/2$ elements of the series. The scalar `stv` is the mean of the first 50 observations. The full sample is then restored.

The loop is quite simple. It loops in increments of 1 from 1 to T. The `--quiet` option is used to suppress screen output. For the first observation, the vector `sm1[1]` receives the initial forecast, `stv`. For all subsequent smoothed values the exponential smoothing is carried out. Once the loop ends the matrix is converted back into a series so that it can be graphed using regular **gretl** functions.

```
1  open "@gretldir\data\poe\okun.gdt"
2  matrix y = { g }
3  scalar T = $nobs
4  matrix sm1 = zeros(T,1)
5  scalar a = .38
6  smpl 1 round((T+1)/2)
7  scalar stv = mean(y)
8  smpl full
9  loop i=1..T --quiet
10     if i = 1
11         matrix sm1[i]=stv
12     else
13         matrix sm1[$i]=a*y[$i]+(1-a)*sm1[i-1]
14     endif
15 endloop
16 series exsm = sm1
17 gnuplot g exsm --time-series
```

The time-series plot of GDP growth and the smoothed series is found in Figure 9.18. Increasing the smoothing parameter to 0.8 reduces the smoothing considerably. The script appears at the end of the chapter, and merely changes the value of `a` in line 5 to 0.8. The figure appears below in the bottom panel of Figure 9.18.

Gretl actually includes a function that can smooth a series in a single line of code. The `movavg` function. To exponentially smooth the series `g`

```
scalar tmid = round(($nobs+1)/2)
scalar a = .38
series exsm = movavg(g, a, tmid)
```

The function takes three argumants. The first is the series for which you want to find the moving average. The second is smoothing parameter, α. The final argument is teh number of initial observations to average to produce y_0. This will duplicate what we did in the script. It is worth mentioning that the movavg function will take a regular moving average if the middle argument is set to a positive integer, with the integer being the number of terms to average.

9.9 Multiplier Analysis

Multiplier analysis refers to the effect, and the timing of the effect, of a change in one variable on the outcome of another variable. The simplest form of multiplier analysis is based on a finite distributed lag model

$$y_t = \alpha + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_q x_{t-q} + e_t \tag{9.19}$$

The estimated coefficients from this model can be used to produce impact, delay and interim multipliers. The **impact multiplier** is the impact of a one unit change in x_t on the mean of y_t. Since x and y are in the same time period the effect is contemporaneous and therefore equal to the initial impact of the change. The ***s*-period delay multiplier** is

$$\frac{\partial E(y_t)}{\partial x_{t-s}} = \beta_s \tag{9.20}$$

is the effect of a change in x s-periods in the past on the average value of the dependent variable in the current period. If x_t is increased by 1 unit and then maintained at its new level in subsequent periods $(t+1), (t+2), \ldots$, then one can compute the **interim multiplier**. An interim multiplier simply adds the immediate effect (impact multiplier), β_0, to subsequent delay multipliers to measure the cumulative effect. So in period $t+1$ the interim effect is $\beta_0 + \beta_1$. In period $t+2$, it will be $\beta_0 + \beta_1 + \beta_2$, and so on. The **total multiplier** is the final effect on y of the sustained increase after q or more periods have elapsed; it is given by $\sum_{s=0}^{q} \beta_s$.

The ARDL model adds lagged values of the dependent variable to the AR model,

$$y_t = \delta + \theta_1 y_{t-1} + \cdots + \theta_p y_{t-p} + \delta_0 x_t + \delta_1 x_{t-1} + \cdots + \delta_q x_{t-q} + v_t \tag{9.21}$$

and this makes the multiplier analysis a little harder. Basically, this needs to be transformed into an infinite distributed lag model using the properties of the lag operator, L. That is, $L^i x_t = x_{t-i}$.

This puts the model into the familiar AR form and the usual definitions of the multipliers can be applied. This is discussed in detail in *POE4* and will not be replicated in any detail here.

For the ARDL(1,1) model used to describe Okun's law we have

$$\Delta u_t = \delta + \theta_1 \Delta u_{t-1} + \delta_0 g_t + \delta_1 g_{t-1} + v_t$$

Written with the lag operator, L

$$(1 - \theta_1 L)\,\Delta u_t = \delta + (\delta_0 + \delta_1 L)\,g_t + v_t$$

$$\Delta u_t = (1 - \theta_1 L)^{-1}\delta + (1 - \theta_1 L)^{-1}(\delta_0 + \delta_1 L)\,g_t + (1 - \theta_1 L)^{-1} v_t$$

$$\begin{aligned}\Delta u_t &= \alpha + \beta_0 g_t + \beta_1 g_{t-1} + \beta_2 g_{t-2} + \beta_3 g_{t-3} + \cdots + e_t \\ &= \alpha + \left(\beta_0 + \beta_1 L + \beta_2 L^2 + \beta_3 L^3 + \cdots\right) g_t + e_t\end{aligned}$$

This is just an infinite distributed lag model. The coefficients for the multipliers involve the β coefficients, which must be solved for in terms of the estimated parameters of the ARDL. The solutions given in *POE4* are

$$\beta_0 = \delta_0 \tag{9.22}$$

$$\beta_1 = \delta_1 + \beta_0\theta_1 \tag{9.23}$$

$$\beta_j = \beta_{j-1}\theta_1 \quad \text{for } j \geq 2 \tag{9.24}$$

The **gretl** code to accomplish this is simple to construct. In terms of the model of Okun's Law,

```
open "@gretldir\data\poe\okun.gdt"
diff u
ols d_u(0 to -1) g(0 to -1) const
scalar b0 = $coeff(g)
scalar b1 = $coeff(d_u_1)*b0+$coeff(g_1)
scalar b2 = b1*$coeff(d_u_1)
scalar b3 = b2*$coeff(d_u_1)
```

This can be automated by using a loop to construct the multipliers. Once this is done, it is simple to graph the result up to an arbitrary number of periods.

The script is:

```
scalar h = 8
matrix mult = zeros(h,2)
loop i=1..h
   mult[i,1] = i-1
   scalar b0 = $coeff(g)
   scalar b1 = $coeff(d_u_1)*b0+$coeff(g_1)
```

```
 7     if i=1
 8         mult[i,2]=b0
 9     elif i=2
10         mult[i,2]=b1
11     else
12         mult[i,2]=mult[i-1,2]*$coeff(d_u_1)
13     endif
14 endloop
15 printf "\nThe impact and delay multipliers are \n %10.5f\n", mult
16 gnuplot 2 1 --matrix=mult --output=display --with-lines --suppress-fitted
```

Although it took a few lines of code, the results (Figure 9.19 below) look great and the code can easily be reused for other models. It assumes that you have already estimated the ARDL(1,1) for Okun data as done in the previous script. The first thing to do is to decide how many multipliers to compute. I chose 8 and initialized a matrix of zeros that is 8×2. We will put lags in the first column and the corresponding multiplier in the second.

The loop begins in line 3 and `i` will start at 1 and end at 8, with increments of 1. The first two multipliers are computed manually. Then a series of if statements follows. Since there are three forms of the multiplier in equations (9.22) – (9.24), there are 3 if statements. When the index is equal 1, the impact multiplier is placed in `m`. When the index is equal to 2, the period one delay is placed in `m`. The last condition fills in `m` for any $i > 2$.

Next, we want to be able to plot the multipliers against lag. This is done from a matrix using the `--matrix` option to **gnuplot**. Also, using the `output=display` option sends the plot to the screen which allows for subsequent editing via **gretl**'s **gnuplot** interface.

The other option is to convert the matrix to data series and use the regular **gretl** GUI to make the plots. This requires opening an empty dataset and setting the observations to equal 8. This is done using the `nulldata` command. The `--preserve` option is required because without it the matrix containing the multipliers would be cleared from memory. This option preserves the contents of all existing matrices and scalars. The lags are read out of the first column and the multipliers from the second.

```
nulldata 8 --preserve
series m = mult[,2]
series lag = mult[,1]
setinfo m -d "Multipliers" -n "Multiplier"
gnuplot m index --with-lines --output=display --suppress-fitted
```

The edited outcome appears in Figure 9.19 below. The figure shows that an increase in GDP growth leads to an initial reduction in the unemployment rate of about 0.18; the effect diminishes over time and lasts about six or seven quarters.

9.10 Appendix

9.10.1 Durbin-Watson Test

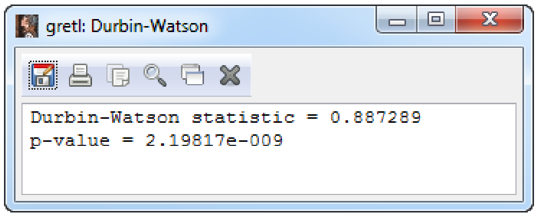
The Durbin-Watson statistic is produced with every time-series regression estimated by least squares. To access the *p*-value associated with the test, which is computed using the Imhoff procedure, use the accessor `$dwpval`. An example based on the Phillips curve is:

```
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
setinfo inf -d "Australian Inflation Rate" -n "Inflation Rate"
setinfo d_u -d "Change in Australian Civilian Unemployment Rate\
    (Seasonally adjusted)" -n "D.Unemployment Rate"
ols inf d_u const
scalar dw_p = $dwpval
print dw
```

The result, including the last line of the regression output that shows the estimated value of ρ and the DW statistic, is:

```
rho                  0.549882   Durbin-Watson        0.887289

                dw_p =  2.1981736e-009
```

The DW statistic is 0.887 and its *p*-value is well below the 5% threshold, indicating significant autocorrelation. The GUI gives a slightly prettier result. It has to be called from the model window as `Tests>Durbin-Watson p-value`.



```
Durbin-Watson statistic = 0.887289
p-value = 2.19817e-009
```

Many interpret a significant *DW* statistic as evidence of general model misspecification.

9.10.2 FGLS and Other Estimators

The feasible GLS estimator of the AR(p) model can be estimated using **gretl** in a number of ways. For first order autocorrelated models the `ar1` command can be used. There are a number of estimators available by option including the Cochrane-Orcutt (iterated), the Prais-Winsten (iterated), and the Hildreth-Lu search procedure. Examples are:

```
list x = d_u const
ar1 inf x                 # Cochrane-Orcutt (default)
ar1 inf x --pwe           # Prais-Winsten
ar1 inf x --hilu --no-corc # Hildreth-Lu
```

The results are collected in a model table below.

AR(1) Errors
Dependent variable: inf

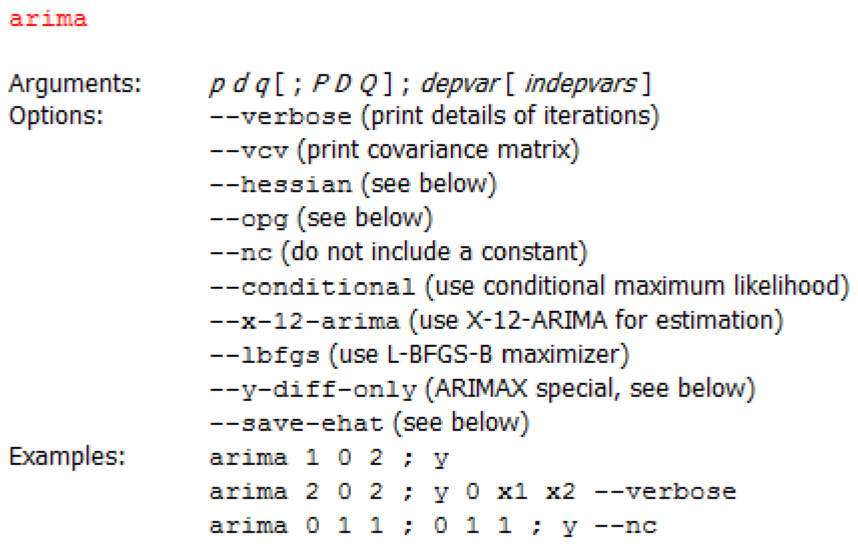
	(CO)	(PW)	(HL)
const	0.7609**	0.7862**	0.7608**
	(0.1238)	(0.1218)	(0.1245)
d_u	−0.6944**	−0.7024**	−0.6953**
	(0.2429)	(0.2430)	(0.2430)
ρ	0.55739	0.55825	.56
n	89	90	89
$\bar{R}^2$	0.3407	0.3418	0.3406

Standard errors in parentheses
* indicates significance at the 10 percent level
** indicates significance at the 5 percent level
CO = Cochrane Orcutt, PW=Prais-Winsten, HL=Hildreth-Lu

You can see that there are minor differences produced by these options. If the `--no-corc` option is not used with `--hilu` then the Hildreth-Lu estimator is modified slightly to perform additional iterations as the end. Notice that the Prais-Winsten is the only procedure to use all 90 observations.

For higher order models there are two commands worth taking note of. The `ar` command estimates a linear regression with arbitrary autocorrelation structure. It uses a generalization of the Cochrane-Orcutt iterative procedure to obtain estimates.

The other estimator is `arima`, the syntax for which appears below:



arima

```
Arguments:     p d q [ ; P D Q ] ; depvar [ indepvars ]
Options:       --verbose (print details of iterations)
               --vcv (print covariance matrix)
               --hessian (see below)
               --opg (see below)
               --nc (do not include a constant)
               --conditional (use conditional maximum likelihood)
               --x-12-arima (use X-12-ARIMA for estimation)
               --lbfgs (use L-BFGS-B maximizer)
               --y-diff-only (ARIMAX special, see below)
               --save-ehat (see below)
Examples:      arima 1 0 2 ; y
               arima 2 0 2 ; y 0 x1 x2 --verbose
               arima 0 1 1 ; 0 1 1 ; y --nc
```

The default estimation method for `arima` in **gretl** is to estimate the parameters of the model using the "native" **gretl** ARMA functionality, with estimation by exact maximum likelihood using the Kalman filter. You can estimate the parameters via conditional maximum likelihood as well.

Estimating the simple AR(1) regression using these estimators is done:

```
ar 1 ; inf x
arima 1 0 0 ; inf x
```

For the `ar` command, list the lag numbers for the desired residuals. In the case of AR(1) this is just 1. This is followed by a semicolon and then the regression to estimate. The `arima` syntax is similar, except you specify `p`, `d`, and `q`, where `p` is the order of the desired autocorrelation, `d` is the number of differences to take of the time-series, and `q` is the order of any moving average terms you might have in the residuals.

The outcome for the simple ARIMA(1,0,0) ia

```
ARMAX, using observations 1987:2-2009:3 (T = 90)
Estimated using Kalman filter (exact ML)
Dependent variable: inf
Standard errors based on Hessian

             coefficient   std. error     z       p-value
  --------------------------------------------------------
  const        0.786212    0.120601      6.519   7.07e-011 ***
  phi_1        0.558827    0.0877359     6.369   1.90e-010 ***
  d_u         -0.702558    0.242234     -2.900   0.0037    ***
```

```
Mean dependent var   0.791111   S.D. dependent var   0.636819
Mean of innovations -0.003996   S.D. of innovations  0.510937
Log-likelihood      -67.45590   Akaike criterion     142.9118
Schwarz criterion    152.9110   Hannan-Quinn         146.9441

                        Real  Imaginary    Modulus  Frequency
  -----------------------------------------------------------
  AR
    Root  1           1.7895     0.0000     1.7895     0.0000
  -----------------------------------------------------------
```

These are very similar to the ones above. The coefficient labeled `phi_1` is the estimate of the autocorrelation parameter. The root of this equation is `1/phi_1`. The roots (or modulus) must be greater than 1 in absolute value in order for the model to be stationary.

9.11 Script

```
open "@gretldir\data\poe\okun.gdt"
set echo off
# change variable attributes
setinfo g -d "percentage change in U.S. Gross Domestic Product, seasonally \
adjusted" -n "Real GDP growth"
setinfo u -d "U.S. Civilian Unemployment Rate  (Seasonally adjusted)" -n \
"Unemployment Rate"

# plot series and save output to files
gnuplot g --with-lines --time-series --output="@workdir\okun_g.plt"
gnuplot u --with-lines --time-series --output="@workdir\okun_u.plt"

# graphing multiple time-series
scatters g u --with-lines

diff u
setinfo d_u -d "Change in U.S. Civilian Unemployment \
Rate  (Seasonally adjusted)" -n \
"D.Unemployment Rate"
scatters g d_u --with-lines --output=display

# distributed lag models
ols d_u const g(0 to -3)
smpl 1986:1 2009:3
ols d_u const g(0 to -2)

gnuplot g g_1

# correlogram and confidence interval
corrgm g 12
```

```
matrix ac = corrgm(g, 12)
matrix lb = ac[,1]-1.96/sqrt($nobs)
matrix ub = ac[,1]+1.96/sqrt($nobs)
matrix all = lb~ac[,1]~ub
colnames(all, "Lower  AC Upper ")
printf "\nAutocorrelations and 95%% confidence intervals\n %9.4f\n", all

# Phillips curve
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
setinfo inf -d "Australian Inflation Rate" -n "Inflation Rate"
setinfo d_u -d "Change in Australian Civilian \
Unemployment Rate  (Seasonally adjusted)" -n \
"D.Unemployment Rate"
scatters inf d_u --with-lines

ols inf const d_u
series ehat = $uhat
gnuplot ehat --time-series
corrgm ehat

# LM tests
ols ehat const d_u ehat(-1)
scalar NR2 = $trsq
pvalue X 1 NR2

ols ehat const d_u ehat(-1 to -4)
scalar NR2 = $trsq
pvalue X 4 NR2

ols inf const d_u
modtest 1 --autocorr
modtest 4 --autocorr --quiet

# HAC standard errors
open "@gretldir\data\poe\phillips_aus.gdt"
set hac_kernel bartlett
set hac_lag nw2
diff u
ols inf const d_u
modeltab add
ols inf const d_u --robust
modeltab add
modeltab show
modeltab free

# nonlinear least squares estimation of regression w/AR(1) errors
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
ols inf const d_u --quiet

```

```
scalar beta1 = $coeff(const)
scalar beta2 = $coeff(d_u)
scalar rho = 0

nls inf = beta1*(1-rho) + rho*inf(-1) + beta2*(d_u-rho*d_u(-1))
    params rho beta1 beta2
end nls
scalar delta = $coeff(beta1)*(1-$coeff(rho))
scalar delta1 = -$coeff(rho)*$coeff(beta2)
printf "\nThe estimated delta is %.3f and the estimated delta1\
is %.3f.\n",delta,delta1
scalar sser=$ess

# estimation of more general model
ols inf const inf(-1) d_u(0 to -1)
scalar sseu=$ess
scalar fstat = (sser-sseu)/(sseu/$df)
pvalue X 1 fstat
pvalue F 1 $df fstat
omit d_u(-1)

ols inf const inf(-1) d_u(0 to -1)
modeltab add
ols inf const inf(-1) d_u(0)
modeltab add
modeltab show
modeltab free

# model selection function
function matrix modelsel (series y, list xvars)
    ols y xvars --quiet
    scalar sse = $ess
    scalar N = $nobs
    scalar K = nelem(xvars)
    scalar aic = ln(sse/N)+2*K/N
    scalar bic = ln(sse/N)+K*ln(N)/N
    matrix A = { K, N, aic, bic}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "K = %d, N = %d, AIC = %.4f SC = %.4f.\n",K,N,aic,bic
    return A
end function

# using the modelsel function
list x = const inf(-1) d_u(0 to -1)
matrix a = modelsel(inf,x)
list x0 = const
matrix b = modelsel(inf,x)
list x = const d_u inf(-1)

# putting the model selection results into a matrix
open "@gretldir\data\poe\phillips_aus.gdt"
```

```
diff u
smpl 1988:3 2009:3
matrix A = {}
scalar q = 0
loop p = 1..6 --quiet
        if p = 1
            list x = const inf(-1) d_u
        else
            list x = const inf(-1 to -$p) d_u
        endif
        matrix a = $p~q~modelsel(inf,x)
        matrix A = A | a
        modelsel(inf,x)
    endloop
scalar q = 1
loop p = 1..6 --quiet
        if p = 1
            list x = const inf(-1) d_u(0 to -1)
        else
            list x = const inf(-1 to -$p) d_u(0 to -1)
        endif
        matrix a = $p~q~modelsel(inf,x)
        matrix A = A | a
    endloop
colnames(A,"p q K N AIC SC ")
print A

smpl full
ols inf const inf(-1 to -4) d_u --robust

# improved modelsel2 function for ARDL
function matrix modelsel2 (list xvars)
    ols xvars --quiet
    scalar sse = $ess
    scalar N = $nobs
    scalar K = nelem(xvars)-1
    scalar aic = ln(sse/N)+2*K/N
    scalar bic = ln(sse/N)+K*ln(N)/N
    matrix A = { K, N, aic, bic}
    #   printf "\nDependent variable and Regressors: %s\n",varname(xvars)
    #   printf "K = %d, N = %d, AIC = %.4f SC = %.4f.\n",K,N,aic,bic
    return A
end function

# using modelsel2
open "@gretldir\data\poe\okun.gdt"
diff u
smpl 1986:1 2009:3
matrix A = {}
loop p = 0..2  --quiet
    loop q = 1..3 --quiet
```

```
        if p=0
            list vars = d_u g(0 to -q) const
        else
            list vars = d_u(0 to -p) g(0 to -q) const
        endif
        matrix a = p~q~modelsel2(vars)
        matrix A = A | a
    endloop
endloop
colnames(A,"p q K N AIC SC ")
print A
function modelsel clear

smpl full
ols d_u(0 to -1) g(0 to -1) const
loop i=1..4
    modtest $i --autocorr --quiet
endloop

open "@gretldir\data\poe\okun.gdt"
smpl 1986:3 2009:3
matrix A = {}
scalar q=0
loop p = 1..5  --quiet
        list vars = g(0 to -p) const
        matrix a = p~q~modelsel2(vars)
        matrix A = A | a
    endloop
colnames(A,"p q K N AIC SC ")
print A
function modelsel clear

# loop to test for autocorrelation in ARDL
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
ols inf(0 to -1) d_u const
loop i=1..5
    modtest $i --autocorr --quiet
endloop

# loop to test for autocorrelation at several lags
open "@gretldir\data\poe\okun.gdt"
ols g(0 to -2) const
series res = $uhat
corrgm res
loop i = 1..4
    modtest $i --autocorr --quiet
endloop

# model selection for Okun data
open "@gretldir\data\poe\okun.gdt"
```

```
smpl 1986:3 2009:3
matrix A = {}
scalar q=0
loop p = 1..5  --quiet
        list vars = g(0 to -p) const
        matrix a = p~q~modelsel2(vars)
        matrix A = A | a
    endloop
colnames(A,"p q K N AIC SC ")
print A

# estimation of preferred model and a forecast
open "@gretldir\data\poe\okun.gdt"
ols g(0 to -2) const
dataset addobs 3
fcast 2009:4 2010:2 --plot="@workdir\ar2plot1.plt"

# multiplier analysis
open "@gretldir\data\poe\okun.gdt"
matrix y = { g }
scalar T = $nobs
matrix sm1 = zeros(T,1)
scalar a = .38
smpl 1 round((T+1)/2)
scalar stv = mean(y)
smpl full
loop i=1..T --quiet
    if i = 1
        matrix sm1[i]=stv
    else
        matrix sm1[$i]=a*y[$i]+(1-a)*sm1[i-1]
    endif
endloop
series exsm = sm1
gnuplot g exsm --time-series

scalar a = .8
loop i=1..T --quiet
    if i = 1
        matrix sm1[i]=stv
    else
        matrix sm1[$i]=a*y[$i]+(1-a)*sm1[i-1]
    endif
endloop
series exsm8 = sm1
gnuplot g exsm8 --time-series

open "@gretldir\data\poe\okun.gdt"
diff u
ols d_u(0 to -1) g(0 to -1) const
scalar b0 = $coeff(g)
scalar b1 =$coeff(d_u_1)*b0+$coeff(g_1)
scalar b2 = b1*$coeff(d_u_1)
scalar b3 = b2*$coeff(d_u_1)

# Matrix & Series Plot
open "@gretldir\data\poe\okun.gdt"
diff u
ols d_u(0 to -1) g(0 to -1) const
scalar h = 8
matrix mult = zeros(h,2)
loop i=1..h
   mult[i,1] = i-1
   scalar b0 = $coeff(g)
   scalar b1 = $coeff(d_u_1)*b0+$coeff(g_1)
   if i=1
       mult[i,2]=b0
   elif i=2
       mult[i,2]=b1
   else
       mult[i,2]=mult[i-1,2]*$coeff(d_u_1)
   endif
endloop

gnuplot 2 1 --matrix=mult --output=display --with-lines --suppress-fitted

printf "\nThe impact and delay multipliers are \n %10.5f\n", mult

nulldata 8 --preserve
series m = mult[,2]
series lag = mult[,1]
setinfo m -d "Multipliers" -n "Multiplier"
gnuplot m index --with-lines --output=display --suppress-fitted

# appendix
open "@gretldir\data\poe\phillips_aus.gdt"
diff u
setinfo inf -d "Australian Inflation Rate" -n "Inflation Rate"
setinfo d_u -d "Change in Australian Civilian \
Unemployment Rate  (Seasonally adjusted)" -n \
"D.Unemployment Rate"

# Durbin-Watson with p-value
list x = d_u const
ols inf x
scalar dw_p = $dwpval
print dw_p

# various ways to estimate AR(1) regression
ar1 inf x
modeltab add
ar1 inf x --pwe
```

```
modeltab add
ar1 inf x --hilu --no-corc
modeltab add
modeltab show
modeltab free

ar 1 ; inf x
arima 1 0 0 ; inf x
```

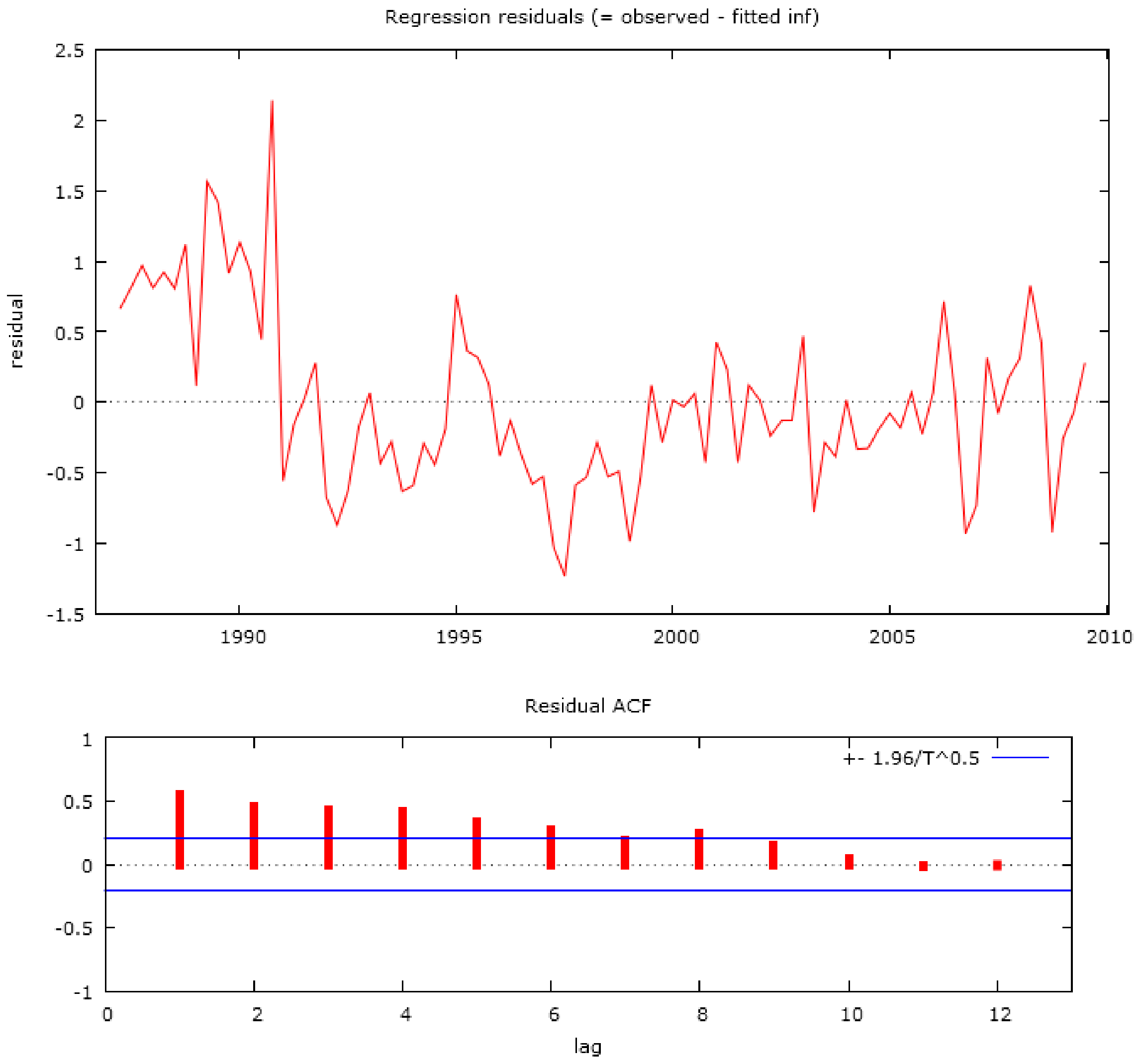
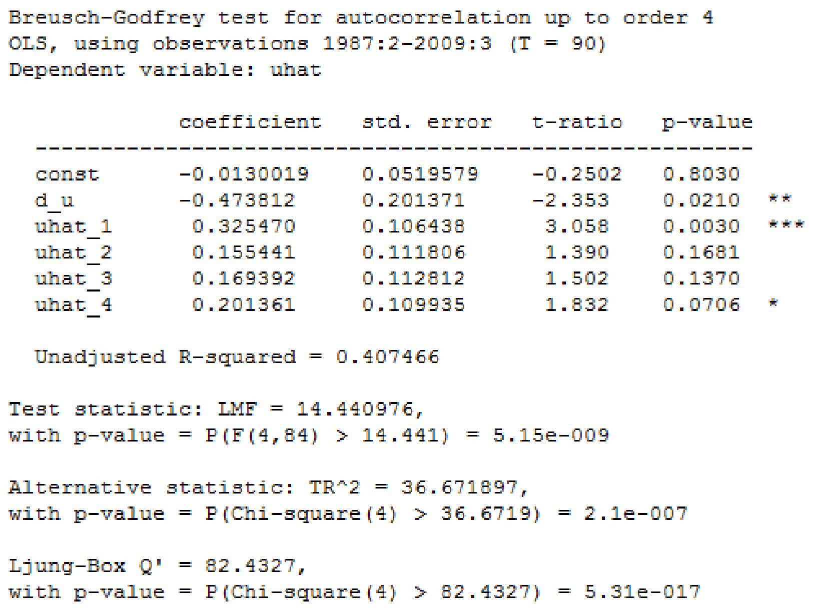
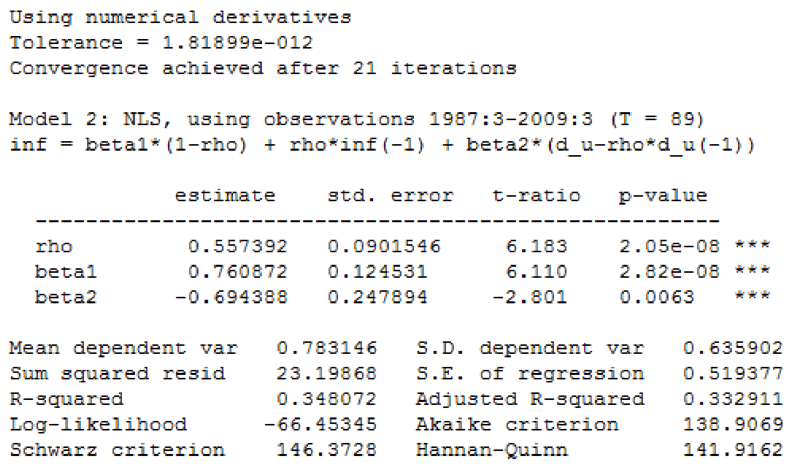


Figure 9.11: This plot shows that the residuals from the simple Phillips curve model are serially correlated. Australia, 1987:1 - 2009:3.



```
Breusch-Godfrey test for autocorrelation up to order 4
OLS, using observations 1987:2-2009:3 (T = 90)
Dependent variable: uhat

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const      -0.0130019    0.0519579    -0.2502   0.8030
  d_u        -0.473812     0.201371     -2.353    0.0210  **
  uhat_1      0.325470     0.106438      3.058    0.0030  ***
  uhat_2      0.155441     0.111806      1.390    0.1681
  uhat_3      0.169392     0.112812      1.502    0.1370
  uhat_4      0.201361     0.109935      1.832    0.0706  *

  Unadjusted R-squared = 0.407466

Test statistic: LMF = 14.440976,
with p-value = P(F(4,84) > 14.441) = 5.15e-009

Alternative statistic: TR^2 = 36.671897,
with p-value = P(Chi-square(4) > 36.6719) = 2.1e-007

Ljung-Box Q' = 82.4327,
with p-value = P(Chi-square(4) > 82.4327) = 5.31e-017
```

Figure 9.12: Using `Test>Autocorrelation` from the model pull-down menu will generate the following output. The alternative hypothesis is AR(4).



```
Using numerical derivatives
Tolerance = 1.81899e-012
Convergence achieved after 21 iterations

Model 2: NLS, using observations 1987:3-2009:3 (T = 89)
inf = beta1*(1-rho) + rho*inf(-1) + beta2*(d_u-rho*d_u(-1))

             estimate    std. error   t-ratio   p-value
  ------------------------------------------------------
  rho         0.557392   0.0901546     6.183    2.05e-08 ***
  beta1       0.760872   0.124531      6.110    2.82e-08 ***
  beta2      -0.694388   0.247894     -2.801    0.0063   ***

Mean dependent var   0.783146   S.D. dependent var   0.635902
Sum squared resid    23.19868   S.E. of regression   0.519377
R-squared            0.348072   Adjusted R-squared   0.332911
Log-likelihood      -66.45345   Akaike criterion     138.9069
Schwarz criterion    146.3728   Hannan-Quinn         141.9162
```

Figure 9.13: Nonlinear least squares results for the AR(1) regression model.

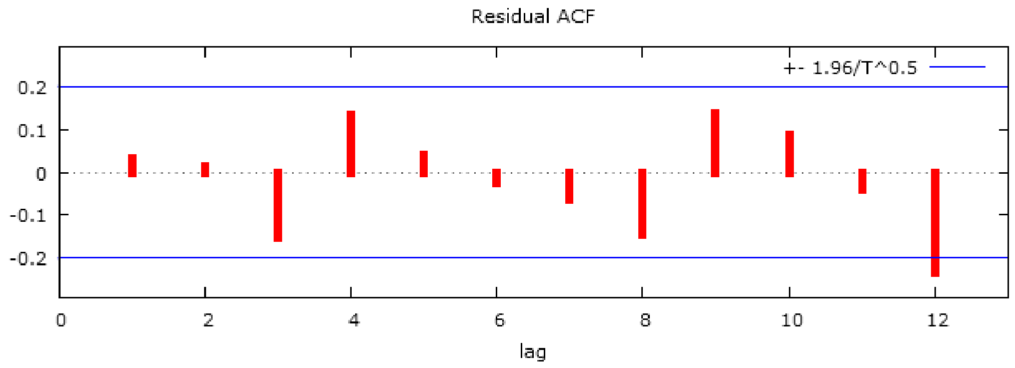
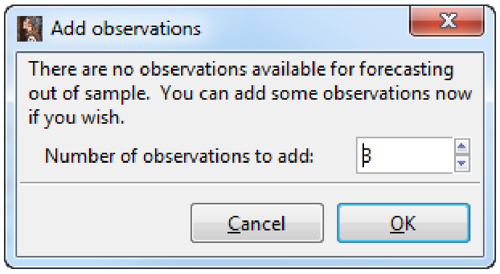


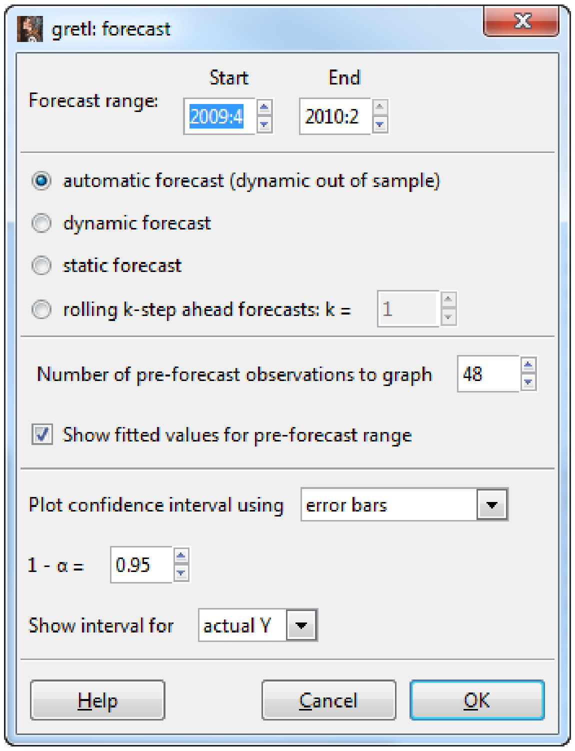
Figure 9.14: Residual correlogram for Okun AR(2)



There are no observations available for forecasting out of sample. You can add some observations now if you wish.

Number of observations to add: 3

Figure 9.15: Using `Data>Add observations` from the main **gretl** pull-down menu will extend the sample period. This is necessary to generate forecasts.


gretl: forecast

Forecast range: Start 2009:4 End 2010:2

[✓ automatic forecast (dynamic out of sample)]

dynamic forecast

static forecast

rolling k-step ahead forecasts: k = 1

Number of pre-forecast observations to graph 48

[✓] Show fitted values for pre-forecast range

Plot confidence interval using error bars

1 - α = 0.95

Show interval for actual Y

Figure 9.16: Forecast dialog box

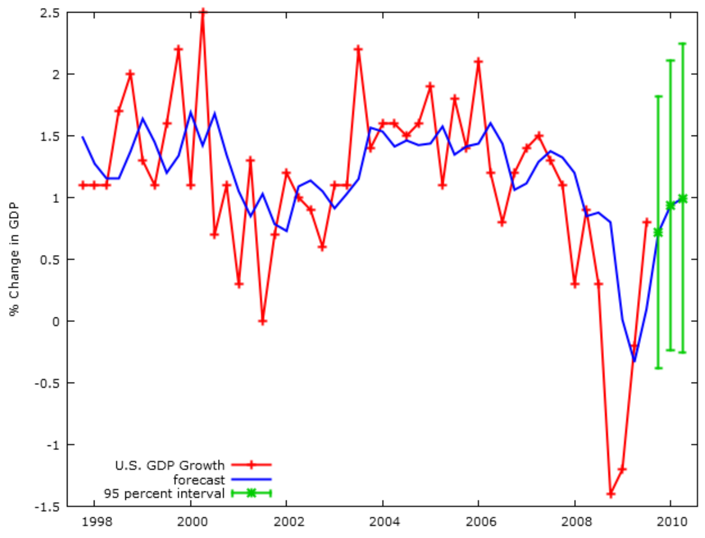


Figure 9.17: **Gretl** calls **gnuplot** to generate a graph of the time-series and the forecast.

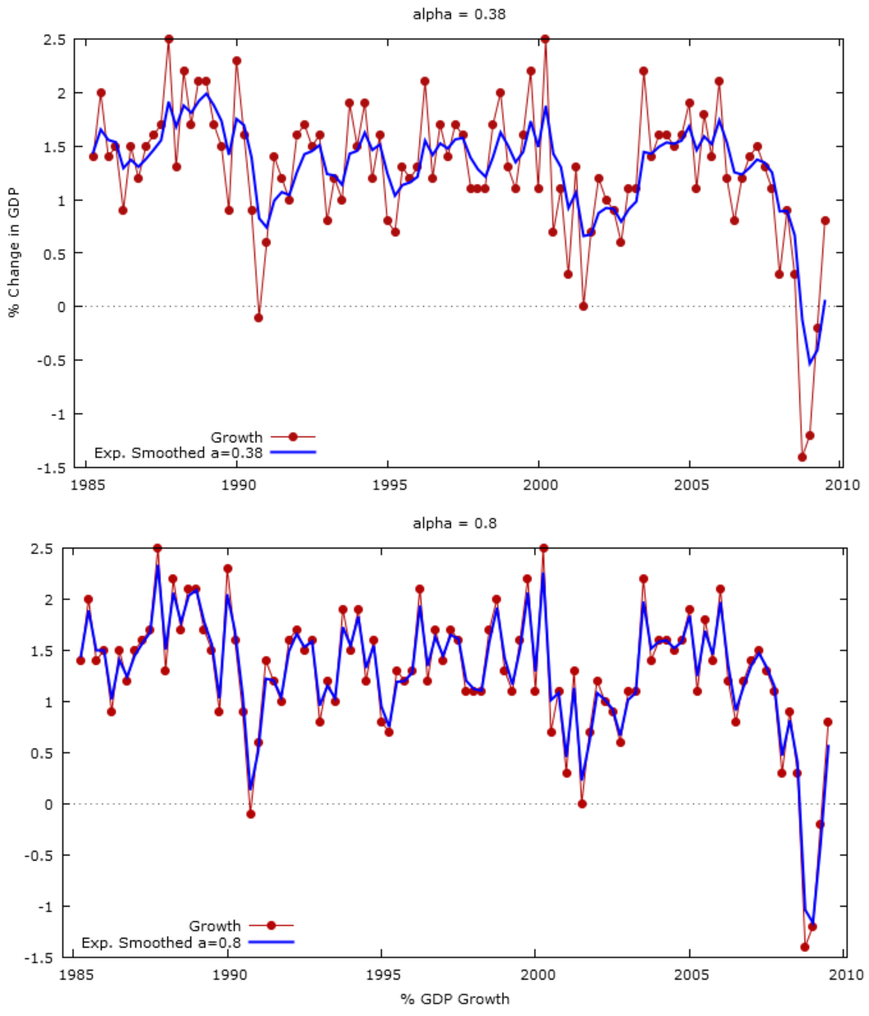


Figure 9.18: GDP growth and exponentially smoothed growth. The smaller the smoothing parameter α, the greater the smoothing.

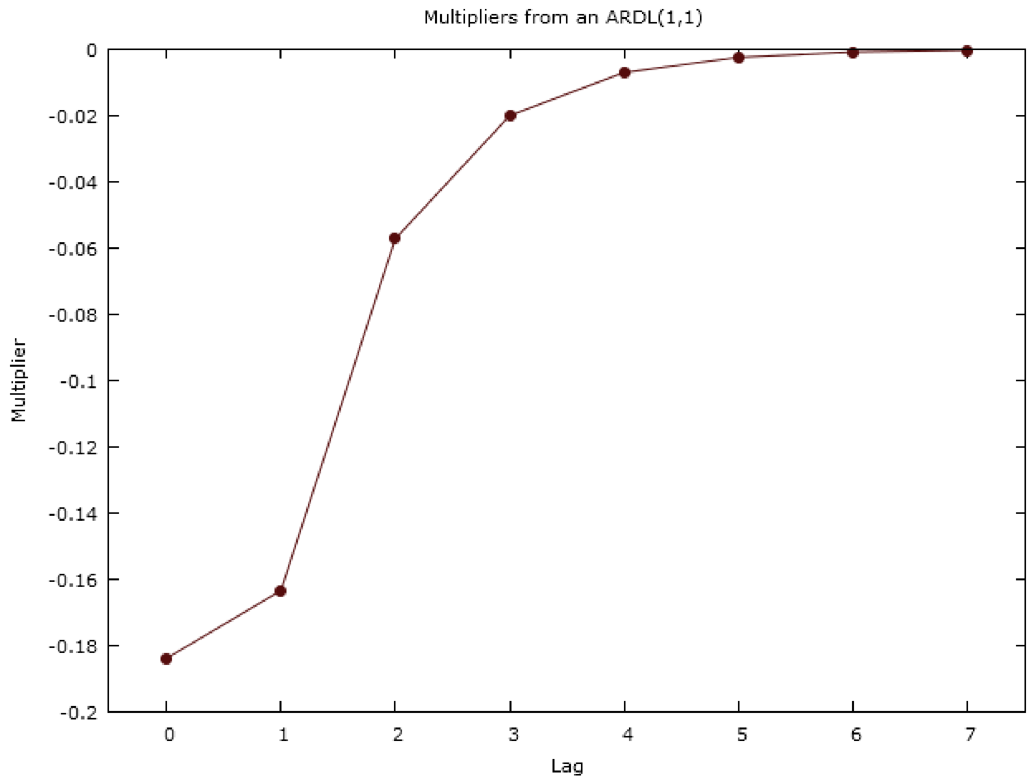


Figure 9.19: Impact and delay multipliers for an ARDL(1,1) of the change in unemployment caused by 1% increase in U.S. GDP growth.

Chapter 10

Random Regressors and Moment Based Estimation

In this chapter you will learn to use instrumental variables to obtain consistent estimates of a model's parameters when its independent variables are correlated with the model's errors.

10.1 Basic Model

Consider the linear regression model

$$y_i = \beta_1 + \beta_2 x_i + e_i \quad i = 1, 2, \ldots, N \tag{10.1}$$

Equation (10.1) suffers from a significant violation of the usual model assumptions when its explanatory variable is contemporaneously correlated with the random error, i.e., $Cov(e_i, x_i) = E(e_i x_i) \neq 0$. When a regressor is correlated with the model's errors, the regressor is often referred to as being **endogenous**.[1] If a model includes an endogenous regressor, least squares is known to be both biased and inconsistent.

An **instrument** is a variable, z, that is correlated with x but not with the error, e. In addition, the instrument does not directly affect y and thus does not belong in the actual model as a separate regressor. It is common to have more than one instrument for x. All that is required is that these instruments, $z_1, z_2, \ldots, z_s$, be correlated with x, but not with e. Consistent estimation of (10.1) is possible if one uses the **instrumental variables** or **two-stage least squares estimator**, rather than the usual OLS estimator.

[1]There is a certain sloppiness associated with the use of endogenous in this way, but it has become standard practice in econometrics.

10.2 IV Estimation

Gretl handles this estimation problem with ease using what is commonly referred to as *two-stage least squares*. In econometrics, the terms two-stage least squares (TSLS) and instrumental variables (IV) estimation are often used interchangeably. The 'two-stage' terminology is a legacy of the time when the easiest way to estimate the model was to actually use two separate least squares regressions. With better software, the computation is done in a single step to ensure the other model statistics are computed correctly. Since the software you use invariably expects you to specify 'instruments,' it is probably better to think about this estimator in those terms from the beginning. Keep in mind though that **gretl** uses the old-style term *two-stage least squares* (`tsls`) even as it asks you to specify instruments in it dialog boxes and scripts.

10.2.1 Least Squares Estimation of a Wage Equation

The example is model of wages estimated using *mroz.gdt* using the 428 women in the sample that are in the labor force. The model is

$$\ln(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 exper^2 + e \tag{10.2}$$

In all likelihood a woman's wages will depend on her ability as well as education and experience. Ability is omitted from the model, which poses no particular problem as long as it is not correlated with either education or experience. The problem in this example, however, is that ability is likely to be correlated with education. The opportunity cost of additional education for those of high ability is low and they tend to get more of it. Hence, there is an **endogeneity** problem in this model. The model is estimated using least squares to produce:

OLS, using observations 1–428
Dependent variable: l_wage

	Coefficient	Std. Error	t-ratio	p-value
const	−0.522041	0.198632	−2.6282	0.0089
educ	0.107490	0.0141465	7.5983	0.0000
exper	0.0415665	0.0131752	3.1549	0.0017
sq_exper	−0.000811193	0.000393242	−2.0628	0.0397

Mean dependent var	1.190173	S.D. dependent var	0.723198
Sum squared resid	188.3051	S.E. of regression	0.666420
R^2	0.156820	Adjusted R^2	0.150854
$F(3, 424)$	26.28615	P-value(F)	1.30e–15
Log-likelihood	−431.5990	Akaike criterion	871.1979
Schwarz criterion	887.4344	Hannan–Quinn	877.6105

The estimated return to another year of schooling is 10.75%. That seems fairly high and if education and the omitted ability are correlated, then it is being estimated inconsistently by least squares.

10.2.2 Two-Stage Least Squares

To perform Two-Stage Least Squares (TSLS) or Instrumental Variables (IV) estimation you need instruments that are correlated with your independent variables, but not correlated with the errors of your model. In the wage model, we will need some variables that are correlated with education, but not with the model's errors. We propose that mother's education (*mothereduc*) is suitable. The mother's education is unlikely to enter the daughter's wage equation directly, but it is reasonable to believe that daughters of more highly educated mothers tend to get more education themselves. These propositions can and will be be tested later. In the meantime, estimating the wage equation using the instrumental variable estimator is carried out in the following example. First, load the *mroz.gdt* data into **gretl**. Then, to open the basic **gretl** dialog box that computes the IV estimator choose `Model>Instrumental Variables>Two-Stage Least Squares` from the pull-down menu as shown below in Figure 10.1. This opens the dialog box shown in Figure 10.2.

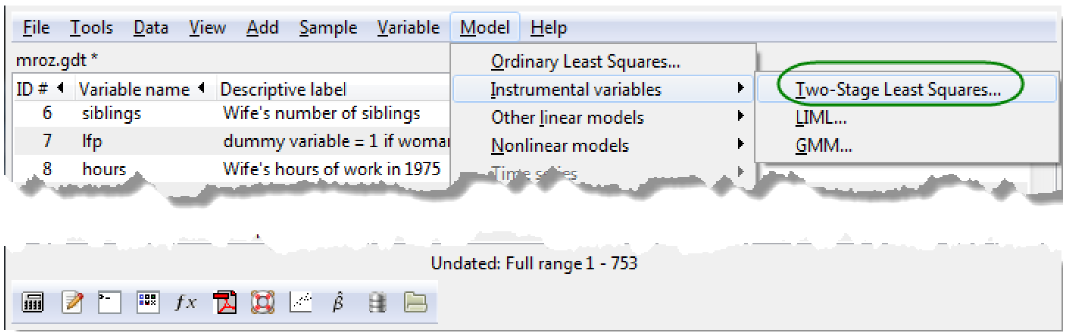

File Tools Data View Add Sample Variable Model Help
mroz.gdt *
ID # Variable name Descriptive label
6 siblings Wife's number of siblings
7 lfp dummy variable = 1 if woma
8 hours Wife's hours of work in 1975
Ordinary Least Squares...
Instrumental variables
Other linear models
Nonlinear models
Time series
Two-Stage Least Squares...
LIML...
GMM...
Undated: Full range 1 - 753

Figure 10.1: Two-stage least squares estimator from the pull-down menus

In this example we choose `l_wage` as the dependent variable, put all of the desired instruments into the *Instruments* box, and put all of the independent variables, including the one(s) measured with error, into the *Independent Variables* box. If some of the right-hand side variables for the model are exogenous, they should be referenced in both lists. That's why the `const`, `exper`, and `sq_exper` variables appear in both places. Press the **OK** button and the results are found in Table 10.1. Notice that **gretl** ignores the sound advice offered by the authors of your textbook and computes an R^2. Keep in mind, though, **gretl** computes this as the squared correlation between observed and fitted values of the dependent variable, and you should resist the temptation to interpret R^2 as the proportion of variation in `l_wage` accounted for by the model.

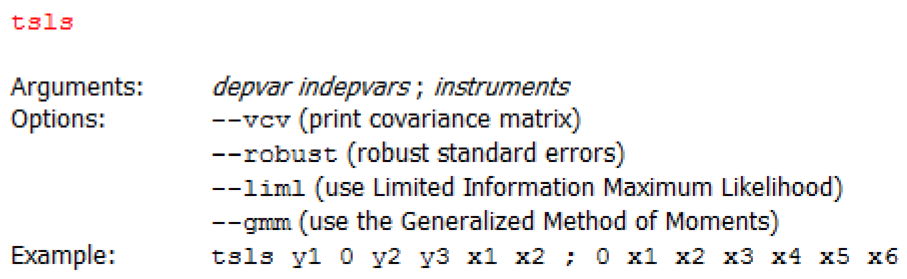
If you prefer to use a script, the syntax is very simple.

TSLS, using observations 1–428
Dependent variable: l_wage
Instrumented: educ
Instruments: const mothereduc exper sq_exper

	Coefficient	Std. Error	z	p-value
const	0.198186	0.472877	0.4191	0.6751
educ	0.0492630	0.0374360	1.3159	0.1882
exper	0.0448558	0.0135768	3.3039	0.0010
sq_exper	−0.000922076	0.000406381	−2.2690	0.0233

Mean dependent var	1.190173	S.D. dependent var	0.723198
Sum squared resid	195.8291	S.E. of regression	0.679604
R^2	0.135417	Adjusted R^2	0.129300
$F(3, 424)$	7.347957	P-value(F)	0.000082
Log-likelihood	−3127.203	Akaike criterion	6262.407
Schwarz criterion	6278.643	Hannan–Quinn	6268.819

Table 10.1: Results from two-stage least squares estimation of the wage equation.


tsls

Arguments:	*depvar indepvars* ; *instruments*
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)
	`--liml` (use Limited Information Maximum Likelihood)
	`--gmm` (use the Generalized Method of Moments)
Example:	`tsls y1 0 y2 y3 x1 x2 ; 0 x1 x2 x3 x4 x5 x6`

The basic syntax is this: `tsls y x ; z`, where `y` is the dependent variable, `x` are the regressors, and `z` the instruments. Thus, the **gretl** command `tsls` calls for the IV estimator to be used and it is followed by the linear model you wish to estimate.

The script for the example above is

```
list x = const educ exper sq_exper
list z = const exper sq_exper mothereduc
tsls l_wage x ; z
```

In the script, the regressors for the wage equation are collected into a list called `x`. The instruments, which should include **all** exogenous variables in the model including the constant, are placed in the list called `z`. Notice that `z` includes **all** of the exogenous variables in `x`. Here the dependent variable, `y`, is replaced with its actual value from the example, (`l_wage`).

It is certainly possible to compute two-stage least squares in two steps, but in practice it is not a good idea to do so. The estimates of the slopes and intercept will be the same as you get using the regular `tsls` IV estimator. The standard errors will not be computed correctly though. To demonstrate, we will do the estimation in two steps and compare results. The **gretl** code to do two step estimation is

```
smpl wage>0 --restrict
ols educ z
series educ_hat = $yhat
```

Notice that the sample had to be restricted to those wages greater than zero using the `--restrict` option. If you fail to do this, the first stage regression will be estimated with all 753 observations instead of the 428 used in `tsls`. TSLS is implicitly limiting the first stage estimation to the non-missing values of `l_wage`. You can see that the coefficient estimates are the same as those in Table 10.1, but the standard errors are not.

10.2.3 Partial Correlations

Valid instruments are supposed to be correlated with the endogenous regressor. However, an important determinant of the statistical properties of the IV estimator is the degree of correlation between the instrument and the endogenous regressor. Furthermore, it is the independent correlation between the instrument and the endogenous regressor that is important. The higher, the better.

One way to get at this in a multiple regression model is to partial out the correlation in variables measured with error that is due to the exogenous regressors. Whatever common variation that remains will measure the independent correlation between the variable measured with error and the instrument. This sounds complicated, but it is not. It is simple to do in **gretl**.

```
ols educ const exper sq_exper
series e1 = $uhat
ols mothereduc const exper sq_exper
series e2 = $uhat
ols e1 e2
corr e1 e2
```

The first statement regresses `const`, `exper`, and `sq_exper` on `educ` and saves the residuals, `e1`. The residuals contain all variation in `educ` not accounted for by the regressors. In effect, the variation in `const`, `exper`, and `sq_exper` has been **partialled out** of the variable measured with error, `educ`. The second regression does the same for the instrument, `mothereduc`. The residuals, `e2`, have the correlation with `const`, `exper`, and `sq_exper` partialled out. Regressing `e2` onto `e1` yields, 0.26769.

This turns out to be exactly the coefficient on `mothereduc` in the first-stage regression. This is no coincidence since regression coefficients are the effect of one variable on another, holding the remaining regressors constant.[2]

```
First Stage Regression:  OLS, using observations 1-428
Dependent variable: educ

               coefficient   std. error   t-ratio    p-value
  -----------------------------------------------------------
  const         9.77510      0.423889     23.06     7.57e-077 ***
  exper         0.0488615    0.0416693     1.173    0.2416
  sq_exper     -0.00128106   0.00124491   -1.029    0.3040
  mothereduc    0.267691     0.0311298     8.599    1.57e-016 ***
```

The correlation between the two sets of residuals yields what is called a partial correlation. This is a correlation where the common effects of `const`, `exper`, and `sq_exper` have been removed. The partial correlation between `e1` and `e2` is 0.3854. Partial correlations play a key role in testing for weak instruments.

10.3 Specification Tests

There are three specification tests you will find useful with instrumental variables estimation. By default, **Gretl** computes each of these whenever you estimate a model using two-stage least squares. Below I'll walk you through doing it manually and we'll compare the manual results to the automatically generated ones.

10.3.1 Hausman Test

The first test is to determine whether the independent variable(s) in your model is (are) in fact uncorrelated with the model's errors. If so, then least squares is more efficient than the IV estimator. If not, least squares is inconsistent and you should use the less efficient, but consistent, instrumental variable estimator. The null and alternative hypotheses are $H_o: \; Cov(x_i, e_i) = 0$ against $H_a: \; Cov(x_i, e_i) \neq 0$. The first step is to use least squares to estimate the first stage of TSLS

$$x_i = \gamma_1 + \theta_1 z_{i1} + \theta_2 z_{i2} + \nu_i \tag{10.3}$$

and to save the residuals, $\hat{\nu}_i$. Then, add the residuals to the original model

$$y_i = \beta_1 + \beta_2 x_i + \delta\hat{\nu}_i + e_i \tag{10.4}$$

Estimate this equation using least squares and use the t-ratio on the coefficient δ to test the hypothesis. If it is significantly different from zero then the regressor, x_i is not exogenous or

[2]This demonstrates an important outcome of the Frisch-Waugh-Lovell Theorem.

predetermined with respect to e_i and you should use the IV estimator (TSLS) to estimate β_1 and β_2. If it is not significant, then use the more efficient estimator, OLS.

The **gretl** script for the Hausman test applied to the wage equation is:

```
open "c:\Program Files\gretl\data\poe\mroz.gdt"
logs wage
list x = const educ exper sq_exper
list z2 = const exper sq_exper mothereduc fathereduc
ols educ z2 --quiet
series ehat2 = $uhat
ols l_wage x ehat2
```

Notice that the equation is **overidentified**. There are two additional instruments, `mothereduc` and `fathereduc`, that are being used for a lone endogenous regressor, `educ`. Overidentification basically means that you have more instruments than necessary to estimate the model. Lines 5 and 6 of the script are used to get the residuals from least squares estimation of the first stage regression, and the last line adds these to the wage model, which is estimated by least squares. The t-ratio on `ehat2` =1.671, which is not significant at the 5% level. We would conclude that the instruments are exogenous.

You may have noticed that whenever you use two-stage least squares in **gretl** that the program automatically produces the test statistic for the Hausman test. There are several different ways of computing this statistic so don't be surprised if it differs from the one you compute manually using the above script.

10.3.2 Testing for Weak Instruments

To test for weak instruments, regress each independent variable suspected of being contemporaneously correlated with the error (x_k) onto all of the instruments (internal and external). Suppose x_K is the endogenous regressor. The first stage regression is:

$$x_K = \gamma_1 + \gamma_2 x_2 + \cdots + \gamma_{K-1} x_{K-1} + \theta_1 z_1 + \cdots + \theta_L z_L + \nu_K \tag{10.5}$$

In this notation, the z_1, ..., z_L are the **external** instruments. The others, x_2, ..., z_{K-1} are exogenous and are used as instruments for themselves (i.e., internal to the model). If the F-statistic associated with the hypothesis that the coefficients on the external instruments, θ_1, ..., θ_L are jointly zero is less than 10, then you conclude that the instruments are weak. If it is greater than 10, you conclude that the instruments are strong enough. The following script uses least squares to perform three such tests. The first regression assumes there is only one instrument, $z1$; the second that the single instrument is $z2$; the third assumes both are instruments.

```
open "@gretldir\data\poe\mroz.gdt"
smpl wage>0 --restrict
```

```
logs wage
square exper
list x = const educ exper sq_exper
list z2 = const exper sq_exper mothereduc fathereduc
ols educ z2
omit mothereduc fathereduc
```

When `omit` follows an OLS regression, **gretl** estimates a restricted model where the variables listed after it are omitted from the model above. It then performs a joint hypothesis test that the coefficients of the omitted variables are zero against the alternative that one or more are not zero. The `--quiet` option reduces the amount of output you have to wade through by suppressing the regressions; only the test results are printed. The output from **gretl** appears in Figure 10.3 below: Since the F value $= 55.4$, which is well beyond 10. We reject the hypothesis that the (external) instruments `mothereduc` and `fathereduc` are weak in favor of the alternative that they are strong.

Gretl proves its worth here. Whenever you estimate a model using two stage least squares, **gretl** will compute the test statistic for the weak instruments test.

10.3.3 Sargan Test

The final test is the **Sargan test** of the overidentifying restrictions implied by an overidentified model. Recall that to be overidentified just means that you have more instruments than you have endogenous regressors. In our example we have a single endogenous regressor (`educ`) and two instruments, (`mothereduc` and `fatehreduc`). The first step is to estimate the model using TSLS using all the instruments. Save the residuals and then regress these on the instruments alone. TR^2 from this regression is approximately χ^2 with the number of surplus instruments as your degrees of freedom. **Gretl** does this easily since it saves TR^2 as a part of the usual regression output, where T is the sample size (which we are calling N in cross-sectional examples). The script for the Sargan test follows:

```
1  open "@gretldir\data\poe\mroz.gdt"
2  smpl wage>0 --restrict
3  logs wage
4  square exper
5  list x = const educ exper sq_exper
6  list z2 = const exper sq_exper mothereduc fathereduc
7  tsls l_wage x; z2
8  series ehat2 = $uhat
9  ols ehat2 z2
10 scalar test = $trsq
11 pvalue X 2 test
```

The first 6 lines open the data, restricts the sample, generates logs and squares, and creates the lists of regressors and instruments. In line 7 the model is estimated using TSLS with the variables in list `x` as regressors and those in `z2` as instruments. In line 8 the residuals are saved as `ehat2`. Then in line 9 a regression is estimated by ordinary least squares using the residuals and instruments as regressors. TR^2 is collected and the p-value computed in the last line.

The result is:

```
Generated scalar test = 0.378071

Chi-square(2): area to the right of 0.378071 = 0.827757
(to the left: 0.172243)
```

The p-value is large and the null hypothesis that the overidentifying restrictions are valid cannot be rejected. The instruments are determined to be ok. Rejection of the null hypothesis can mean that the instruments are either correlated with the errors or that they are omitted variables in the model. In either case, the model as estimated is misspecified.

Finally, **gretl** produces these tests whenever you estimate a model using `tsls`. If the model is exactly identified, then the Sargan test results are omitted. Here is what the output looks like in the wage example:

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: $\chi^2(1) = 2.8256$
  with p-value = 0.0927721

Sargan over-identification test –
  Null hypothesis: all instruments are valid
  Test statistic: LM = 0.378071
  with p-value = $P(\chi^2(1) > 0.378071) = 0.538637$

Weak instrument test –
  First-stage $F(2, 423) = 55.4003$

```
Critical values for desired TSLS maximal size, when running
  tests at a nominal 5% significance level:

     size      10%      15%      20%      25%
    value    19.93    11.59     8.75     7.25

  Maximal size is probably less than 10%
```

You can see that the Hausman test statistic differs from the one we computed manually using the script. However, the p-value associated with this version and ours above are virtually the same. The results from the instrument strength test and from the Sargan test for overdentification are the same. In conclusion, there is no need to compute any of these tests manually, unless you want to.

Finally, you will also see that some additional information is being printed at the bottom of the test for weak instruments. The rule-of-thumb we have suggested is that if the $F > 10$ then instruments are relatively strong. This begs the question, why not use the usual 5% critical value from the F-distribution to conduct the test? The answer is that instrumental variables estimators (though consistent) are biased in small samples. The weaker the instruments, the greater the bias. In fact, the bias is inversely related to the value of the F-statistic. An $F = 10$ is roughly equivalent to $1/F = 10\%$ bias in many cases. The other problem caused by weak instruments is that they affect the asymptotic distribution of the usual t- and F-statistics. This table is generated to give you a more specific idea of what the actual size of the weak instruments test is. For instance, if you are willing to reject weak instruments 10% of the time, then use a critical value of 19.93. The rule-of-thumb value of 10 would lead to actual rejection of weak instruments somewhere between 15% and 20% of the time. Since our $F = 55.4 > 19.93$ we conclude that our test has a size less than 10%. If so, you would expect the resulting TSLS estimator based on these very strong instruments to exhibit relatively small bias.

10.3.4 Multiple Endogenous Regressors and the Cragg-Donald F-test

[3]Cragg and Donald (1993) have proposed a test statistic that can be used to test for weak identification (i.e., weak instruments). In order to compute it manually, you have to obtain a set of canonical correlations. These are not computed in **gretl** so we will use another free software, **R**, to do part of the computations. On the other hand, **gretl** prints the value of the Cragg-Donald statistic by default so you won't have to go to all of this trouble. Still, to illustrate a very powerful feature of **gretl** we will use **R** to compute part of this statistic.

One solution to identifying weak instruments in models with more than one endogenous regressor is based on the use of canonical correlations. Canonical correlations are a generalization of the usual concept of a correlation between two variables and attempt to describe the association between two **sets** of variables.

Let N denote the sample size, B the number of righthand side endogenous variables, G the number of exogenous variables included in the equation (including the intercept), L the number of external instruments–i.e., ones not included in the regression. If we have two variables in the first set of variables and two variables in the second set then there are two canonical correlations, r_1 and r_2.

A test for weak identification–which means that the instruments are correlated with endogenous regressors, but not very highly–is based on the Cragg-Donald F-test statistic

$$\text{Cragg-Donald} - F = [(N - G - B)/L] \times [r_B^2/(1 - r_B^2)] \qquad (10.6)$$

The Cragg-Donald statistic reduces to the usual weak instruments F-test when the number of endogenous variables is $B = 1$. Critical values for this test statistic have been tabulated by Stock and Yogo (2005), so that we can test the null hypothesis that the instruments are weak, against the alternative that they are not, for two particular consequences of weak instruments.

[3]The computations in this section use **R**. You should refer to D for some hints about using **R**.

The problem with weak instruments is summarized by Hill et al. (2011, p. 435):

Relative Bias: In the presence of weak instruments the amount of bias in the IV estimator can become large. Stock and Yogo consider the bias when estimating the coefficients of the endogenous variables. They examine the maximum IV estimator bias relative to the bias of the least squares estimator. Stock and Yogo give the illustration of estimating the return to education. If a researcher believes that the least squares estimator suffers a maximum bias of 10%, and if the relative bias is 0.1, then the maximum bias of the IV estimator is 1%.

Rejection Rate (Test Size): When estimating a model with endogenous regressors, testing hypotheses about the coefficients of the endogenous variables is frequently of interest. If we choose the $\alpha = 0.05$ level of significance we expect that a true null hypothesis is rejected 5% of the time in repeated samples. If instruments are weak, then the actual rejection rate of the null hypothesis, also known as the test size, may be larger. Stock and Yogo's second criterion is the maximum rejection rate of a true null hypothesis if we choose $\alpha = 0.05$. For example, we may be willing to accept a maximum rejection rate of 10% for a test at the 5% level, but we may not be willing to accept a rejection rate of 20% for a 5% level test.

The script to compute the statistic manually is given below:

```
open "@gretldir\data\poe\mroz.gdt"
smpl wage>0 --restrict
logs wage
square exper
series nwifeinc = (faminc-wage*hours)/1000
list x = mtr educ kidsl6 nwifeinc const
list z = kidsl6 nwifeinc mothereduc fathereduc const
tsls hours x ; z
scalar df = $df
```

This first section loads includes much that we've seen before. The data are loaded, the sample restricted to the wage earners, the log of wage is taken, the square is experience is added to the data. Then a new variable is computed to measure family income from all other members of the household. The next part estimates a model of `hours` as a function of `mtr`, `educ`, `kidsl6`, `nwifeinc`, and a constant. Two of the regressors are endogenous: `mtr` and `educ`. The external instruments are `mothereduc` and `fathereduc`; these join the internal ones (`const, kidsl6, nwifeinc`) in the instrument list. The degrees of freedom from this regression is saved to compute $(N - G - B)/L$.

The next set of lines partial's out the influence of the internal instruments on each of the endogenous regressors and on the external instruments.

```
list w = const kidsl6 nwifeinc
ols mtr w --quiet
```

```
12 series e1 = $uhat
13 ols educ w --quiet
14 series e2 = $uhat
15 ols mothereduc w --quiet
16 series e3 = $uhat
17 ols fathereduc w --quiet
18 series e4 = $uhat
```

Now this is where it gets interesting. From here we are going to call a separate piece of software called **R** to do the computation of the canonical correlations. Lines 19-25 hold what **gretl** refers to as a **foreign block**.

```
19 foreign language=R --send-data --quiet
20     set1 <- gretldata[,29:30]
21     set2 <- gretldata[,31:32]
22     cc1 <- cancor(set1,set2)
23     cc <- as.matrix(cc1$cor)
24     gretl.export(cc)
25 end foreign
26 
27 vars = mread("@dotdir/cc.mat")
28 print vars
29 scalar mincc = minc(vars)
30 scalar cd = df*(mincc^2)/(2*(1-mincc^2))
31 printf "\nThe Cragg-Donald Statistic is %10.4f.\n",cd
```

A foreign block takes the form

Foreign Block syntax

```
foreign language=R [--send-data] [--quiet]
... R commands ...
end foreign
```

and achieves the same effect as submitting the enclosed **R** commands via the GUI in the noninteractive mode (see section 30.3 of the **Gretl** Users Guide). In other words, it allows you to use **R** commands from within **gretl** . Of course, you have to have installed **R** separately, but this greatly expands what can be done using **gretl**. The `--send-data` option arranges for auto-loading of the data from the current **gretl** session. The `--quiet` option prevents the output from **R** from being echoed in the **gretl** output. The block is closed with an `end foreign` command.

Inside our foreign block we create two sets of variables. The first set includes the residuals, `e1` and `e2` computed above. There are obtained from a matrix called `gretldata`. This is the name that **gretl** gives to data matrices that are passed into **R**. You have to pull the desired variables out of `gretldata`. In this case I used a rather inartful but effective means of doing so. These two variables are located in the 29th and 30th columns of `gretldata`. These also happen to be their ID numbers in **gretl**. Line 20 puts these two variables into `set1`.

The second set of residuals is put into `set2`. Then, **R**'s `cancor` function is used to find the canonical correlations between the two sets of residuals. The entire set of results is stored in **R** as `cc`. This object contains many results, but we only need the actual canonical correlations between the two sets. The canonical correlations are stored within `cc` as `cor`. They are retrieved as `cc$cor` and put into a matrix with **R**'s as.matrix command. These are exported to **gretl** as `cc.mat`. **R** adds the `.mat` suffix. `cc.mat` is placed in your working directory.

The next step is to read the `cc.mat` into **gretl**. Then in line we take the smallest canonical correlation and use it in line to compute the Cragg-Donald statistic. The result printed to the screen is:

```
? printf "\nThe Cragg-Donald Statistic is %6.4f.\n",cd
The Cragg-Donald Statistic is     0.1006.
```

It matches the automatic one produced by `tsls`, which is shown below, perfectly! It also shows that these instruments are **very** weak.

```
Weak instrument test -
  Cragg-Donald minimum eigenvalue = 0.100568
  Critical values for desired TSLS maximal size, when running
  tests at a nominal 5% significance level:

     size      10%      15%      20%      25%
    value     7.03     4.58     3.95     3.63

  Maximal size may exceed 25%
```

Of course, you can do this exercise without using **R** as well. **Gretl**'s matrix language is very powerful and you can easily get the canonical correlations from two sets of regressors. The following funcrion[4] does just that.

```
function matrix cc(list Y, list X)
   matrix mY = cdemean({Y})
   matrix mX = cdemean({X})

   matrix YX = mY'mX
   matrix XX = mX'mX
   matrix YY = mY'mY

   matrix ret = eigsolve(qform(YX, invpd(XX)), YY)
```

[4]Function supplied by **gretl** guru Riccardo Lucchetti.

```
   return sqrt(ret)
end function
```

The function is called `cc` and takes two arguments, just as the one in **R**. Feed the function two lists, each containing the variable names to be included in each set for which the canonical correlations are needed. Then, the variables in each set are demeaned using the very handy `cdemean` function. This function centers the columns of the matrix argument around the column means. Then the various cross-products are taken (YX, XX, YY) and the eigenvalues for $|Q - \lambda YY| = 0$, where $Q = (YX)(XX)^{-1}(YX)^T$, are returned.

Then, to get the value of the Cragg-Donald F, assemble the two sets of residuals and use the `cc` function to get the canonical correlations.

```
list E1 = e1 e2
list E2 = e3 e4

l = cc(E1, E2)
scalar mincc = minc(l)
scalar cd = df*(mincc^2)/(2*(1-mincc^2))
printf "\nThe Cragg-Donald Statistic is %10.4f.\n",cd
```

10.4 Simulation

In appendix 10F of *POE4*, the authors conduct a Monte Carlo experiment comparing the performance of OLS and TSLS. The basic simulation is based on the model

$$y = x + e \tag{10.7}$$
$$x = \pi z_1 + \pi z_2 + \pi z_3 + v \tag{10.8}$$

The z_i are exogenous instruments that are each N(0,1). The errors, e and v, are

$$\begin{pmatrix} e \\ v \end{pmatrix} \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right] \tag{10.9}$$

The parameter π controls the strength of the instruments and is set to either 0.1 or 0.5. The parameter ρ controls the endogeneity of x. When $\rho = 0$, x is exogenous. When $\rho = 0.8$ it is seriously endogenous. Sample size is set to 100 and 10,000 simulated samples are drawn.

The **gretl** script to perform the simulation appears below:

```
scalar N = 100
nulldata N
```

```
 3 scalar rho = 0.8  # set r = (0.0 or 0.8)
 4 scalar p = 0.5    # set p = (0.1 or 0.5)
 5 matrix S = {1, rho; rho, 1}
 6 matrix C = cholesky(S)
 7
 8 series z1 = normal(N,1)
 9 series z2 = normal(N,1)
10 series z3 = normal(N,1)
11 series xs = p*z1 + p*z2 + p*z3
12 list z = z1 z2 z3
13
14 loop 10000 --progressive --quiet
15     matrix errors = mnormal(N,2)*C'
16     series v = errors[,1]
17     series e = errors[,2]
18     x = xs + v
19     y = x + e
20     ols x const z --quiet
21     scalar f = $Fstat
22     ols y 0 x --quiet
23     scalar b_ols = $coeff(x)
24     tsls y 0 x; 0 z --quiet
25     scalar b_tsls = $coeff(x)
26     store coef.gdt b_ols b_tsls f
27     print b_ols b_tsls f
28 endloop
```

The top part of the script initializes all of the parameters for the simulation. The sample size is set to 100, an empty dataset is created, the values of ρ and π are set, then the covariance matrix is created and the Cholesky decomposition is taken. The Cholesky decomposition is a trick used to create correlation among the residuals. There are more transparent ways to do this (e.g., `e = rho*v + normal(0,1)`), but this is a useful trick to use, especially when you want to correlate more than two series. The systematic part of `x` is created and called `xs` and a list to contain the instruments is created as well.

The `loop` uses the `--progressive` option and is set to do 10,000 iterations. The matrix called errors uses the Cholesky decomposition of the variance covariance to create the correlated errors. The first column we assign to `v` and the second to `e`. The endogenous regressor `x` is created by adding `v` to the systematic portion of the model, and then the dependent variable in the regression is created. The first regression in line 20 is the reduced form. The overall F statistic from this regression can serve as the test for weak instruments since there are no other exogenous variables in the model. The `omit` form of the F-test won't work in a progressive loop so I avoided it here. The slope estimates for least squares and two-stage least squares are collected, stored to *coef.gdt*, and printed.

For this particular parameterization, I obtained the following result:

```
Statistics for 10000 repetitions
       Variable     mean         std. dev.
          b_ols       1.42382      0.0532148
         b_tsls       1.00887       0.106816
              f       30.6130        7.88943
```

With strong instruments, TSLS is basically unbiased. Least squares is seriously biased. Notice that the average value of the weak instrument test is 30.6, indicating the strong instruments. Try changing `p` and `rho` to replicate the findings in Table 10F.1 of *POE4*.

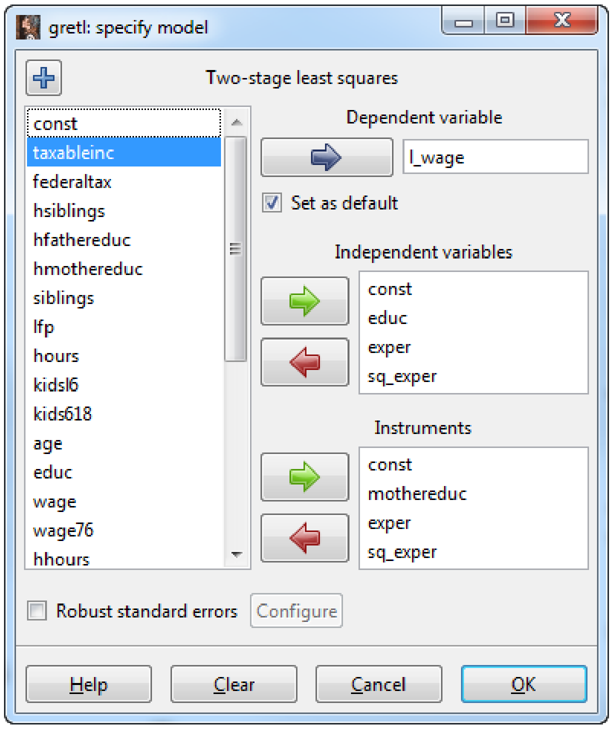
10.5 Script

```
set echo off
open "@gretldir\data\poe\mroz.gdt"
logs wage
square exper

list x = const educ exper sq_exper
list z = const exper sq_exper mothereduc
# least squares and IV estimation of wage eq
ols l_wage x
tsls l_wage x ; z

# tsls--manually
smpl wage>0 --restrict
ols educ z
series educ_hat = $yhat
ols l_wage const educ_hat exper sq_exper

# partial correlations--the FWL result
ols educ const exper sq_exper
series e1 = $uhat
ols mothereduc const exper sq_exper
series e2 = $uhat
ols e1 e2
corr e1 e2

list z = const exper sq_exper mothereduc
list z1 = const exper sq_exper fathereduc
list z2 = const exper sq_exper mothereduc fathereduc

# Hausman test with different sets of instruments
ols educ z --quiet
series ehat = $uhat
ols l_wage x ehat

ols educ z1 --quiet
```

```
series ehat1 = $uhat
ols l_wage x ehat1

ols educ z2 --quiet
series ehat2 = $uhat
ols l_wage x ehat2

# weak instruments
open "@gretldir\data\poe\mroz.gdt"
smpl wage>0 --restrict
logs wage
square exper
list x = const educ exper sq_exper
list z2 = const exper sq_exper mothereduc fathereduc
ols educ z2
omit mothereduc fathereduc

# Sargan test of overidentification
tsls l_wage x; z2
series uhat2 = $uhat
ols uhat2 z2
scalar test = $trsq
pvalue X 2 test

tsls l_wage x ; z2

open "@gretldir\data\poe\mroz.gdt"
smpl wage>0 --restrict
logs wage
square exper
list x = const educ exper sq_exper
list z2 = const exper sq_exper mothereduc fathereduc
tsls l_wage x; z2
series ehat2 = $uhat
ols ehat2 z2
scalar test = $trsq
pvalue X 2 test

# Cragg-Donald F
open "@gretldir\data\poe\mroz.gdt"
smpl wage>0 --restrict
logs wage
square exper
series nwifeinc = (faminc-wage*hours)/1000
list x = mtr educ kidsl6 nwifeinc const
list z = kidsl6 nwifeinc mothereduc fathereduc const
tsls hours x ; z
scalar df = $df
list w = const kidsl6 nwifeinc
ols mtr w --quiet
series e1 = $uhat
```

```
ols educ w --quiet
series e2 = $uhat
ols mothereduc w --quiet
series e3 = $uhat
ols fathereduc w --quiet
series e4 = $uhat

# canonical correlations in R
foreign language=R --send-data --quiet
    set1 <- gretldata[,29:30]
    set2 <- gretldata[,31:32]
    cc1 <- cancor(set1,set2)
    cc <- as.matrix(cc1$cor)
    gretl.export(cc)
end foreign

vars = mread("@dotdir/cc.mat")
print vars
scalar mincc = minc(vars)
scalar cd = df*(mincc^2)/(2*(1-mincc^2))
printf "\nThe Cragg-Donald Statistic is %6.4f.\n",cd

# canonical correlations in gretl
function matrix cc(list Y, list X)
   matrix mY = cdemean({Y})
   matrix mX = cdemean({X})

   matrix YX = mY'mX
   matrix XX = mX'mX
   matrix YY = mY'mY

   matrix ret = eigsolve(qform(YX, invpd(XX)), YY)
   return sqrt(ret)
end function

list E1 = e1 e2
list E2 = e3 e4

l = cc(E1, E2)
scalar mincc = minc(l)
scalar cd = df*(mincc^2)/(2*(1-mincc^2))
printf "\nThe Cragg-Donald Statistic is %10.4f.\n",cd

# simulation for ols and tsls
scalar N = 100
nulldata N
scalar rho = 0.8  # set r = (0.0 or 0.8)
scalar p = 0.5    # set p = (0.1 or 0.5)
matrix S = {1, rho; rho, 1}
matrix C = cholesky(S)

```

```
series z1 = normal(N,1)
series z2 = normal(N,1)
series z3 = normal(N,1)
series xs = p*z1 + p*z2 + p*z3
list z = z1 z2 z3

loop 10000 --progressive --quiet
    matrix errors = mnormal(N,2)*C'
    series v = errors[,1]
    series e = errors[,2]
    x = xs + v
    y = x + e
    ols x const z --quiet
    scalar f = $Fstat
    ols y 0 x --quiet
    scalar b_ols = $coeff(x)
    tsls y 0 x; 0 z --quiet
    scalar b_tsls = $coeff(x)
    store coef.gdt b_ols b_tsls f
    print b_ols b_tsls f
endloop
```



Dependent variable: l_wage

Set as default [✓]

Independent variables: const, educ, exper, sq_exper

Instruments: const, mothereduc, exper, sq_exper

Robust standard errors [ ]

Figure 10.2: Two-stage least squares dialog box

```
Model 2: OLS, using observations 1-428
Dependent variable: educ

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const      12.3694       0.322313     38.38     3.53e-140 ***
  exper       0.0564919    0.0450935     1.253    0.2110
  sq_exper   -0.00190433   0.00134523   -1.416    0.1576

Mean dependent var    12.65888   S.D. dependent var    2.285376
Sum squared resid     2219.216   S.E. of regression    2.285101
R-squared             0.004923   Adjusted R-squared    0.000241
F(2, 425)             1.051372   P-value(F)            0.350365
Log-likelihood       -959.5039   Akaike criterion      1925.008
Schwarz criterion     1937.185   Hannan-Quinn          1929.817

Comparison of Model 1 and Model 2:

  Null hypothesis: the regression parameters are zero for the variables
    mothereduc, fathereduc

  Test statistic: F(2, 423) = 55.4003, with p-value = 4.26891e-022
  Of the 3 model selection statistics, 0 have improved.
```

Figure 10.3: Results from using the `omit` statement after least squares

Chapter 11

Simultaneous Equations Models

In Chapter 11 of *POE4* the authors present a model of supply and demand. The econometric model contains two equations and two dependent variables. The distinguishing factor for models of this type is that the values of two (or more) of the variables are jointly determined. This means that a change in one of the variables causes the other to change and vice versa. The estimation of a simultaneous equations model is demonstrated using the truffle example which is explained below.

11.1 Truffle Example

Consider a supply and demand model for truffles:

$$q_i = \alpha_1 + \alpha_2 p_i + \alpha_3 ps_i + \alpha_4 di_i + e_i^d \tag{11.1}$$

$$q_i = \beta_1 + \beta_2 p_i + \beta_3 pf_i + e_i^s \tag{11.2}$$

The first equation (11.1) is demand and q us the quantity of truffles traded in a particular French market, p is the market price of truffles, ps is the market price of a substitute good, and di is per capita disposable income of the local residents. The supply equation (11.2) contains the variable pf, which is the price of a factor of production. Each observation is indexed by i, $i = 1, 2, \ldots, N$. As explained in the text, prices and quantities in a market are jointly determined; hence, in this econometric model p and q are both endogenous to the system.

11.2 The Reduced Form Equations

The **reduced form equations** express each endogenous variable as a linear function of every exogenous variable in the entire system. So, for our example

$$q_i = \pi_{11} + \pi_{21} ps_i + \pi_{31} di_i + \pi_{41} pf_i + \nu_{i1} \tag{11.3}$$
$$p_i = \pi_{12} + \pi_{22} ps_i + \pi_{32} di_i + \pi_{42} pf_i + \nu_{i2} \tag{11.4}$$

Since each of the independent variables is exogenous with respect to q and p, the reduced form equations (11.3) and (11.4) can be estimated using least squares. In **gretl** the script is

```
open "@gretldir\data\poe\truffles.gdt"
list z = const ps di pf
ols q z
ols p z
```

The **gretl** results appear in Table 11.1 Each of the variables are individually different from zero

$$\widehat{\text{q}} = \underset{(2.434)}{7.89510} + \underset{(4.605)}{0.656402}\,\text{ps} + \underset{(3.094)}{2.16716}\,\text{di} - \underset{(-4.181)}{0.506982}\,\text{pf}$$
$$T = 30 \quad \bar{R}^2 = 0.6625 \quad F(3,26) = 19.973 \quad \hat{\sigma} = 2.6801$$
$$(t\text{-statistics in parentheses})$$

$$\widehat{\text{p}} = \underset{(-4.072)}{-32.5124} + \underset{(4.868)}{1.70815}\,\text{ps} + \underset{(4.409)}{7.60249}\,\text{di} + \underset{(4.536)}{1.35391}\,\text{pf}$$
$$T = 30 \quad \bar{R}^2 = 0.8758 \quad F(3,26) = 69.189 \quad \hat{\sigma} = 6.5975$$
$$(t\text{-statistics in parentheses})$$

Table 11.1: The least squares estimates of the reduced form equations.

at 5%. The overall F-statistics are 19.97 and 69.19, both significant at 5% as well.

11.3 The Structural Equations

The structural equations are estimated using two-stage least squares. The basic **gretl** commands for this estimator are discussed in Chapter 10. The instruments consist of ***all*** exogenous variables, i.e., the same variables you use to estimate the reduced form equations (11.3) and (11.4).

The **gretl** commands to open the truffle data and estimate the structural equations using two-stage least squares are:

```
1 open "@gretldir\data\poe\truffles.gdt"
2 list z = const ps di pf
3 tsls q const p ps di; z
4 tsls q const p pf; z
```

The second line of the script estimates puts all of the exogenous variables into a `list` called `z`. These variables are the ones used to compute the first-stage regression, i.e., the list of instruments. Line 3 estimates the coefficients of the demand equation by TSLS. The **gretl** command `tsls` calls for the two-stage least squares estimator and it is followed by the structural equation you wish to estimate. List the dependent variable (`q`) first, followed by the regressors (`const p ps di`). A semicolon separates the model to be estimated from the list of instruments, now contained in the list, `z`. The fourth line uses the same format to estimate the parameters of the supply equation. Refer to section 10.2, and Figures 10.1 and 10.2 specifically, about using the GUI to estimate the model.

The results from two-stage least squares estimation of the demand equation appear below in Table 11.2 The coefficient on price in the demand equation is -0.374 and it is significantly negative at 5% level. It is good to know that demand curves have a negative slope! The Hausman test for the exogeneity of price is equal to 132 with a near 0 p-value. Price is clearly not exogenous. The test for weak instruments exceeds 10. Additional information from the results yields

```
Critical values for desired TSLS maximal size, when running
  tests at a nominal 5% significance level:

     size      10%      15%      20%      25%
    value    16.38     8.96     6.66     5.53

  Maximal size is probably less than 10%
```

Clearly, the set of instruments is fairly strong. There is no Sargan test because the model is not overidentified. With one endogenous variable there is only 1 external instrument provided by `pf` from the supply equation.

The results for the supply equation are in Table 11.3 In this case, the coefficient on price is positive (as expected). The model is suitably overidentified according to the Sargan test (p-value=0.216 $>$ 0.05), and the instruments are suitably strong (First-stage F-statistic (2, 26) = 41.4873). The outcome of the Hausman test looks suspicious. The statistic is close to zero. A manual check can easily be done using the script:

TSLS of Demand, using observations 1–30
Dependent variable: q
Instrumented: p
Instruments: const ps di pf

	Coefficient	Std. Error	z	p-value
const	−4.27947	5.54388	−0.7719	0.4402
ps	1.29603	0.355193	3.6488	0.0003
di	5.01398	2.28356	2.1957	0.0281
p	−0.374459	0.164752	−2.2729	0.0230

Mean dependent var	18.45833	S.D. dependent var	4.613088
Sum squared resid	631.9171	S.E. of regression	4.929960
R^2	0.226805	Adjusted R^2	0.137590
$F(3, 26)$	5.902645	P-value(F)	0.003266
Log-likelihood	−193.8065	Akaike criterion	395.6130
Schwarz criterion	401.2178	Hannan–Quinn	397.4061

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: $\chi^2(1) = 132.484$
  with p-value = 1.17244e-030

Weak instrument test –
  First-stage $F(1, 26) = 20.5717$

Table 11.2: Two-stage least square estimates of the demand of truffles.

```
ols p x
series v = $uhat
ols q const p pf v
omit v
```

The first step is to regress all instruments on the endogenous regressor, `p`. Get the residuals and add them to the structural equation for supply. Reestimate by least squares and check the t-ratio on the added residual. If it is significant, then `p` is endogenous. In this example, we confirm the **gretl** calculation. This suggests that the supply equation can safely be estimated by least squares. Doing so using:

```
ols q const p pf
```

reveals that the results are almost identical to those from TSLS. This is an implication of having a Hausman statistic that is so small. See the appendix in Chapter 10 of *POE4* for a nice explanation for this.

```
                 TSLS of supply, using observations 1–30
                          Dependent variable: q
                             Instrumented: p
                      Instruments: const ps di pf

                 Coefficient   Std. Error        z        p-value

        const     20.0328      1.22311        16.3785    0.0000
        pf        −1.00091     0.0825279     −12.1281    0.0000
        p          0.337982    0.0249196      13.5629    0.0000

    Mean dependent var   18.45833   S.D. dependent var   4.613088
    Sum squared resid    60.55457   S.E. of regression   1.497585
    R^2                  0.901878   Adjusted R^2         0.894610
    F(2, 27)             95.25929   P-value(F)           5.85e–13

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: χ^2(1) = 2.62751e-007
  with p-value = 0.999591

Sargan over-identification test –
  Null hypothesis: all instruments are valid
  Test statistic: LM = 1.53325
  with p-value = P(χ^2(1) > 1.53325) = 0.215625

Weak instrument test –
  First-stage F(2, 26) = 41.4873
```

Table 11.3: Two-stage least square estimates of the demand of truffles.

11.4 Fulton Fish Example

The following script estimates the reduced form equations using least squares and the demand equation using two-stage least squares for Graddy's Fulton Fish example.

In the example, ln(*quan*) and ln(*price*) are endogenously determined. There are several potential instruments that are available. The variable *stormy* may be useful in identifying the demand equation. In order for the demand equation to be identified, there must be at least one variable available that effectively influences the supply of fish without affecting its demand. Presumably, stormy weather affects the fishermen's catch without affecting people's appetite for fish! Logically, *stormy* may be a good instrument.

The model of demand includes a set of indicator variables for day of the week. Friday is omitted to avoid the dummy variable trap. These day of week variables are not expected to affect supply; fishermen catch the same amount on average on any working day. Day of the week may affect demand though, since people in some cultures buy more fish on some days than others.

The demand equation is:

$$\ln(quan) = \alpha_1 + \alpha_2 \ln(price) + \alpha_3 mon + \alpha_4 tue + \alpha_5 wed + \alpha_6 thu + e_d \tag{11.5}$$

Supply is affected by the weather in the previous three days, which is captured in the indicator variable *stormy*.

$$\ln(quan) = \beta_1 + \beta_2 \ln(price) + \beta_3 stormy + e_s \tag{11.6}$$

In both demand and supply equations, $\ln(price)$ is the right-hand side endogenous variable. Identification of the demand equation requires *stormy* to be significantly correlated with *lprice*. This can be determined by looking at the t-ratio in the *lprice* reduced form equation.

For supply to be identified, at least one of the day of the week dummy variables (*mon tue wed thu*) that are excluded from the supply equation, has to be significantly correlated with *lprice* in the reduced form. If not, the supply equation cannot be estimated; it is not identified.

Proceeding with the analysis, open the data and estimate the reduced form equations for *lquan* and *lprice*. Go ahead and conduct the joint test of the day of the week variables using the `--quiet` option.

```
1 open "@gretldir\data\poe\fultonfish.gdt"
2 #Estimate the reduced form equations
3 list days = mon tue wed thu
4 list z = const stormy days
5 ols lquan z
6 omit days --quiet
7 ols lprice z
8 omit days --quiet
```

Notice how the `list` command is used. A separate list is created to contain the indicator variables. This allows us to add them as a set to the list of instruments in line 4 and to test their joint significance in the reduced form equation for price in lines 6 and 8. The reduced form results for *lquan* appear below:

Model 1: OLS estimates using the 111 observations 1–111
Dependent variable: lquan

Variable	Coefficient	Std. Error	t-statistic	p-value
const	8.810	0.147	59.922	0.000
stormy	−0.388	0.144	−2.698	0.008
mon	0.101	0.207	0.489	0.626
tue	−0.485	0.201	−2.410	0.018
wed	−0.553	0.206	−2.688	0.008
thu	0.054	0.201	0.267	0.790

Standard error of residuals ($\hat{\sigma}$)	0.681790
Unadjusted R^2	0.193372
$F(5, 105)$	5.03429
p-value for $F()$	0.000356107

and the results for *lprice*

Model 2: OLS estimates using the 111 observations 1–111
Dependent variable: lprice

Variable	Coefficient	Std. Error	t-statistic	p-value
const	−0.272	0.076	−3.557	0.001
stormy	0.346	0.075	4.639	0.000
mon	−0.113	0.107	−1.052	0.295
tue	−0.041	0.105	−0.394	0.695
wed	−0.012	0.107	−0.111	0.912
thu	0.050	0.104	0.475	0.636

Unadjusted R^2	0.178889
$F(5, 105)$	4.57511
p-value for $F()$	0.000815589

In the reduced form equation for price, *stormy* is highly significant with a t-ratio of 4.639. This implies that the demand equation is identified and can be estimated with the data. A joint test of the significance of the daily indicator variables reveals that they are not jointly significant; the F-statistic has a p-value of only 0.65. Since the daily indicators are being used as instruments to estimate supply, the supply structural equation is not identified by the data and can't be estimated without better variables.

The two-stage least squares estimates of the demand equation are obtained using:

```
#TSLS estimates of demand
tsls lquan const lprice days ; z
```

to produce the result:

Model 3: TSLS estimates using the 111 observations 1–111
Dependent variable: lquan
Instruments: stormy

Variable	Coefficient	Std. Error	t-statistic	p-value
const	8.506	0.166	51.189	0.000
mon	−0.025	0.215	−0.118	0.906
tue	−0.531	0.208	−2.552	0.011
wed	−0.566	0.213	−2.662	0.008
thu	0.109	0.209	0.523	0.601
lprice	−1.119	0.429	−2.612	0.009

Mean of dependent variable	8.52343
S.D. of dependent variable	0.741672
Sum of squared residuals	52.0903
Standard error of residuals ($\hat{\sigma}$)	0.704342
$F(5, 105)$	5.13561
p-value for $F()$	0.000296831

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: $\chi^2_1 = 2.4261$
  with p-value = 0.119329

First-stage $F(1, 105) = 21.5174$

The coefficient on *lprice* is negative and significant. It also appears that demand is significantly lower on Tuesday and Wednesday compared to Fridays. The Hausman test for the exogeneity of *lprice* is not rejected at 5%. This suggests that least squares might be a suitable means of estimating the parameters in this case. Also, the instruments appear to be sufficiently strong, i.e., the $F = 21.51 > 10$.

11.5 Alternatives to TSLS

There are several alternatives to the standard IV/TSLS estimator. Among them is the limited information maximum likelihood (LIML) estimator, which was first derived by Anderson and Rubin (1949). There is renewed interest in LIML because evidence indicates that it performs better than TSLS when instruments are weak. Several modifications of LIML have been suggested by Fuller (1977) and others. These estimators are unified in a common framework, along with TSLS, using the idea of a k-class of estimators. LIML suffers less from test size aberrations than the TSLS estimator, and the Fuller modification suffers less from bias. Each of these alternatives will be considered below.

In a system of M simultaneous equations let the endogenous variables be $y_1, y_2, \ldots, y_M$. Let there be K exogenous variables $x_1, x_2, \ldots, x_K$. The first structural equation within this system is

$$y_1 = \alpha_2 y_2 + \beta_1 x_1 + \beta_2 x_2 + e_1 \tag{11.7}$$

The endogenous variable y_2 has reduced form $y_2 = \pi_{12}x_1 + \pi_{22}x_2 + \cdots + \pi_{K2}x_K + v_2 = E(y_2) + v_2$, which is consistently estimated by least squares. The predictions from the reduced form are

$$\widehat{E(y_2)} = \hat{\pi}_{12}x_1 + \hat{\pi}_{22}x_2 + \cdots + \hat{\pi}_{K2}x_K \tag{11.8}$$

and the residuals are $\hat{v}_2 = y_2 - \widehat{E(y_2)}$.

The two-stage least squares estimator is an IV estimator using $\widehat{E(y_2)}$ as an instrument. A k-class estimator is an IV estimator using instrumental variable $y_2 - k\hat{v}_2$. The LIML estimator uses $k = \hat{l}$ where $\hat{l}$ is the minimum ratio of the sum of squared residuals from two regressions. The explanation is given on pages 468-469 of *POE4*. A modification suggested by Fuller (1977) that uses the k-class value

$$k = \hat{\ell} - \frac{a}{N-K} \tag{11.9}$$

where K is the total number of instrumental variables (included and excluded exogenous variables) and N is the sample size. The value of a is a constant-usually 1 or 4. When a model is just identified, the LIML and TSLS estimates will be identical. It is only in overidentified models that the two will diverge. There is some evidence that LIML is indeed superior to TSLS when instruments are weak and models substantially overidentified.

With the Mroz data we estimate the *hours* supply equation

$$hours = \beta_1 + \beta_2 mtr + \beta_3 educ + \beta_4 kidsl6 + \beta_5 nwifeinc + e \tag{11.10}$$

A script can be used to estimate the model via LIML. The following one is used to replicate the results in Table 11B.3 of *POE4*.

```
1  open "@gretldir\data\poe\mroz.gdt"
2  square exper
3  series nwifeinc = (faminc-wage*hours)/1000
4  smpl hours>0 --restrict
5  list x = mtr educ kidsl6 nwifeinc const
6  list z1 = educ kidsl6 nwifeinc const exper
7  list z2 = educ kidsl6 nwifeinc const exper sq_exper largecity
8  list z3 = kidsl6 nwifeinc const mothereduc fathereduc
9  list z4 = kidsl6 nwifeinc const mothereduc fathereduc exper
10
11 tsls hours x; z1 --liml
12 tsls hours x; z2 --liml
13 tsls hours x; z3 --liml
14 tsls hours x; z4 --liml
```

LIML estimation uses the `tsls` command with the `--liml` option. The results from LIML estimation of the hours equation, (11.10) the fourth model in line 14, are given below. The variables *mtr* and *educ* are endogenous, and the external instruments are *mothereduc*, *fathereduc*, and *exper*; two endogenous variables with three external instruments suggests that the model is overidentified in this specification.

LIML, using observations 1–428
Dependent variable: hours
Instrumented: mtr educ
Instruments: const nwifeinc mothereduc fathereduc exper kidsl6

	Coefficient	Std. Error	z	p-value
const	18587.9	3683.61	5.0461	0.0000
mtr	−19196.5	4003.68	−4.7947	0.0000
educ	−197.259	64.6212	−3.0525	0.0023
nwifeinc	−104.942	20.6867	−5.0729	0.0000
kidsl6	207.553	163.252	1.2714	0.2036

Mean dependent var	1302.930	S.D. dependent var	776.2744
Sum squared resid	3.11e+08	S.E. of regression	857.3679
Log-likelihood	−5989.014	Akaike criterion	11988.03
Schwarz criterion	12008.32	Hannan–Quinn	11996.04

Smallest eigenvalue = 1.00288
LR over-identification test: $\chi^2(1) = 1.232$ [0.2670]

The LIML results are easy to replicate using matrix commands. Doing so reveals some of **hansl**'s power.

```
matrix y1 = { hours, mtr, educ }
matrix w = { kidsl6, nwifeinc, const, exper, mothereduc, fathereduc}
matrix z = { kidsl6, nwifeinc, const}
matrix Mz = I($nobs)-z*invpd(z'*z)*z'
matrix Mw = I($nobs)-w*invpd(w'*w)*w'
matrix Ez= Mz*y1
matrix W0 = Ez'*Ez
matrix Ew = Mw*y1
matrix W1 = Ew'*Ew
matrix G = inv(W1)*W0
matrix l = eigengen(G, null)
scalar minl = min(l)
printf "\nThe minimum eigenvalue is %.8f \n",minl
matrix X = { mtr, educ, kidsl6, nwifeinc, const }
matrix y = { hours }
matrix kM = (I($nobs)-(minl*Mw))
matrix b =invpd(X'*kM*X)*X'*kM*y
a=rownames(b, " mtr educ kidsl6 nwifeinc const ")
printf "\nThe liml estimates are \n %.6f \n", b
```

The equations that make this magic are found in Davidson and MacKinnon (2004, pp. 537-538).

Hansl's straightforward syntax makes translating the algebra into a computation quite easy. The result from the script is:

```
The liml estimates are
      mtr -19196.516697
     educ -197.259108
   kidsl6 207.553130
 nwifeing -104.941545
    const 18587.905980
```

which matches the ones produced by **gretl**'s `tsls` with `--liml` option.

Fuller's modification relies on a user chosen constant and makes a small change in k of the k-class estimator. In the script that ends the chapter, the value of a is set to 1 and the model is reestimated using Fuller's method. The modification is quite simple to make and the chapter ending script shows the actual details.

11.6 Script

```
set echo off
open "@gretldir\data\poe\truffles.gdt"
# reduce form estimation
list x = const ps di pf
ols q x
ols p x

# demand and supply of truffles
open "@gretldir\data\poe\truffles.gdt"
list x = const ps di pf
tsls q const p ps di; x
tsls q const p pf; x

# Hausman test
ols p x
series v = $uhat
ols q const p pf v
omit v

# supply estimation by OLS
ols q const p pf

# Fulton Fish
open "@gretldir\data\poe\fultonfish.gdt"
#Estimate the reduced form equations
list days = mon tue wed thu
list z = const stormy days
ols lquan z
ols lprice z
omit days --quiet

tsls lquan const lprice days  ; z

# LIML
open "@gretldir\data\poe\mroz.gdt"
square exper
series nwifeinc = (faminc-wage*hours)/1000
smpl hours>0 --restrict
list x = mtr educ kidsl6 nwifeinc const
list z1 = educ kidsl6 nwifeinc const exper
list z2 = educ kidsl6 nwifeinc const exper sq_exper largecity
list z3 = kidsl6 nwifeinc const mothereduc fathereduc
list z4 = kidsl6 nwifeinc const mothereduc fathereduc exper

# LIML using tsls
tsls hours x; z1 --liml
tsls hours x; z2 --liml
tsls hours x; z3 --liml
tsls hours x; z4 --liml
tsls hours x; z4

# LIML using matrices
matrix y1 = { hours, mtr, educ }
matrix w = { kidsl6, nwifeinc, const, exper, mothereduc, fathereduc}
matrix z = { kidsl6, nwifeinc, const}
matrix Mz = I($nobs)-z*invpd(z'*z)*z'
matrix Mw = I($nobs)-w*invpd(w'*w)*w'
matrix Ez= Mz*y1
matrix W0 = Ez'*Ez
matrix Ew = Mw*y1
matrix W1 = Ew'*Ew
matrix G = inv(W1)*W0
matrix l = eigengen(G, null)
scalar minl = min(l)
printf "\nThe minimum eigenvalue is %.8f \n",minl
matrix X = { mtr, educ, kidsl6, nwifeinc, const }
matrix y = { hours }
matrix kM = (I($nobs)-(minl*Mw))
matrix b =invpd(X'*kM*X)*X'*kM*y
a=rownames(b, " mtr educ kidsl6 nwifeinc const ")
printf "\nThe liml estimates are \n %.6f \n", b

# Fuller's Modified LIML a=1
scalar fuller_l=minl-(1/($nobs-cols(w)))
printf "\nThe minimum eigenvalue is %.8f \n",minl
matrix X = { mtr, educ, kidsl6, nwifeinc, const }
matrix y = { hours }
matrix kM = (I($nobs)-(fuller_l*Mw))
matrix b =invpd(X'*kM*X)*X'*kM*y
a=rownames(b, " mtr educ kidsl6 nwifeinc const ")
printf "\nThe liml estimates using Fuller a=1 \n %.6f \n", b
tsls hours mtr educ kidsl6 nwifeinc const ; z4 --liml
```

Chapter 12

Regression with Time-Series Data: Nonstationary Variables

The main purpose this chapter is to use **gretl** to explore the time-series properties of your data. One of the basic points we make in econometrics is that the properties of the estimators and their usefulness for point estimation and hypothesis testing depend on how the data behave. For instance, in a linear regression model where errors are correlated with regressors, least squares won't be consistent and consequently it should not be used for either estimation or subsequent testing.

In time-series regressions the data need to be **stationary**. Basically this requires that the means, variances and covariances of the data series do not depend on the time period in which they are observed. For instance, the mean and variance of the probability distribution that generated GDP in the third quarter of 1973 cannot be different from the one that generated the 4th quarter GDP of 2006. Observations on stationary time-series can be correlated with one another, but the nature of that correlation can't change over time. U.S. GDP is growing over time (not mean stationary) and may have become less volatile (not variance stationary). Changes in information technology and institutions may have shortened the persistence of shocks in the economy (not covariance stationary). Nonstationary time-series have to be used with care in regression analysis. Methods to effectively deal with this problem have provided a rich field of research for econometricians in recent years.

12.1 Series Plots

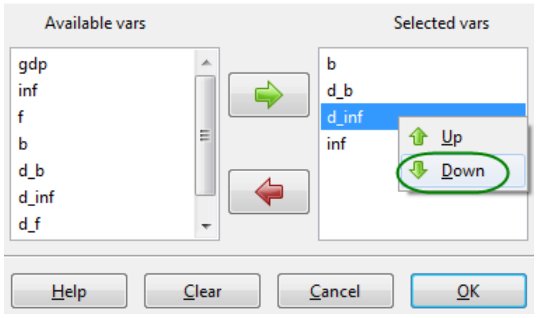
The first thing to do when working with time-series is to take a look at the data graphically. A time-series plot will reveal potential problems with your data and suggest ways to proceed statistically. In **gretl** time-series plots are simple to generate since there is a built-in function that performs this task. Open the data file *usa.gdt* and create the first differences using the `diff` command. The first differences of your time-series are added to the data set and each of the differenced series is prefixed with '`d_`', e.g., $\Delta gdp_t = gdp_t - gdp_{t-1} \Rightarrow$ `d_gdp`.

```
open "@gretldir\data\poe\usa.gdt"
diff b inf f gdp
setinfo b -d "3-year Bond rate" -n "3-year Bond rate"
setinfo d_b -d "Change in the 3-year Bond rate" -n "D.BOND"
setinfo inf -d "annual inflation rate" -n "inflation rate"
setinfo d_inf -d "Change in the annual inflation rate" -n "D.INFLATION"
setinfo gdp -d "real US gross domestic product" -n "Real GDP"
setinfo d_gdp -d "= first difference of gdp" -n "D.GDP"
setinfo f -d "federal funds rate" -n "Fed Funds Rate"
setinfo d_f -d "= first difference of f" -n "D.FED_FUNDS"
```

Next, I want to add descriptions and labels for graphing. This is done using the `setinfo` command. Recall, the `-d` switch changes the description and `-n` assigns a label to be used in graphs. Text needs to be enclosed in double quotes.

Plotting the series can be done in any number of ways. The easiest is to use `view>multiple graphs>Time series` from the pull-down menu. This will allow you to graph the eight series in two batches. Two batches are required since the maximum number of series that can be graphed simultaneously is currently limited to six.

Use your mouse to select four of the series. I chose `inf`, `d_inf`, `f`, `d_f`. Once these are highlighted choose `View>Multiple graphs>Time-series` from the pull-down menu. These variables should appear in the 'Selected vars' box. You can change the ordering of the variables by highlighting a variable and a right mouse click. The Up/Down box opens and clicking **Down** will place `d_inf` below `inf` in the list.



Then, select `Add>First differences of selected variables` from the pull-down menu as shown in Figure 12.2. You can gain more control over how the graphs look by plotting the series individually and then editing the graphs to taste. For instance, here is the plot of the change in the bond rate, with recessionary periods highlighted (Figure 12.3).

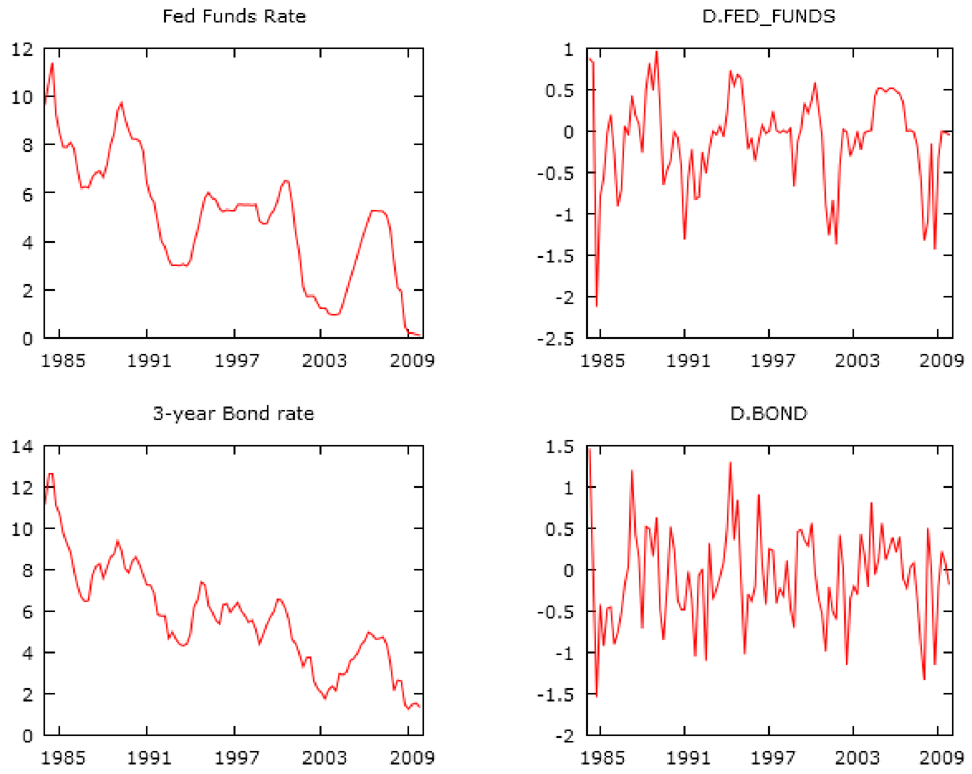


Figure 12.1: Highlight `inf`, `d_inf`, `f`, and `d_f` using the mouse. Then choose `View>Multiple graphs>Time-series` from the pull-down menu to open the dialog box. Click **OK** reveals this graph.

The next thing to do is to create a set of summary statistics. In this case, the textbook has you produce summary statistics for subsamples of the data. The first subsample consists of the 52 observations from 1984:2 to 1996:4. The second also contains 52 observations and continues from 1997:1 to 2009:4. The `summary` command is used to obtain the summary statistics on the desired subsample. In the script, open the data file *usa.gdt* and change the sample to 1984:2-1996:4 using the command `smpl 1984:2 1996:4`. Issue the `summary --simple` command to print the condensed set of summary statistics of all variables in memory to the screen. Finally, restore the sample to the full range using `smpl full`.

Ordinarily, **gretl**'s `smpl` functions are cumulative. This means that whatever modifications you make to the sample are made based on the sample that is already in memory. In this example though, we were able to load the second subperiod without having to first restore the full sample. This is undocumented so it may stop working at some point. If so, just issue a `smpl full` command after getting summary statistics for the first subset.

The script is

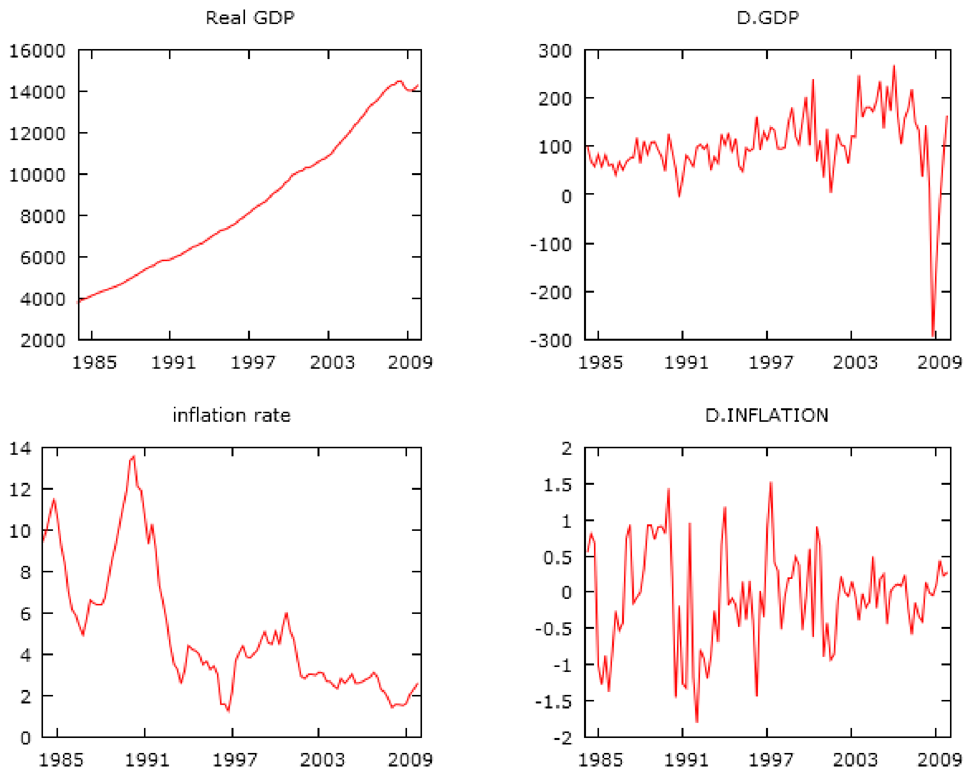


Figure 12.2: Add the first differences of the selected series from the pull-down menu.

```
smpl 1984:2 1996:4
summary --simple
smpl 1997:1 2009:4
summary --simple
smpl full
```

This produces

```
Summary statistics, using the observations 1984:2 - 1996:4

                     Mean         Minimum        Maximum       Std. Dev.
gdp                5813.0          3906.3         8023.0          1204.6
inf                6.9037          1.2800         13.550          3.3378
f                  6.4173          2.9900         11.390          2.1305
b                  7.3431          4.3200         12.640          1.9398
d_b              -0.10294         -1.5400         1.4500         0.63128
d_inf            -0.16059         -1.8000         1.4300         0.83201
d_f             -0.086471         -2.1200        0.97000         0.58607
d_gdp              82.659         -4.6000         161.80          29.333

Full data range: 1984:1 - 2009:4 (n = 104)
```

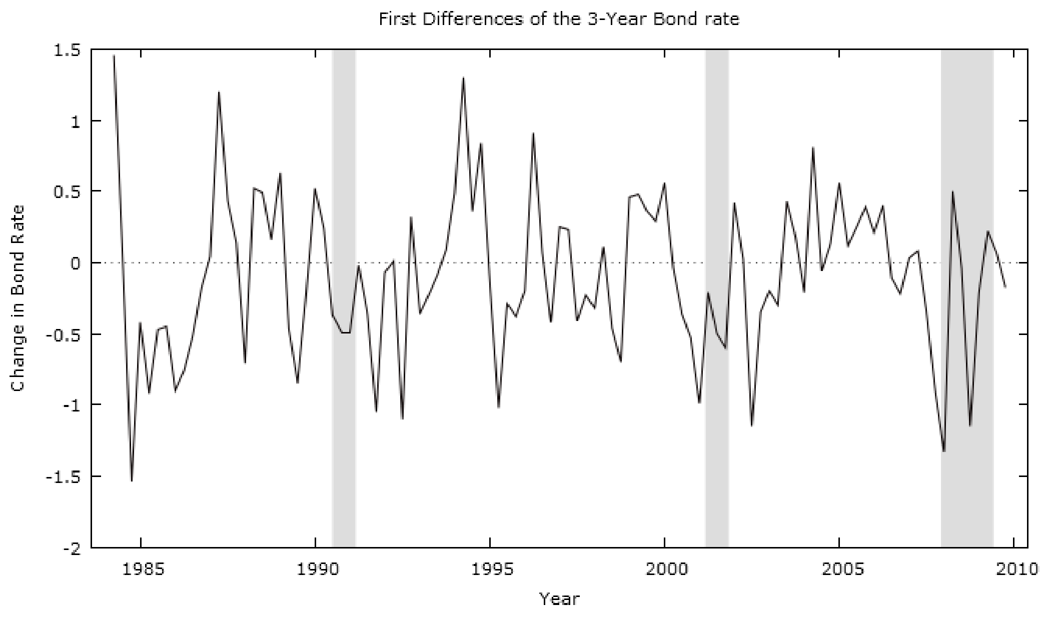


Figure 12.3: Individual plots can be edited using the **edit controls**. This one shows the first differences of the 3 year bond rate. Recessions are shaded grey.

```
Current sample: 1997:1 - 2009:4 (n = 52)

Summary statistics, using the observations 1997:1 - 2009:4

                       Mean          Minimum          Maximum       Std. Dev.
gdp                  11458.           8137.0           14485.          2052.1
inf                  3.2194           1.4500           6.0400          1.1166
f                    3.4875          0.12000           6.5200          2.0253
b                    3.9771           1.2700           6.5600          1.5643
d_b               -0.087500          -1.3300          0.81000         0.47885
d_inf              0.025192         -0.93000           1.5200         0.46174
d_f               -0.099231          -1.4300          0.59000         0.51429
d_gdp                120.27          -293.70           267.90          92.920
```

Notice that the `--simple` option is used to suppress other summary statistics like the median, skewness and kurtosis. If these statistics interest you, feel free to remove the option.

One can limit the summary statistics to certain variables by creating a `list` that follows summary. For instance, to limit the summary statistics to the variables in levels you could use:

```
list levels = gdp inf f b
summary levels --simple
```

The levels of each time series are put into a list called `levels`. The summary statistics of all the contents can then be obtained using `summary levels`.

12.2 Spurious Regressions

It is possible to estimate a regression and find a statistically significant relationship even if none exists. In time-series analysis this is actually a common occurrence when data are not stationary. This example uses two data series, *rw1* and *rw2*, that were generated as independent random walks.

$$\begin{aligned} rw_1 &: y_t = y_{t-1} + v_{1t} \\ rw_2 &: x_t = x_{t-1} + v_{2t} \end{aligned} \tag{12.1}$$

The errors are independent standard normal random deviates generated using a pseudo-random number generator. As you can see, x_t and y_t are not related in any way. To explore the empirical relationship between these unrelated series, load the *spurious.gdt* data and declare the data to be time-series.

```
open "@gretldir\data\poe\spurious.gdt"
setobs 1 1 --special-time-series
```

The sample information at the bottom of the main **gretl** window indicates that the data have already been declared as time-series and that the full range (1-700) is in memory. The first thing to do is to plot the data using a time-series plot. To place both series in the same time-series graph, select `View>Graph specified vars>Time-series plots` from the pull-down menu. This will reveal the 'define graph' dialog box. Place both series into the 'Selected vars' box and click **OK**. The result appears in top part of Figure 12.4 (after editing) below. The XY scatter plot is obtained similarly, except use `View>Graph specified vars>X-Y scatter` from the pull-down menu. Put *rw1* on the y axis and *rw2* on the x axis.

The linear regression confirms this. Left click on the graph to reveal a pop-up menu, from which you choose **Edit**. This brings up the plot controls shown in Figure 4.16. Select the **linear fit option** to reveal the regression results in Table 12.1.

The coefficient on *rw2* is positive (0.842) and significant ($t = 40.84 > 1.96$). However, these variables are not related to one another! The observed relationship is purely **spurious**. The cause of the spurious result is the nonstationarity of the two series. This is why you must check your data for stationarity whenever you use time-series in a regression.

OLS, using observations 1–700
Dependent variable: rw1

	Coefficient	Std. Error	t-ratio	p-value
const	17.8180	0.620478	28.7167	0.0000
rw2	0.842041	0.0206196	40.8368	0.0000

Sum squared resid	51112.33	S.E. of regression	8.557268
R^2	0.704943	Adjusted R^2	0.704521

Table 12.1: Least squares estimation of a spurious relationship.

12.3 Tests for Stationarity

The (augmented) Dickey-Fuller test can be used to test for the stationarity of your data. To perform this test, a few decisions have to be made regarding the time-series. The decisions are usually made based on visual inspection of the time-series plots. By looking at the plots you can determine whether the time-series have a linear or quadratic trend. If the trend in the series is quadratic then the differenced version of the series will have a linear trend in them. In Figure 12.1 you can see that the Fed Funds rate appears to be trending downward and its difference appears to wander around some constant amount. Ditto for bonds. This suggests that the Augmented Dickey Fuller test regressions for each of the series should contain a constant, but not a time trend.

The GDP series in the upper left side of Figure 12.2 appears to be slightly quadratic in time. The differenced version of the series that appears below it has a slight upward drift and hence I would choose an augmented Dickey-Fuller (ADF) test that included a constant and a time trend. As you may have guessed, analyzing time-series in this way is a bit like reading entrails and there is something of an art to it. Our goal is to reduce some of the uncertainty using formal tests whenever we can, but realize that choosing the appropriate test specification requires some judgement by the econometrician.

The next decision is to pick the number of lagged terms to include in the ADF regressions. Again, this is a judgement call, but the residuals from the ADF regression should be void of any autocorrelation. **Gretl** is helpful in this respect since it reports the outcome of an autocorrelation test whenever the built-in ADF routines are used. Below is the example from your text, where the stationarity of the Fed Funds rate and the three year bond series are explored.

To perform the ADF test on the Fed Funds rate, use the cursor to highlight the series and click `Variable>Unit root tests>Augmented Dickey-Fuller test` from the pull-down menu as shown in Figure 12.5 below. This brings up the dialog box shown in the next Figure, 12.6. Notice that here is where you inform **gretl** whether you want to include a constant, trend, trend squared, seasonal indicators, etc. We have chosen to start with a maximum lag of 4 and to allow **gretl** to test-down to the number of lags required. We have also chosen to perform the test two ways, one including a constant in the ADF regression and the other including a constant and trend. Also, we have not checked the box to have **gretl** report the results from the regression. To make the results a bit more transparent it is often a good idea to see the regression results that generate the test statistics, and we invite you to try it for yourself.

At the bottom of the dialog you must choose whether you want to use the level or the difference of the variable. Choosing the **level**, as shown in the box, puts the **difference** on the left-hand side of the regression. This can be a bit confusing, but in reality it should not be. Remember, you are testing to see whether the levels values of the series are stationary. Choosing this box is telling **gretl** that you want to test the nonstationarity of the series' levels.

If you want to check to see whether the differences are nonstationary, then click the radio button below the one indicated. Click **OK** and the results appear as in Figure 12.7.

The test results are quite informative. First it tells you that you one lagged value was selected (from a maximum of 4) to include in the model. It reveals that the first set of statistics is for a test based on a regression with a constant. It provides you with an estimate of γ, which it refers to as `a-1`, the t-ratio for γ, and the correct p-value for the statistic as computed by Davidson and MacKinnon. It also reports an estimated autocorrelation coefficient for the errors (-0.051) which should be small if you have chosen the correct number of lags in the ADF regression.

The null hypothesis of the ADF test is that the time-series has a unit root and is not stationary. *If you reject this hypothesis then you conclude that the series is stationary.* To not reject the null means that the level is *not* stationary. Here, the test statistic for the stationarity of the Fed Funds rate is -2.50482 which has a p-value of 0.1143. Nonstationarity of the Fed Funds rate can not be rejected in this case at the usual 5 or 10% levels of significance.

One more thing should be said about the ADF test results. **Gretl** expresses the model in a slightly different way than your textbook. The model is

$$(1-L)y_t = \beta_0 + (\alpha - 1)y_{t-1} + \alpha_1 \Delta y_{t-1} + e_t \tag{12.2}$$

The coefficient β_0 is included because you believe the series has a trend, $(\alpha-1) = \gamma$ is the coefficient of interest in the Dickey-Fuller regression, and α_1 is the parameter for the term that 'augments' the Dickey-Fuller regression. It is included to eliminate autocorrelation in the model's errors, e_t, and more lags can be included if needed to accomplish this. The notation on the left side of the equation $(1-L)y_t$ makes use of the lag operator, L. The lag operator performs the magic $Ly_t = y_{t-1}$. Thus, $(1-L)y_t = y_t - Ly_t = y_t - y_{t-1} = \Delta y_t$.

The script to perform the ADF test is:

```
open "@gretldir\data\poe\usa.gdt"
adf 4 f --c --ct --test-down
```

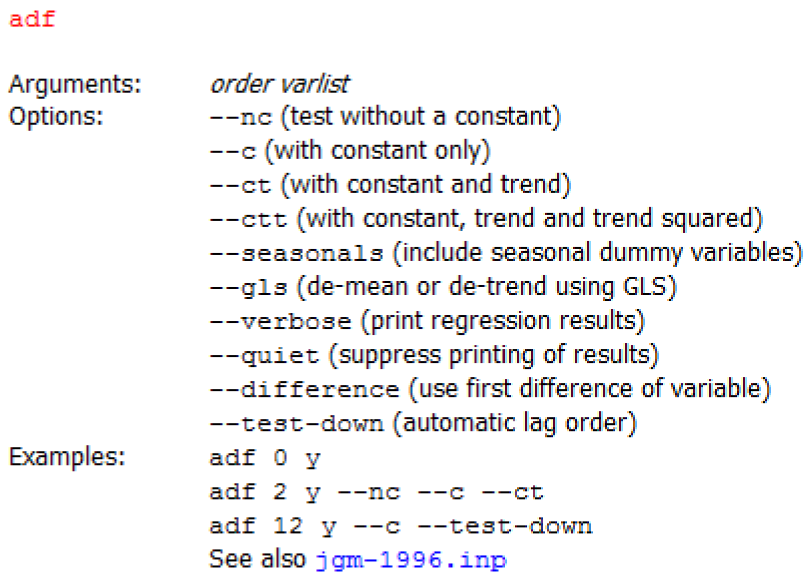
The syntax is fairly straightforward. The first number after `adf` gives **gretl** the lag number for the ADF test. Next is the series name to test. There are three options: 1) `--c` tells **gretl** to run the ADF regressions with a constant (when the series has a trend) 2) `--ct` tells **gretl** to run the regression with a constant and trend (the series and its difference have trends) and 3) `--test-down` informs **gretl** that you want to use the lag number 4 as a starting point and to reduce the number of lags to find the optimal model.

When testing down, **gretl** follows the algorithm.

1. Estimate the ADF regression with the given maximum lags, k_m, of the dependent variable included as regressors.
2. **Gretl** checks to see if the last lag significantly different from zero at the 10% level. If it is, perform the ADF test with lag order k_m. If the coefficient on the last lag is not significant, reduce the lag number by one, $k_{m-1} = k_m - 1$ and repeat.
3. if k_1 is insignificant, execute the test with lag order 0.

You could also use model selection rules via a user written function to eliminate lags from the ADF regressions.

The `adf` syntax from the **gretl** command help is summarized:



```
adf

Arguments:    order varlist
Options:      --nc (test without a constant)
              --c (with constant only)
              --ct (with constant and trend)
              --ctt (with constant, trend and trend squared)
              --seasonals (include seasonal dummy variables)
              --gls (de-mean or de-trend using GLS)
              --verbose (print regression results)
              --quiet (suppress printing of results)
              --difference (use first difference of variable)
              --test-down (automatic lag order)
Examples:     adf 0 y
              adf 2 y --nc --c --ct
              adf 12 y --c --test-down
              See also jgm-1996.inp
```

The augmented version of the Dickey-Fuller test adds lagged differences to the model. For the model with a constant and no trend this would be:

$$\Delta y_t = \alpha + \gamma y_{t-1} + \sum_{s=1}^{m} a_s \Delta y_{t-s} + v_t \qquad (12.3)$$

You have to pick the number of lags to include. Essentially, one should include just enough lags of Δy_{t-s} to ensure that the residuals are uncorrelated. The number of lagged terms can also be determined by examining the autocorrelation function (ACF) of the residuals, or the significance of the estimated lag coefficients. The latter is what **gretl** does when you use the `--test-down` option. Notice that **gretl** also includes the autocorrelation coefficient in the output. Thus, it serves as a final check of the adequacy of your ADF regression.

In the example from *POE4*, the Federal Funds rate (`f`) and the 3-year Bond rate (`b`) are considered. The series plots show that the data wander about, indicating that they may be nonstationary. To perform the Dickey-Fuller tests, first decide whether to use a constant and/or a trend. Since the levels of the series fluctuate around a nonzero mean and the differences around zero, we include a constant. Then decide on how many lagged difference terms to include on the right-hand side of the equation. You can also use the model selection rules described in chapter 9, to choose lags for the ADF.

To use this method as a means of model selection, we have to return to our `modelsel2` function and estimate the ADF regressions manually. The model selection function is shown below:

Model Selection function

```
function matrix modelsel2 (list xvars)
    ols xvars --quiet
    scalar sse = $ess
    scalar N = $nobs
    scalar K = nelem(xvars)-1
    scalar aic = ln(sse/N)+2*K/N
    scalar bic = ln(sse/N)+K*ln(N)/N
    matrix A = { K, N, aic, bic}
  %  printf "\nDependent variable and Regressors: %s\n",varname(xvars)
  %  printf "K = %d, N = %d, AIC = %.4f SC = %.4f.\n",K,N,aic,bic
    return A
end function
```

This is exactly the same version of the model selection function used in chapter 9. Refer to page 217 for details. We will use it in a loop to select lag lengths for the ADF regressions.

The ADF regressions require only a single loop over the lagged differences in the series. The loop is:

```
diff b
matrix A = {}
loop p = 1..4  --quiet
    list vars = d_b(0 to -p) b(-1) const
    matrix a = p~modelsel2(vars)
    matrix A = A | a
endloop
colnames(A,"p K N AIC SC ")
print A
```

The matrix result is:

```
          p             K             N           AIC            SC
     1.0000        3.0000        104.00       -1.3674       -1.2911
     2.0000        4.0000        104.00       -1.3544       -1.2527
     3.0000        5.0000        104.00       -1.4241       -1.2970
     4.0000        6.0000        104.00       -1.4291       -1.2765
```

The AIC is minimized at 4 lags and the SC at 3.

Finally, here is a way to automate the model selection process for an arbitrary time-series. We want to be able to feed the `modelsel2` function the name of a time-series to compute an ADF regression. We'll create another function that has two inputs; one for the maximum lag over which to search and another for the name of the series. The loop will create the differences, generate the variable list needed to estimate a ADF regression that contains a constant, send the list to the `modelsel2` routine, and collect its output into a matrix, and print the matrix. It is fairly simple, but effective.

```
function scalar modelseladf (scalar p, series *y)
    diff y
    matrix A = {}
    loop i = 1..p  --quiet
        list vars = d_y(0 to -i) y(-1) const
        matrix a = i~modelsel2(vars)
        matrix A = A | a
    endloop
    colnames(A,"p K N AIC SC ")
    print A
    return 0
end function
```

The return is a scalar that will be equal to zero, provided everything in the function is executed. The other small difference is that a pointer to the series is used instead of the series itself. This is mainly a way to save some memory and is not strictly necessary. The pointer to the series input in the function is preceded by a `*`. To call the function to get model selection statistics for the inflation series use

```
modelseladf(5, &inf)
```

Because we used a pointer in the function, the series (`inf`) has to be preceded by the ampersand (`&`). The output is printed to the screen:

```
 modelseladf(5, &inf)
A (5 x 5)
             p            K             N           AIC            SC
        1.0000       3.0000        104.00       -1.0233      -0.94704
        2.0000       4.0000        104.00       -1.0257      -0.92404
        3.0000       5.0000        104.00       -1.0906      -0.96351
        4.0000       6.0000        104.00       -1.2439       -1.0914
        5.0000       7.0000        104.00       -1.3028       -1.1248
```

The inflation series appears to have longer lags than the two interest rate series. *AIC* and *SC* are minimized at 5 lags. Schwert (1989) proposed that for $N > 100$ the maximum lag be set to $k_{max} = int[12(T+1)/100]^{0.25}$. If your sample is smaller then use $k_{max} = int[4(T+1)/100]^{0.25}$.[1]

Once you are finished with these functions you can use the `function modelsel2 modelseladf clear` command to remove them from memory.

12.3.1 Other Tests for Nonstationarity

There are other tests for nonstationarity in **gretl** that you may find useful. The first is the DF-GLS test. It performs the modified Dickey-Fuller t-test (known as the DF-GLS test) proposed by Elliott et al. (1996). Essentially, the test is an augmented Dickey-Fuller test, similar to the test performed by **gretl**'s `adf` command, except that the time-series is transformed via a generalized least squares (GLS) regression before estimating the model. Elliott et al. (1996) and others have shown that this test has significantly greater power than the previous versions of the augmented Dickey-Fuller test. Consequently, it is not unusual for this test to reject the null of nonstationarity when the usual augmented Dickey-Fuller test does not.

The `--gls` option performs the DF-GLS test for a series of models that include 1 to k lags of the first differenced, detrended variable. The lag `k` can be set by the user or by the method described in Schwert (1989). As discussed above and in *POE4*, the augmented Dickey-Fuller test involves fitting a regression of the form

$$\Delta y_t = \alpha + \beta y_{t-1} + \delta t + \zeta_1 \Delta y_{t-1} + ... + \zeta_k \Delta y_{t-k} + u_t \tag{12.4}$$

and then testing the null hypothesis $H_0 : \beta = 0$. The DF-GLS test is performed analogously but on GLS-demeaned or GLS-detrended data. The null hypothesis of the test is that the series is a random walk, possibly with drift. There are two possible alternative hypotheses: y_t is stationary about a linear time trend or stationary with a possibly nonzero mean but with no linear time trend. Thus, you can use the `--c` or `--ct` options.

For the levels of the Fed funds rate:

```
Augmented Dickey-Fuller (GLS) test for f
including 6 lags of (1-L)f (max was 12)
```

[1]This tip provided via the **gretl** users group by Grzegorz Konat.

```
sample size 97
unit-root null hypothesis: a = 1

   with constant and trend
   model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: 0.010
   lagged differences: F(6, 90) = 16.433 [0.0000]
   estimated value of (a - 1): -0.115012
   test statistic: tau = -4.14427

                     10%     5%     2.5%     1%
   Critical values: -2.74   -3.03   -3.29   -3.58
```

The test statistic is -4.14, which is in the 1% rejection region for the test. The series is nonstationary. Notice that the lag selected was 6 and that all available observations were used to estimate the model at this point. This is somewhat different from Stata's implementation, which sets the sample to the maximum available for the largest model. Also, notice that we used a trend. This is optional.

For the levels of the Fed funds rate:

```
Augmented Dickey-Fuller (GLS) test for b
including 5 lags of (1-L)b (max was 12)
sample size 98
unit-root null hypothesis: a = 1

   with constant and trend
   model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: -0.005
   lagged differences: F(5, 92) = 4.799 [0.0006]
   estimated value of (a - 1): -0.128209
   test statistic: tau = -3.17998

                     10%     5%     2.5%     1%
   Critical values: -2.74   -3.03   -3.29   -3.58
```

The test statistic is -3.18, which is in the 5% rejection region for the test. The series is nonstationary. Notice that the lag selected was 5 and that one more observation is available to obtain the test statistic.

Gretl also can perform the KPSS test proposed by Kwiatkowski et al. (1992). The `kpss` command computes the KPSS test for each of the specified variables (or their first difference, if the `--difference` option is selected). The null hypothesis is that the variable in question is stationary, either around a level or, if the `--trend` option is given, around a deterministic linear trend.

The statistic itself is very simple

$$\eta = \frac{\sum_{i=1}^{T} S_t^2}{T^2 \tilde{\sigma}^2} \tag{12.5}$$

where $S_t = \sum_{i=1}^{T} e_s$ and $\tilde{\sigma}^2$ is an estimate of the long-run variance of $e_t = (y_t - \bar{y})$. The long run variance is estimated using a bandwidth parameter, m, that the user chooses.

$$\tilde{\sigma}^2 = \sum_{i=-m}^{m} \left(1 - \frac{|i|}{(m+1)}\right) \hat{\gamma}_i \tag{12.6}$$

and where $\hat{\gamma}_i$ is an empirical autocovariance of e_t from order $-m$ to m.

The command calls for the a bandwidth parameter, m (see section 9.6.2 for a brief discussion). For this estimator to be consistent, m must be large enough to accommodate the short-run persistence of e_t, but not too large compared to the sample size T. If you supply a 0, **gretl** will compute an automatic bandwidth of $4(T/100)^{1/4}$.

```
kpss 0 f b
```

The KPSS statistics using automatic bandwidth selection results in:

```
KPSS test for f

T = 104
Lag truncation parameter = 4
Test statistic = 1.36747

                      10%      5%      1%
Critical values: 0.349   0.466   0.734

KPSS test for b

T = 104
Lag truncation parameter = 4
Test statistic = 1.72833

                      10%      5%      1%
Critical values: 0.349   0.466   0.734
```

Both are significantly different from zero and the stationary null hypothesis is rejected at any reasonable level of significance. Also note that the bandwidth was chosen to be 4.

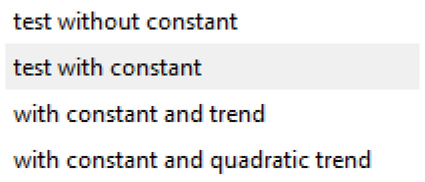
12.4 Cointegration

Two nonstationary series are cointegrated if they tend to move together through time. For instance, we have established that the levels of the Fed Funds rate and the 3-year bond are nonstationary, whereas their differences are stationary. In the opaque language used in time-series literature, each series is said to be “integrated of order 1” or I(1). If the two nonstationary series move together through time then we say they are “cointegrated.” Economic theory would suggest that they should be tied together via arbitrage, but that is no guarantee. In this context, testing for cointegration amounts to a test of the substitutability of these assets.

The basic test is very simple. Regress one I(1) variable on another using least squares. If the series are cointegrated, the residuals from this regression will be stationary. This is verified using augmented Dickey-Fuller test, with a new set of critical values that take into account that the series of residuals used in the test is estimated from data.

The null hypothesis is that the residuals are nonstationary, which implies that the series are not cointegrated. Rejection of this leads to the conclusion that the series are cointegrated. The `coint` function in **gretl** carries out each of the three steps in this test. First, it carries out a Dickey-Fuller test of the null hypothesis that each of the variables listed has a unit root. Then it estimates the cointegrating regression using least squares. Finally, it runs a Dickey Fuller test on the residuals from the cointegrating regression. This procedure, referred to as the Engle-Granger (EG) cointegration test and discussed in chapter 12 of Hill et al. (2011), is the one done in **gretl** by default. **Gretl** can also perform cointegration tests based on maximum likelihood estimation of the cointegrating relationships proposed by Johansen and summarized in Hamilton (1994, chapter 20). The Johansen tests use the `coint2` command, which is explained in **gretl**’s documentation (Cottrell and Lucchetti, 2011, chapter 24).

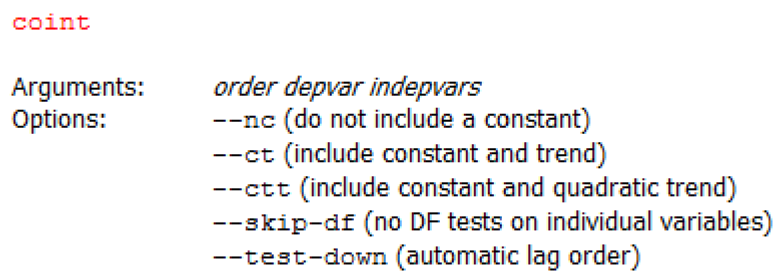
Figure 12.8 shows the dialog box used to test cointegration in this way. To obtain it use `Model>Time series>Cointegration test>Engle-Granger` from the main **gretl** window. In the dialog box you have to indicate how many lags you want in the initial Dickey-Fuller regressions on the the variables, which variables you want to include in the cointegrating relationship, and whether you want a constant, trend, or quadratic trend in the regressions. Testing down from the maximum lag order is chosen via a check-box. To select these additional modeling options you’ll have to click on the down arrow button indicated in Figure 12.8. This will reveal the four choices:



We are choosing the model that contains a constant, which is the default. For the 3-year bond rate and the Fed funds rate series we get the result shown in Figure 12.9.

Since the `--skip-df` option is used, there are only two steps shown in the output. The first is the outcome of the cointegrating regression. It is just a linear regression of `b` and a constant on `f`. The residuals are automatically generated and passed to step 2 that performs the EG test. The model selection occurs because the `--test-down` option is used, which picks a model with 3 lags. The test statistic and its p-value are circled at the bottom. The statistic is -4.32 and it is significant at the 5% level. The unit root null hypothesis is rejected and we conclude that the series are cointegrated.

The syntax and options available for the Engle-Granger test are summarized:



```
coint

Arguments:   order depvar indepvars
Options:     --nc (do not include a constant)
             --ct (include constant and trend)
             --ctt (include constant and quadratic trend)
             --skip-df (no DF tests on individual variables)
             --test-down (automatic lag order)
```

If the specified lag order is positive all the Dickey-Fuller tests use that order, with this qualification: if the `--test-down` option is used, the given value is taken as the maximum and the actual lag order used in each case is obtained by testing down. Basically, a series of *t*-tests on the last lag is used until the last one becomes significant at 10% level.

The syntax for Engle-Granger tests from a script from the console follows

```
coint 4 f b --test-down --skip-df
```

Notice that a maximum of 4 lags are considered; the `--test-down` option will attempt to automatically reduce that number according to the algorithm discussed above. Also, we have chosen to skip the Dickey-Fuller tests for stationarity of `f` and `b` since they have already been done and discussed above.

12.5 Error Correction

Cointegration is a relationship between two nonstationary, I(1), variables. These variables share a common trend and tend to move together in the long-run. In this section, a dynamic relationship between I(0) variables that embeds a cointegrating relationship known as the **short-run error correction model** is examined.

Start with an ARDL(1,1)

$$y_t = \delta + \theta_1 y_{t-1} + \delta_0 x_t + \delta_1 x_{t-1} + v_t \tag{12.7}$$

after some manipulation (see *POE4* for details)

$$\Delta y_t = -(1-\theta_1)(y_{t-1} - \beta_1 - \beta_2 x_{t-1}) + \delta_0 \Delta x_t + \delta_1 \Delta x_{t-1} + v_t \tag{12.8}$$

The term in the second set of parentheses is a cointegrating relationship. The levels of y and x are linearly related. Let $\alpha = (1 - \theta_1)$ and the equation's parameters can be estimated by **nonlinear least squares**.

In **gretl** this is easiest done in a script. There are basically three steps. First open the data and create lags and differences. Second, decide upon reasonable starting values for the numerical optimization procedure. Finally, specify the equation to be estimated by `nls`.

```
1  open "@gretldir\data\poe\usa.gdt"
2  lags 2; f b
3  diff f b
4  ols b const f
5  scalar b1 = $coeff(const)
6  scalar b2 = $coeff(f)
7  ols d_b const d_f(0 to -1)
8  scalar d0 = $coeff(d_f)
9  scalar d1 = $coeff(d_f_1)
10 ols b const b(-1) f(0 to -1)
11 scalar a = 1-$coeff(b_1)
12 nls d_b=-a*(b_1-b1-b2*f_1)+d0*d_f+d1*d_f(-1)
13     params a b1 b2 d0 d1
14 end nls
```

The hardest part of this is giving the routine a decent set of starting values. Here, I used three separate linear regressions to generate reasonable starting values. I estimated the cointegrating relationship via least squares in line 4 to populate β_1 and β_2. To get start values for δ_0 and δ_1 I used a similar strategy, estimating the ARDL(1,1) in its difference form (equation 12.8). Estimate that regression without the cointegrating relationship in it and use these parameters as the starting values. For the parameter `a`, I estimated equation (12.7) and used $a_0 = (1 - \hat{\theta})$.

Since I am reluctant to taking derivatives analytically unless I have to, I tried to estimate the model without them, relying on **gretl**'s excellent numerical versions. The `params` statement in line 13 is required when using numerical derivatives. Fortunately, **gretl** rewarded me with the correct result (as verified in Eviews and Stata).

```
Using numerical derivatives
Tolerance = 1.81899e-012
Convergence achieved after 19 iterations

NLS, using observations 1984:3-2009:4 (T = 102)
d_b = -a*(b_1-b1-b2*f_1)+d0*d_f+d1*d_f(-1)

             estimate    std. error   t-ratio    p-value
  -------------------------------------------------------
  a          0.141877    0.0496561     2.857    0.0052    ***
  b1         1.42919     0.624625      2.288    0.0243    **
  b2          0.776557   0.122475      6.341    7.25e-09  ***
  d0          0.842463   0.0897482     9.387    2.83e-015 ***
  d1         -0.326845   0.0847928    -3.855    0.0002    ***

Mean dependent var  -0.110294   S.D. dependent var   0.537829
Sum squared resid    14.18070   S.E. of regression   0.382352
R-squared            0.514614   Adjusted R-squared   0.494598
Log-likelihood      -44.10409   Akaike criterion     98.20819
Schwarz criterion    111.3331   Hannan-Quinn         103.5229
rho                  0.126909   Durbin-Watson        1.745112
```

Once the model is estimated, you can get the implied estimate of θ_1.

```
scalar theta1 = 1-$coeff(a)
```

which is 0.858123. You can also perform an EG test for stantionarity by constructing the residuals and using `adf`. In this case, you'll have to consult a table of critical values since the EG ones are not available from `adf` routine.

```
series ehat = b-$coeff(b1)-$coeff(b2)*f
adf 1 ehat --nc
```

As before, the null is that (`b`, `f`) are not cointegrated. Since the cointegrating relationship includes a constant term, the critical value is (-3.37). Comparing the calculated value -3.92668 with the critical value, we reject the null hypothesis and conclude that (`b`, `f`) are cointegrated.

12.6 Script

```
open "@gretldir\data\poe\usa.gdt"
set echo off
# take differences
diff b inf f gdp
# change variable attributes
setinfo b -d "3-year Bond rate" -n "3-year Bond rate"
setinfo d_b -d "Change in the 3-year Bond rate" -n "D.BOND"
setinfo inf -d "annual inflation rate" -n "inflation rate"
setinfo d_inf -d "Change in the annual inflation rate" -n "D.INFLATION"
setinfo gdp -d "real US gross domestic product" -n "Real GDP"
setinfo d_gdp -d "= first difference of gdp" -n "D.GDP"
setinfo f -d "federal funds rate" -n "Fed Funds Rate"
setinfo d_f -d "= first difference of f" -n "D.FED_FUNDS"
```

```

# multiple time series plots
scatters inf d_inf f d_f
scatters b d_b gdp d_gdp

# summary statistics for subsamples and full sample
smpl 1984:2 1996:4
summary --simple
smpl 1997:1 2009:4
summary --simple
smpl full

list levels = gdp inf f b
summary levels --simple

# spurious regression
open "@gretldir\data\poe\spurious.gdt"
setobs 1 1 --special-time-series
gnuplot rw1 rw2 --with-lines --time-series
ols rw1 rw2 const

# adf tests
open "@gretldir\data\poe\usa.gdt"
adf 4 f --c --ct --test-down
adf 4 b --c --test-down --verbose

function scalar modelseladf (scalar p, series *y)
    diff y
    matrix A = {}
    loop i = 1..p  --quiet
        list vars = d_y(0 to -i) y(-1) const
        matrix a = i~modelsel2(vars)
        matrix A = A | a
    endloop
    colnames(A,"p K N AIC SC ")
    print A
    return 0
end function

# model selection for adf tests
matrix a = modelseladf(4, &b)
matrix a = modelseladf(4, &f)

smpl full
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
adf mlag f --ct --gls --test-down
adf mlag b --ct --gls --test-down

# kpss test
kpss 0 f b
coint 4 f b inf --test-down --skip-df
```

```

open "@gretldir\data\poe\usa.gdt"
lags 2; f b
diff f b

# nls estimation of cointegrating vector
ols b const f
scalar b1 = $coeff(const)
scalar b2 = $coeff(f)
ols d_b const d_f(0 to -1)
scalar d0 = $coeff(d_f)
scalar d1 = $coeff(d_f_1)
ols b const b(-1) f(0 to -1)
scalar a = 1-$coeff(b_1)
nls d_b=-a*(b_1-b1-b2*f_1)+d0*d_f+d1*d_f(-1)
    params a b1 b2 d0 d1
end nls
scalar theta1 = 1-$coeff(a)
series ehat = b-$coeff(b1)-$coeff(b2)*f
adf 1 ehat --nc
```

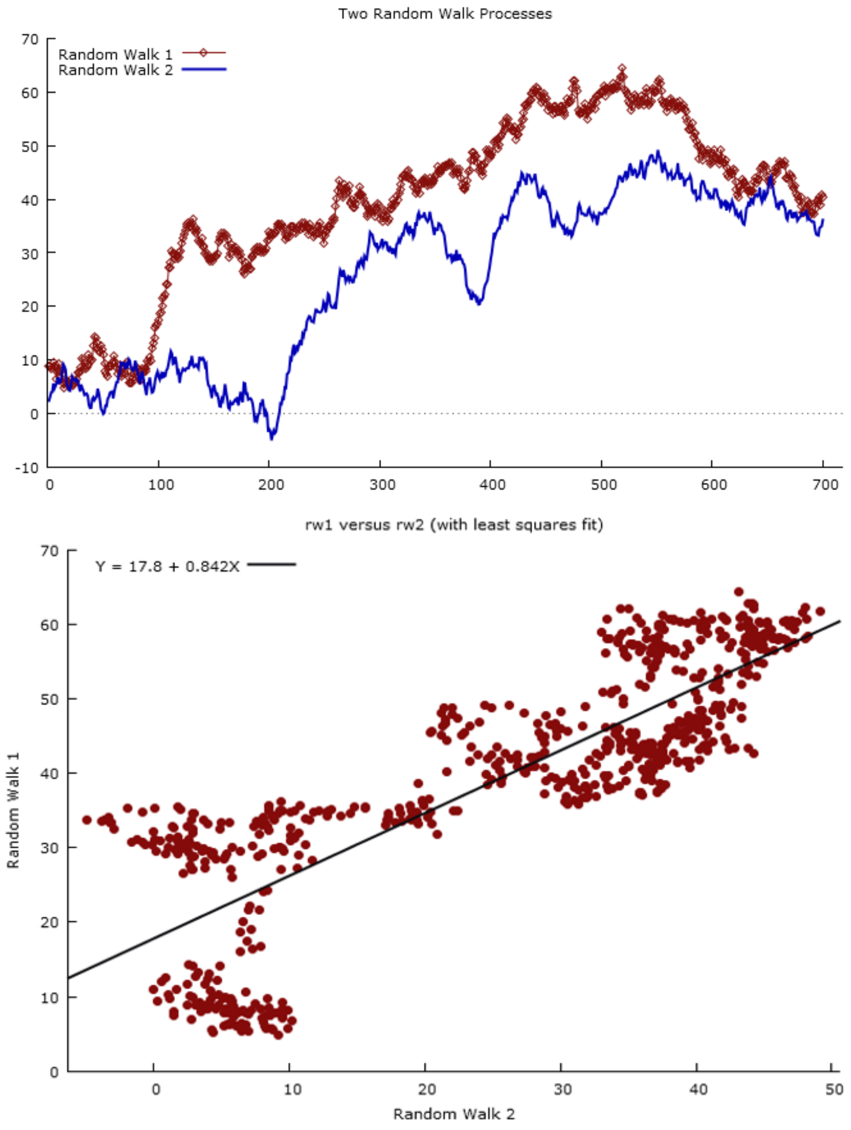


Figure 12.4: The two independent random walk series appear to be related. The top graph is a simple time-series plot and the bottom is an XY scatter with least squares fit.

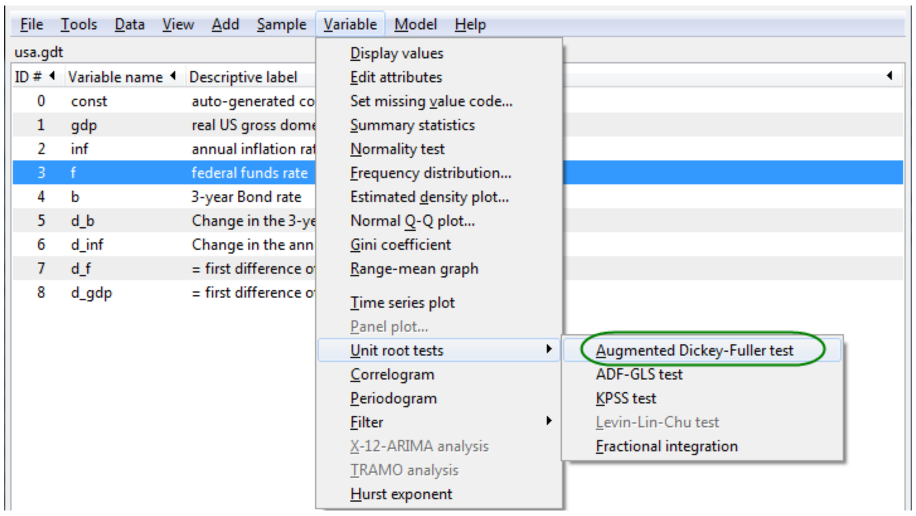
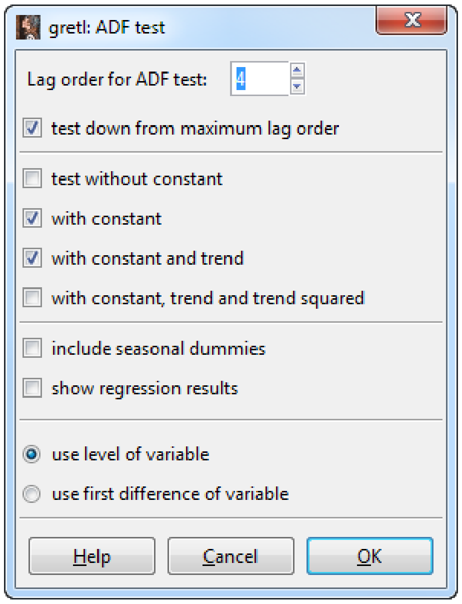


Figure 12.5: Choose the ADF test from the pull-down menu.



Lag order for ADF test: 4

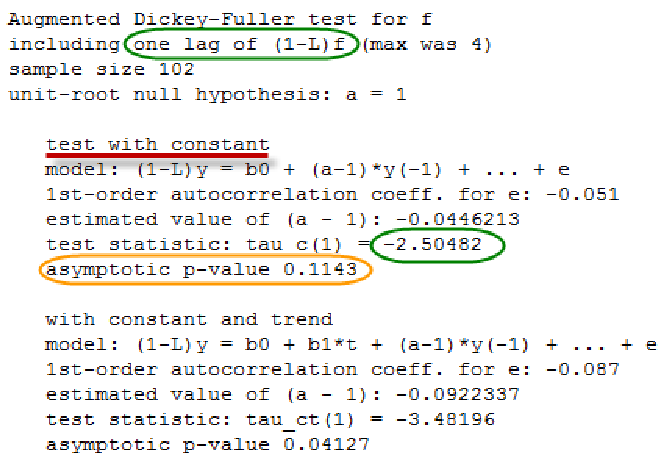
[✓] test down from maximum lag order

[ ] test without constant
[✓] with constant
[✓] with constant and trend
[ ] with constant, trend and trend squared

[ ] include seasonal dummies
[ ] show regression results

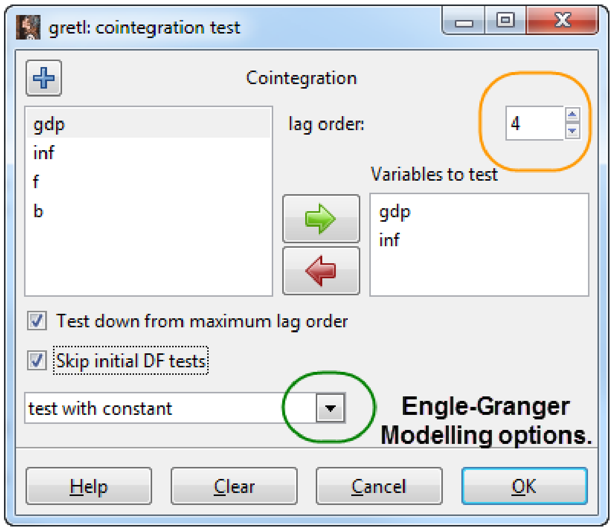
(●) use level of variable
( ) use first difference of variable

Figure 12.6: The ADF test dialog box.



```
Augmented Dickey-Fuller test for f
including one lag of (1-L)f (max was 4)
sample size 102
unit-root null hypothesis: a = 1

   test with constant
   model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: -0.051
   estimated value of (a - 1): -0.0446213
   test statistic: tau_c(1) = -2.50482
   asymptotic p-value 0.1143

   with constant and trend
   model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: -0.087
   estimated value of (a - 1): -0.0922337
   test statistic: tau_ct(1) = -3.48196
   asymptotic p-value 0.04127
```

Figure 12.7: The ADF test results.



gretl: cointegration test

Cointegration

gdp
inf
f
b

lag order: 4

Variables to test
gdp
inf

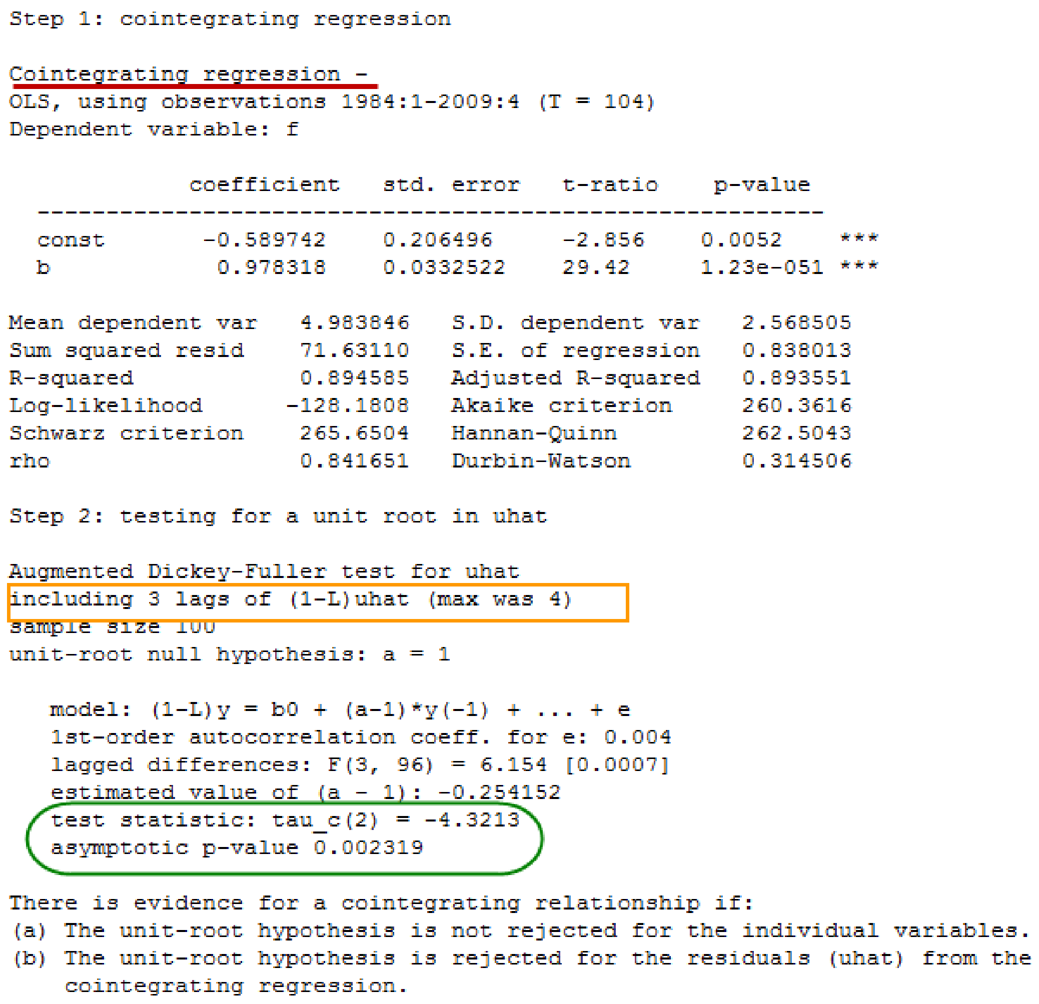
[✓] Test down from maximum lag order

[✓] Skip initial DF tests

test with constant

Engle-Granger Modelling options.

Figure 12.8: The dialog box for the cointegration test.



```
Step 1: cointegrating regression

Cointegrating regression -
OLS, using observations 1984:1-2009:4 (T = 104)
Dependent variable: f

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const       -0.589742    0.206496     -2.856    0.0052    ***
  b            0.978318    0.0332522    29.42     1.23e-051 ***

Mean dependent var   4.983846   S.D. dependent var   2.568505
Sum squared resid    71.63110   S.E. of regression   0.838013
R-squared            0.894585   Adjusted R-squared   0.893551
Log-likelihood      -128.1808   Akaike criterion     260.3616
Schwarz criterion    265.6504   Hannan-Quinn         262.5043
rho                  0.841651   Durbin-Watson        0.314506

Step 2: testing for a unit root in uhat

Augmented Dickey-Fuller test for uhat
including 3 lags of (1-L)uhat (max was 4)
sample size 100
unit-root null hypothesis: a = 1

   model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: 0.004
   lagged differences: F(3, 96) = 6.154 [0.0007]
   estimated value of (a - 1): -0.254152
   test statistic: tau_c(2) = -4.3213
   asymptotic p-value 0.002319

There is evidence for a cointegrating relationship if:
(a) The unit-root hypothesis is not rejected for the individual variables.
(b) The unit-root hypothesis is rejected for the residuals (uhat) from the
    cointegrating regression.
```

Figure 12.9: The results from the Engle-Granger test. The output from the Dickey-Fuller regressions is suppressed using the `the --skip-df` option.

Chapter 13

Vector Error Correction and Vector Autoregressive Models: Introduction to Macroeconometrics

The vector autoregression model is a general framework used to describe the dynamic interrelationship between stationary variables. So, the first step in your analysis should be to determine whether the levels of your data are stationary. If not, take the first differences of your data and try again. Usually, if the levels (or log-levels) of your time-series are not stationary, the first differences will be.

If the time-series are not stationary then the VAR framework needs to be modified to allow consistent estimation of the relationships among the series. The vector error correction model (VECM) is just a special case of the VAR for variables that are stationary in their differences (i.e., I(1)). The VECM can also take into account any cointegrating relationships among the variables.

13.1 Vector Error Correction and VAR Models

Consider two time-series variables, y_t and x_t. Generalizing the discussion about dynamic relationships in chapter 9 to these two interrelated variables yield a system of equations:

$$y_t = \beta_{10} + \beta_{11}y_{t-1} + \beta_{12}x_{t-1} + v_t^y \tag{13.1}$$
$$x_t = \beta_{20} + \beta_{21}y_{t-1} + \beta_{22}x_{t-1} + v_t^x \tag{13.2}$$

The equations describe a system in which each variable is a function of its own lag, and the lag of the other variable in the system. Together the equations constitute a system known as a vector autoregression (VAR). In this example, since the maximum lag is of order one, we have a VAR(1).

If y and x are stationary, the system can be estimated using least squares applied to each equation. If y and x are not stationary in their levels, but stationary in differences (i.e., I(1)), then take the differences and estimate:

$$\Delta y_t = \beta_{11}\Delta y_{t-1} + \beta_{12}\Delta x_{t-1} + v_t^{\Delta y} \tag{13.3}$$

$$\Delta x_t = \beta_{21}\Delta y_{t-1} + \beta_{22}\Delta x_{t-1} + v_t^{\Delta x} \tag{13.4}$$

using least squares. If y and x are I(1) and cointegrated, then the system of equations can be modified to allow for the cointegrating relationship between the I(1) variables. Introducing the cointegrating relationship leads to a model known as the vector error correction (VEC) model.

In this example from *POE4*, we have macroeconomic data on real GDP for a large and a small economy; *usa* is real quarterly GDP for the United States and *aus* is the corresponding series for Australia. The data are found in the *gdp.gdt* dataset and have already been scaled so that both economies show a real GDP of 100 in the year 2000. We decide to use the vector error correction model because (1) the time-series are not stationary in their levels but are in their differences (2) the variables are cointegrated.

In an effort to keep the discussion moving, the authors of *POE4* opted to avoid discussing how they actually determined the series were nonstationary in levels, but stationary in differences. This is an important step and I will take some time here to explain how one could approach this. There are several ways to do this and I'll show you two ways to do it in **gretl**.

13.1.1 Series Plots–Constant and Trends

Our initial impressions of the data are gained from looking at plots of the two series. The data plots are obtained in the usual way after importing the dataset. The data on U.S. and Australian GDP are found in the *gdp.gdt* file and were collected from 1970:1 - 2004:4.[1] Open the data and set the data structure to quarterly time-series using the `setobs 4` command, start the series at 1970:1, and use the `--time-series` option.

```
open "@gretldir\data\poe\gdp.gdt"
setobs 4 1970:1 --time-series
```

One purpose of the plots is to help you determine whether the Dickey-Fuller regressions should contain constants, trends or squared trends. The simplest way to do this is from the console using the `scatters` command.

```
scatters usa diff(usa) aus diff(aus)
```

The `scatters` command produces multiple graphs, each containing one of the listed series. The `diff()` function is used to take the differences of `usa` and `aus`, which appear in the graphs featured in Figure 13.1 below.

[1] *POE4* refers to these variables as U and A, respectively.

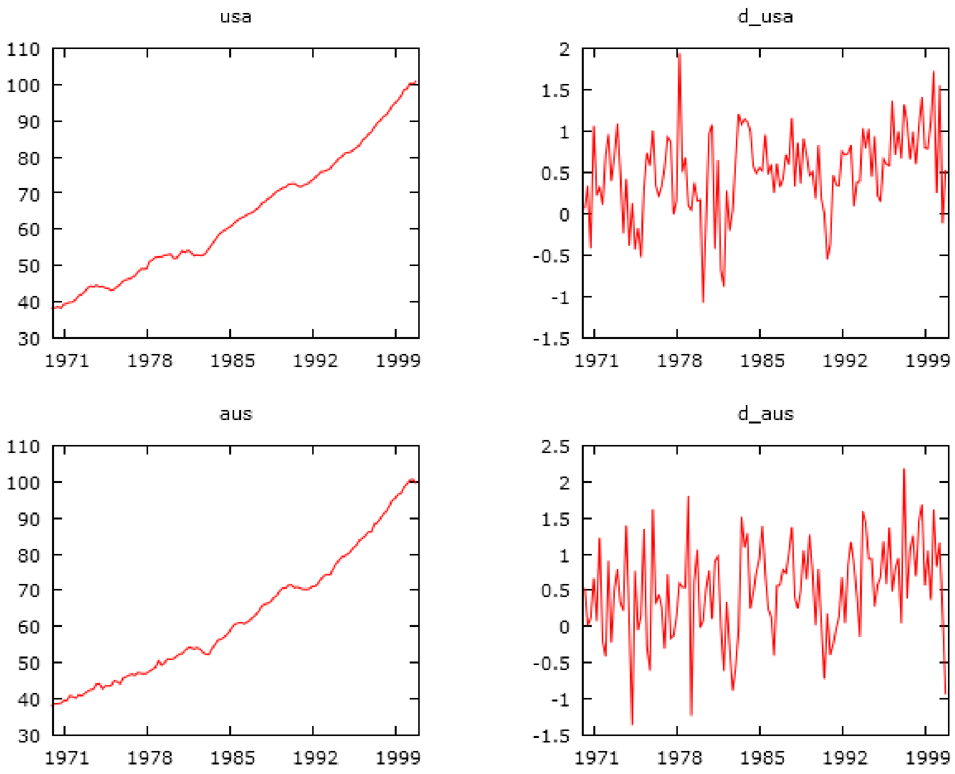


Figure 13.1: The levels of Australian and U.S. GDP appear to be nonstationary and cointegrated. The difference plots have a nonzero mean, indicating a constant in their ADF regressions.

This takes two steps from the pull-down menu. First, use the mouse to highlight the two series and then create the differences using `Add>First differences of selected variables`. Then, select `View>Multiple graphs>Time series`. Add the variables to the selected list box to produce Figure 13.1.

From the time-series plots it appears that the levels are mildly parabolic in time. The differences have a small trend. This means that the augmented Dickey-Fuller (ADF) regressions may need to contain these elements.

13.1.2 Selecting Lag Length

The second consideration is the specification of lags for the ADF regressions. There are several ways to select lags and **gretl** automates one of these. The basic concept is to include enough lags in the ADF regressions to make the residuals white noise. These will be discussed presently.

Testing Down

The first strategy is to include just enough lags so that the last one is statistically significant. **Gretl** automates this using the `--test-down` option for the augmented Dickey-Fuller regressions. Start the ADF regressions with a generous number of lags and **gretl** automatically reduces that number until the t-ratio on the longest remaining lag is significant at the 10 percent level. For the levels series we choose the maximum number using Schwert's method as discussed in chapter 12. The model includes a constant, trend, and trend squared (`--ctt` option), and use the `--test-down` option.

```
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
adf mlag usa --ctt --test-down
adf mlag aus --ctt --test-down
```

The USA series contains a very long significant lag twelve periods into the past. The Australian series shows much less persistence, choosing only 3 in testing down from 12. The result is shown in Figure 13.2. Both ADF statistics are insignificant at the 5% or 10% level, indicating they are nonstationary. This is repeated for the differenced series using the commands:

```
adf mlag diff(usa) --ct --test-down
adf mlag diff(aus) --ct --test-down
```

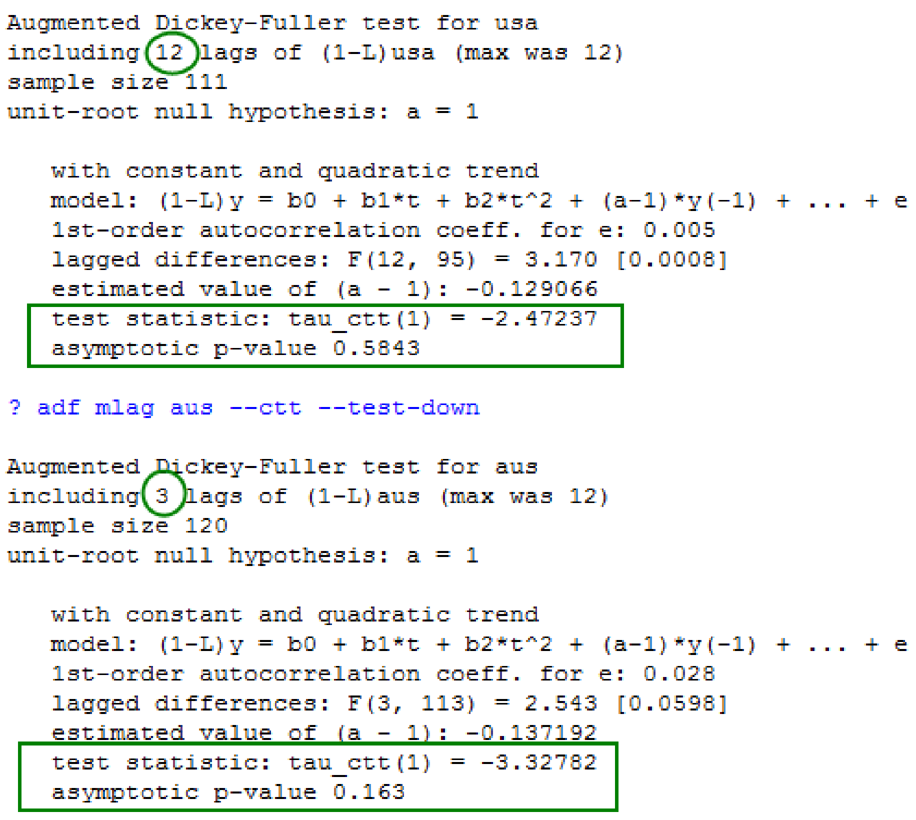
The selected lags for the U.S. and Australia are eleven and seven, respectively. Both ADF statistics are significant at the 5% level and we conclude that the differences are stationary.

Testing Up

The other strategy is to test the residuals from the augmented Dickey-Fuller regressions for autocorrelation. In this strategy you can start with a small model, and test the residuals of the Dickey-Fuller regression for autocorrelation using an LM test. If the residuals are autocorrelated, add another lagged difference of the series to the ADF regression and test the residuals again. Once the LM statistic is insignificant, you quit you are done. This is referred to as **testing-up**. You will still need to start with a reasonable number of lags in the model or the tests will not have desirable properties.

To employ this strategy in **gretl**, you'll have to estimate the ADF equations manually using the `ols` command. Since the data series has a constant and quadratic trend, you have to define a time trend (`genr time`) and possibly trend squared (`square time`) to include in the regressions.[2]

[2] It was not apparent from the plots of the differenced series that a squared trend was required. However, the squared trend was included in the model because it is statistically significant in each of the ADF regressions.



```
Augmented Dickey-Fuller test for usa
including 12 lags of (1-L)usa (max was 12)
sample size 111
unit-root null hypothesis: a = 1

   with constant and quadratic trend
   model: (1-L)y = b0 + b1*t + b2*t^2 + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: 0.005
   lagged differences: F(12, 95) = 3.170 [0.0008]
   estimated value of (a - 1): -0.129066
   test statistic: tau_ctt(1) = -2.47237
   asymptotic p-value 0.5843

? adf mlag aus --ctt --test-down

Augmented Dickey-Fuller test for aus
including 3 lags of (1-L)aus (max was 12)
sample size 120
unit-root null hypothesis: a = 1

   with constant and quadratic trend
   model: (1-L)y = b0 + b1*t + b2*t^2 + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: 0.028
   lagged differences: F(3, 113) = 2.543 [0.0598]
   estimated value of (a - 1): -0.137192
   test statistic: tau_ctt(1) = -3.32782
   asymptotic p-value 0.163
```

Figure 13.2: Based on ADF tests, the levels of Australian and U.S. GDP are nonstationary.

Note this is another one of those cases where you cannot use `series` in place of `genr`. The `genr time` is a special function for `genr`. The other cases include `genr dummy` and `genr unitdum`. You will also need to generate the differences to use in a new function called `lags`. The script to do this follows:

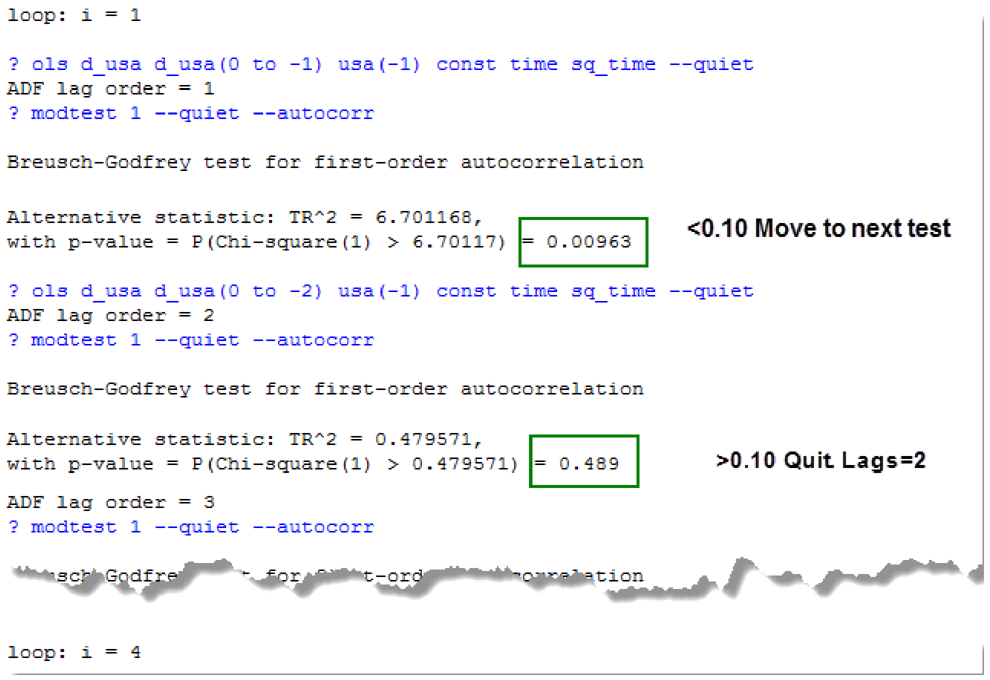
```
genr time
square time
diff usa aus
```

Now, estimate a series of augmented Dickey-Fuller regressions using `ols`. Follow each regression with the *LM* test for autocorrelation of the residuals discussed in chapter 9.

```
loop i=1..12
  ols d_usa(0 to -i) usa(-1)  const time sq_time --quiet
  printf "ADF lag order = %d\n",i
  modtest 1 --autocorr --quiet
end loop
```

The first `ols` regression is the ADF(1). It includes 1 lagged value of the `d_usa` as a regressor in addition to the lagged value of `usa`, a constant, a trend, and a squared trend. **Gretl**'s `variable(i to j)` function creates a series of lags from `i` through `j` of `variable`. So in the first regression, `d_usa(0 to -i)` creates the contemporaneous value and a single lagged value of `d_usa`. Since the contemporaneous value, `d_usa`, appears first in the variable list, it is taken as the dependent variable. A `printf` statement is issued to remind us of which test we are performing. Then the *LM* and other AR tests are conducted using `modtest 1 --autocorr --quiet`. If the *p*-value is greater than 0.10 then, this is your model. If not, consider the outcome of the next loop which has added another lag of `d_usa` to the model. Stop when the *p*-value is greater than 0.10.

In this code example we chose to suppress the results from the first regression so that the output from the tests would fit on one page (Figure 13.3). In practice, you could skip this option and read the Dickey-Fuller *t*-ratio directly from the output. The only disadvantage of this is that the proper *p*-value for it is not computed using the manual approach.



```
loop: i = 1

? ols d_usa d_usa(0 to -1) usa(-1) const time sq_time --quiet
ADF lag order = 1
? modtest 1 --quiet --autocorr

Breusch-Godfrey test for first-order autocorrelation

Alternative statistic: TR^2 = 6.701168,
with p-value = P(Chi-square(1) > 6.70117) = 0.00963

? ols d_usa d_usa(0 to -2) usa(-1) const time sq_time --quiet
ADF lag order = 2
? modtest 1 --quiet --autocorr

Breusch-Godfrey test for first-order autocorrelation

Alternative statistic: TR^2 = 0.479571,
with p-value = P(Chi-square(1) > 0.479571) = 0.489

ADF lag order = 3
? modtest 1 --quiet --autocorr

Breusch-Godfrey test for first-order autocorrelation

loop: i = 4
```

Figure 13.3: Testing up: manually estimate the ADF regressions and use LM tests for autocorrelation to determine the proper lag length.

If you repeat this exercise for `aus` (as we have done in the script at the end of the chapter[3]) you will find that testing up determines ***zero*** lags of `d_aus` are required in the Dickey-Fuller regression; testing down revealed ***three*** lags were needed. The incongruence occurs because we did a poor job of testing up, failing to include enough autocorrelation terms in the *LM* test. This illustrates a danger of testing up. When we conducted the *LM* test using only a single autocorrelation term,

[3]Actually, the *LM* statistic for the ADF(1) was insignificant and a separate DF regression also had an insignificant *LM* statistic, indicating no lags are needed. I made the loop a bit fancier in order to produce the DF statistic by adding a conditional statement for when i=0 as we did earlier in the book.

we had not searched far enough in the past to detect significant autocorrelations that lie further back in time. Adding terms to the autocorrelation test using `modtest 3 --autocorr` would have helped to detect this.

So which is better, testing down or testing up? I think the econometric consensus is that testing down is safer. We'll leave it for future study!

13.1.3 Cointegration Test

Given that the two series are stationary in their differences (i.e., both are I(1)), the next step is to test whether they are cointegrated. In the discussion that follows, we return to reproducing results from *POE4*. To do this, use least squares to estimate the following regression.

$$aus_t = \beta usa_t + e_t \tag{13.5}$$

obtain the residuals, $\hat{e}_t$, and then estimate

$$\Delta\hat{e}_t = \gamma\hat{e}_{t-1} + u_t \tag{13.6}$$

This is the "case 1 test" from chapter 12 of Hill et al. (2011) and the 5% critical value for the t-ratio is -2.76. The following script estimates the model cointegrating regression, saves the residuals, and estimates the regression required for the unit root test.

```
ols aus usa
series uhat = $uhat
ols diff(uhat) uhat(-1)
```

The result is:

$$\widehat{\Delta e_t} = \underset{(0.044279)}{-0.127937}\hat{e}_{t-1} \tag{13.7}$$

$$T = 123 \quad \bar{R}^2 = 0.0640 \quad F(1,122) = 8.3482 \quad \hat{\sigma} = 0.5985$$
(standard errors in parentheses)

The t-ratio is $-0.1279/.0443 = -2.889$ which lies in the rejection region for this test. Therefore, you reject the null hypothesis of no cointegration.

13.1.4 VECM: Australian and U.S. GDP

You have two difference stationary series that are cointegrated. Consequently, an error correction model of the short-run dynamics can be estimated using least squares. A simple error correction model is:

$$\Delta aus_t = \beta_{11} + \beta_{12}\hat{e}_{t-1} + v_{1t} \tag{13.8}$$

$$\Delta usa_t = \beta_{21} + \beta_{22}\hat{e}_{t-1} + v_{2t} \tag{13.9}$$

and the estimates

$$\widehat{\Delta aus}_t = \underset{(8.491)}{0.491706} + \underset{(-2.077)}{-0.0987029}\hat{e}_{t-1}$$

$$\widehat{\Delta usa}_t = \underset{(10.924)}{0.509884} + \underset{(0.790)}{+0.0302501}\hat{e}_{t-1}$$

(t-statistics in parentheses)

which are produced using

```
ols diff(aus) const uhat(-1)
ols diff(usa) const uhat(-1)
```

The significant negative coefficient on $\hat{e}_{t-1}$ indicates that Australian GDP responds to a temporary disequilibrium between the U.S. and Australia.

The U.S. does not appear to respond to a disequilibrium between the two economies; the t-ratio on $\hat{e}_{t-1}$ is insignificant. These results support the idea that economic conditions in Australia depend on those in the U.S. more than conditions in the U.S. depend on Australia. In a simple model of two economy trade, the U.S. is a large closed economy and Australia is a small open economy.

13.1.5 Using gretl's `vecm` Command

The Australian/U.S. GDP example above was carried out manually in a series of steps in order to familiarize you with the structure of the VEC model and how, at least in principle, they are estimated. In most applications, you will probably use other methods to estimate the VECM; they provide additional information that is useful and are usually more efficient. **Gretl** contains a full-featured `vecm` command that estimates a VECM. Chapter 24 of Cottrell and Lucchetti (2011) provides an excellent tutorial on estimating a VECM and includes some examples using **gretl**. Before using the `vecm` command in **gretl**, this is required reading!

One feature of the example in *POE4* that bothers me is that tests for autocorrelation in the error correction models reject the no serial correlation hypothesis. That implies that the lag structure in the error correction models probably needs more thought. Thus, lags are added to the model and it is reestimated using **gretl**'s `vecm` command, the syntax for which is:

vecm

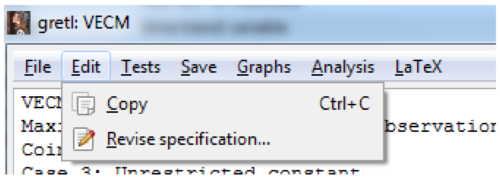
Arguments:	*order rank ylist* [; *xlist*] [; *rxlist*]
Options:	`--nc` (no constant)
	`--rc` (restricted constant)
	`--crt` (constant and restricted trend)
	`--ct` (constant and unrestricted trend)
	`--seasonals` (include centered seasonal dummies)
	`--impulse-responses` (print impulse responses)
	`--variance-decomp` (print variance decompositions)

After some experimentation I decide to use a third order model. Since there are only 2 series, the maximum and only number of cointegrating vectors is 1. The default, 'case 3,' which is an unrestricted constant, is used to model the deterministic components of the model. Choosing the correct case is another part of the art of doing a VECM study and I am not expert enough to give advice on how to do this. I will leave you to your own devices to resolve this tricky issue.

The model is estimated via a script:

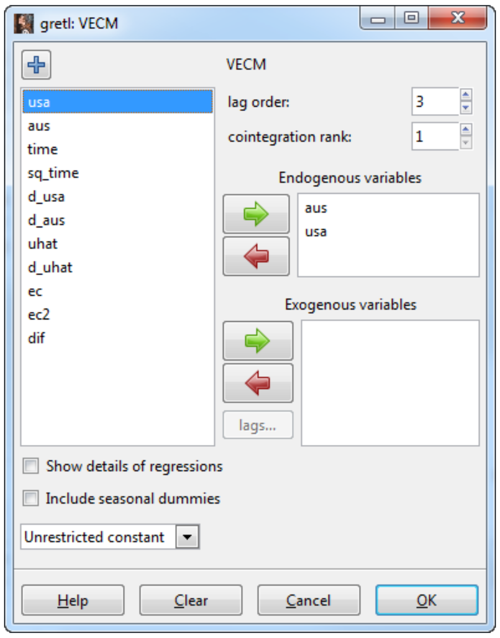
```
3rd order VECM with 1 cointegrating vector--unrestricted constant
vecm 3 1 aus usa
```

The dialog boxes are also useful. Choose `Model>Time-Series>VECM` to bring up the appropriate dialog box shown in Figure 13.4. It allows you to add endogenous variables to the VAR, exogenous variables (which must be I(0)), choose lags, number of cointegrating vectors, and choose the model for the deterministic portion of the trend. One of the advantages of using the dialog is that the model results appear, as usual, in a separate model window. The window gives you immediate access to tests, plots, and additional tools for analysis. Furthermore, there is also a handy facility that allows quick respecificaiton of the model. From the menu bar of the model window choose `Edit>Revise specification` brings up the VECM dialog box again for you to change settings.



```
VECM
Max:                                    bservatior
Coi
Case 3: Unrestricted constant
```

One way to evaluate whether you have made adequate modeling choices is to look at various statistics within the output to check for significance of lags, as well as the magnitudes and signs of the coefficients. Even without the `--verbose` option, the command produces quite a bit of output. Here I divide it into two parts. The first part of our output can be seen below in Figure 13.5 The lag order is given, the selected cointegration rank is shown, and the "case" (unrestricted constant) is identified. Next are the estimates from the cointegrating equation. The adjustment vectors are actually the coefficients on the lagged residuals from the cointegrating relationship. Generally,



lag order: 3
cointegration rank: 1
Endogenous variables: aus, usa
Exogenous variables:
[ ] Show details of regressions
[ ] Include seasonal dummies
Unrestricted constant

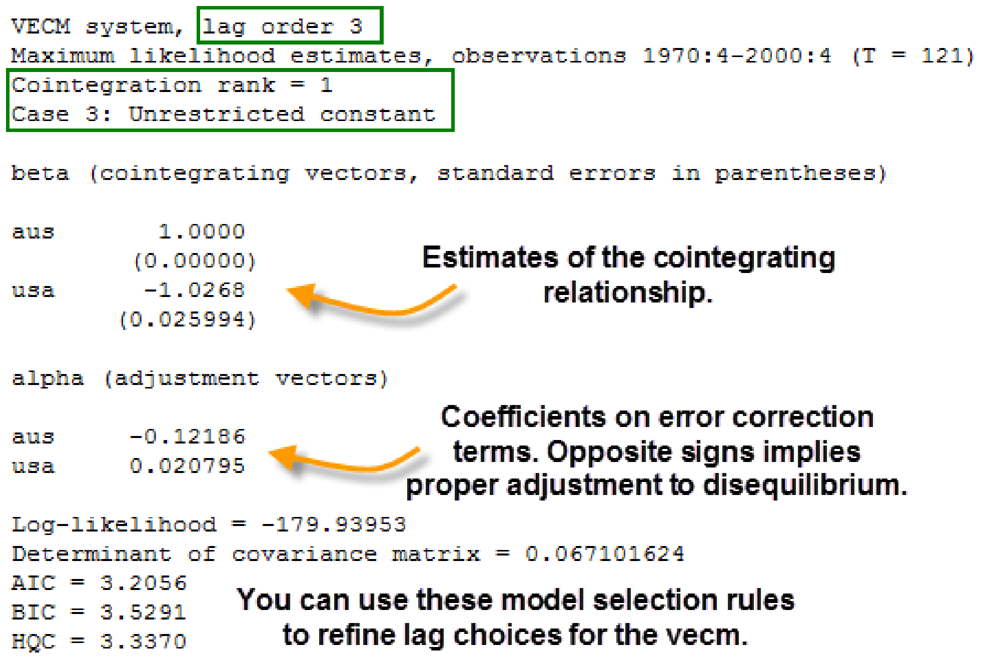
Figure 13.4: The VECM dialog box

these should have opposite signs in two variable models, otherwise the adjustments to shocks may not be equilibrating. Finally, some model selection statistics appear at the bottom that may be useful in determining the order of the VECM. As with the ones computed in our own `modelsel2` function, smaller is better.

The second part of the output appears in Figure 13.6. This shows the estimates from the complete VECM. You will want to check to see if unnecessary lags have been included in the model (insignificant t-ratios on the longest lags), check the value of the Durbin-Watson statistic (it should be close to 2), and check the signs and significance of the error correction terms. In this case the signs are as expected, and only the Australian economy adjusts significantly to shocks in the short-run. Issuing a `modtest 1 --autocorr` after the `vecm` will produce some autocorrelation statistics. Check these to make sure that no autocorrelation remains.

In this example, having 2 lagged differences in the U.S. equation appears to be warranted. The second lag in the Australian equation is also significant at 10%. The signs on the error correction terms make sense. I would conclude that this model is a worthy candidate for further use.

One more check is worth considering. A plot of the error correction terms is shown in Figure 13.7 This plot shows that most of the disequilibrium is negative. Australia is constantly playing catch-up to the U.S. I'm not sure I believe this. You will notice that the coefficient in the cointegrating equation is -1.025. The simple least squares estimation of it was -0.98. I suspect that this parameter should be equal to -1 (these market economies are roughly comparable) and I test for it, using a `restrict` statement. The hypothesis is not rejected at 5% and the restriction is imposed and the plot recast as shown in Figure 13.8. You can see that it has the same basic shape as in Figure 13.7, but the now there are many more positive disequilibria. The regression output from



```
VECM system, lag order 3
Maximum likelihood estimates, observations 1970:4-2000:4 (T = 121)
Cointegration rank = 1
Case 3: Unrestricted constant

beta (cointegrating vectors, standard errors in parentheses)

aus       1.0000
        (0.00000)
usa      -1.0268
        (0.025994)

alpha (adjustment vectors)

aus     -0.12186
usa     0.020795

Log-likelihood = -179.93953
Determinant of covariance matrix = 0.067101624
AIC = 3.2056
BIC = 3.5291
HQC = 3.3370
```

Estimates of the cointegrating relationship.
Coefficients on error correction terms. Opposite signs implies proper adjustment to disequilibrium.
You can use these model selection rules to refine lag choices for the vecm.

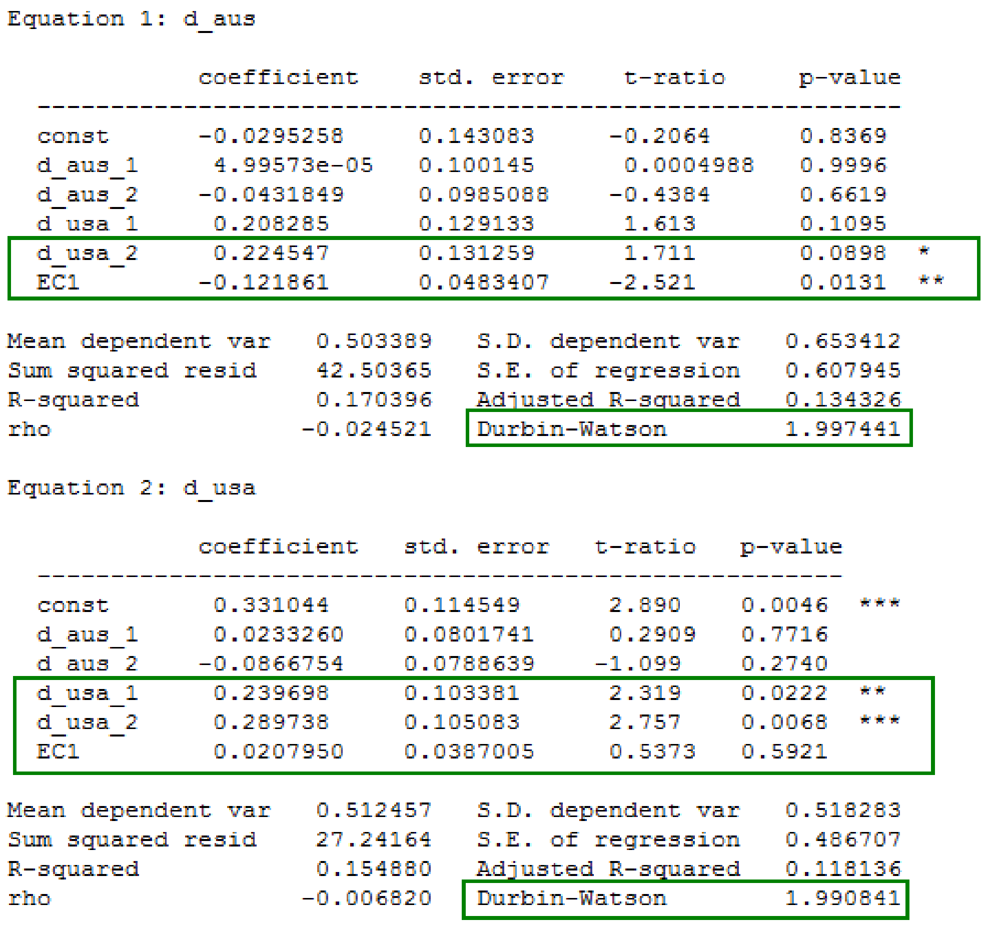
Figure 13.5: The first part of the output from the `vecm 3 1 aus usa` command.

the restricted VECM appears below: The magnitude of the adjustment parameters have become more similar in magnitude. The coefficient for Australia (-0.096929) is significant at 10% and the one for the U.S. is not.

Finally, there are some advantages of working with a script as well. **Gretl** has accessors for some of the output from `vecm`. The `$jbeta` accessor stores the parameters from the cointegrating estimations. `$vecGamma` stores the coefficients on the lagged differences of the cointegrated variables, and `$ec` stores the error correction terms. In the script, I compute the error correction terms manually using `$jbeta`. There are other accessors for the `vecm` results. See the Gretl Users Guide for details.

Restricting the VECM and accessing some results

```
vecm 3 1 aus usa
restrict --full
    b[1]+b[2]=0
end restrict

scalar a = $vecGamma
scalar b =$jbeta
series ec = aus + $jbeta[2,1]*usa
```



```
Equation 1: d_aus

             coefficient    std. error    t-ratio     p-value
  -----------------------------------------------------------
  const      -0.0295258     0.143083     -0.2064      0.8369
  d_aus_1     4.99573e-05   0.100145      0.0004988   0.9996
  d_aus_2    -0.0431849     0.0985088    -0.4384      0.6619
  d_usa_1     0.208285      0.129133      1.613       0.1095
  d_usa_2     0.224547      0.131259      1.711       0.0898   *
  EC1        -0.121861      0.0483407    -2.521       0.0131   **

Mean dependent var   0.503389   S.D. dependent var   0.653412
Sum squared resid    42.50365   S.E. of regression   0.607945
R-squared            0.170396   Adjusted R-squared   0.134326
rho                 -0.024521   Durbin-Watson        1.997441

Equation 2: d_usa

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const       0.331044     0.114549      2.890    0.0046  ***
  d_aus_1     0.0233260    0.0801741     0.2909   0.7716
  d_aus_2    -0.0866754    0.0788639    -1.099    0.2740
  d_usa_1     0.239698     0.103381      2.319    0.0222  **
  d_usa_2     0.289738     0.105083      2.757    0.0068  ***
  EC1         0.0207950    0.0387005     0.5373   0.5921

Mean dependent var   0.512457   S.D. dependent var   0.518283
Sum squared resid    27.24164   S.E. of regression   0.486707
R-squared            0.154880   Adjusted R-squared   0.118136
rho                 -0.006820   Durbin-Watson        1.990841
```

Figure 13.6: The second part of the output from the `vecm 3 1 aus usa` command.

13.2 Vector Autoregression

The vector autoregression model (VAR) is actually a little simpler to estimate than the VEC model. It is used when there is no cointegration among the variables and it is estimated using time-series that have been transformed to their stationary values.

In the example from *POE4*, we have macroeconomic data on *RPDI* and *RPCE* for the United States. The data are found in the *fred.gdt* dataset and have already been transformed into their natural logarithms. In the dataset, y is the log of real disposable income and c is log of real consumption expenditures. As in the previous example, the first step is to determine whether the variables are stationary. If they are not, then you transform them into stationary time-series and test for cointegration.

The data need to be analyzed in the same way as the *GDP* series in the VECM example. Examine the plots to determine possible trends and use the ADF tests to determine which form

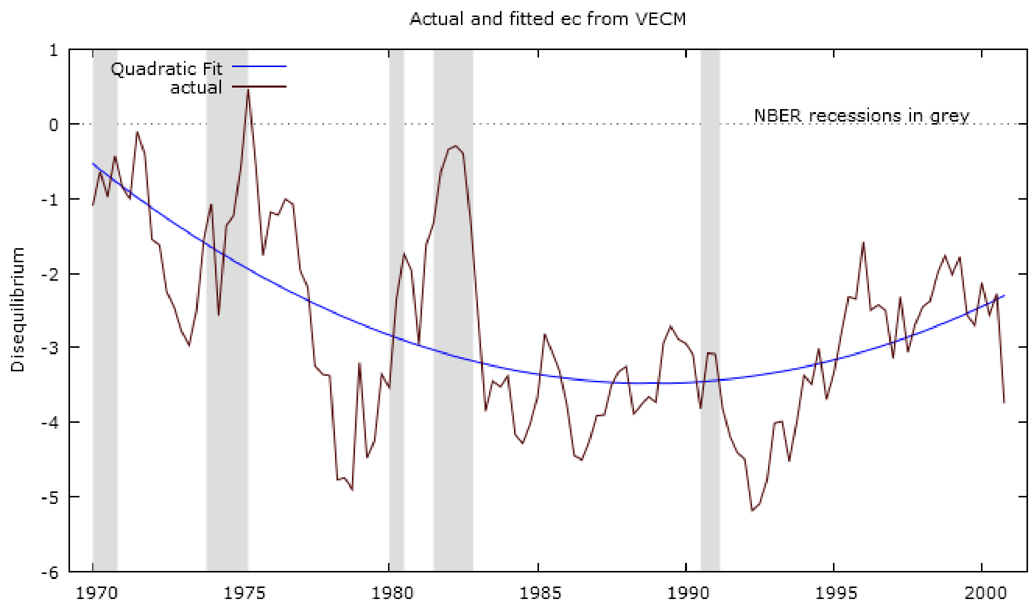


Figure 13.7: Plot of the error correction terms from the `vecm 3 1 aus usa` command.

of the data are stationary. These data are nonstationary in levels, but stationary in differences. Then, estimate the cointegrating vector and test the stationarity of its residuals. If stationary, the series are cointegrated and you estimate a VECM. If not, then a VAR treatment is sufficient.

Open the data and take a look at the time-series plots.

```
open "@gretldir\data\poe\fred.gdt"
scatters c diff(c) y diff(y)
```

The plots appear in Figure 13.10. The levels series appear to be trending together. The differences may be trending downward ever so slightly. The mean of the difference series appears to be greater than zero, suggesting that a least a constant be included in the ADF regressions. Inclusion of a trend could be tested using a t-test based on the regression output.

The other decision that needs to be made is the number of lagged differences to include in the augmented Dickey-Fuller regressions. The principle to follow is to include just enough so that the residuals of the ADF regression are not autocorrelated. The recommendation is to test down using the `--test-down` option of the `adf` command.

```
adf 12 c --ct --test-down --verbose
adf 12 y --ct --test-down --verbose
```

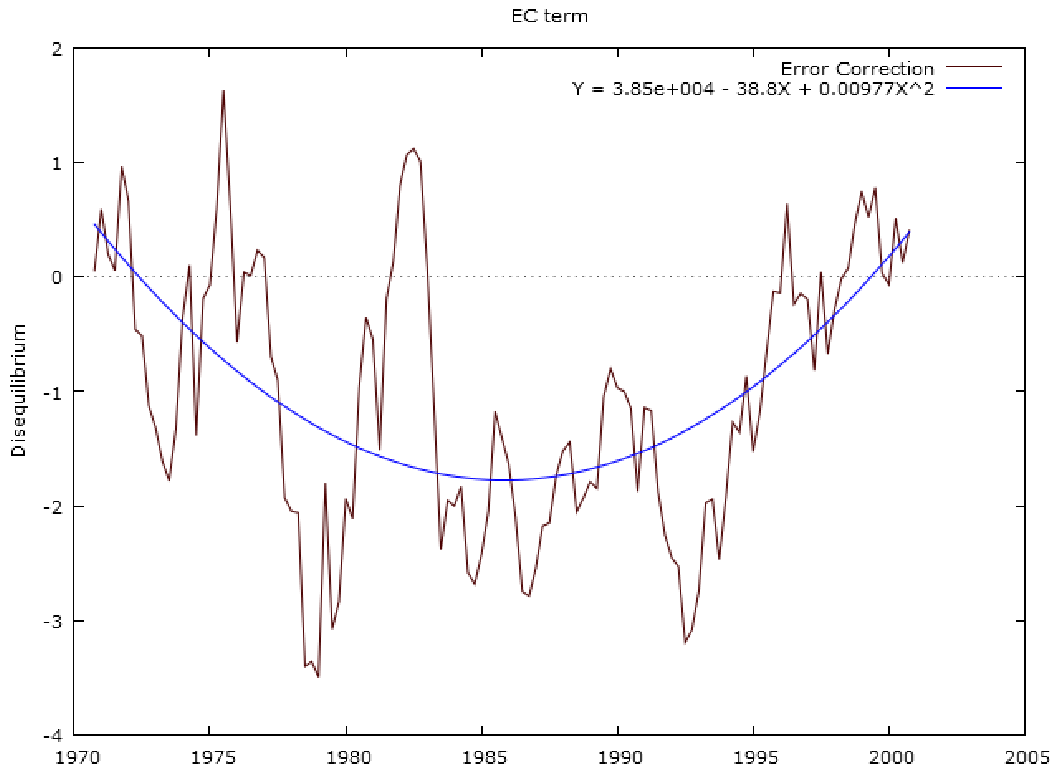


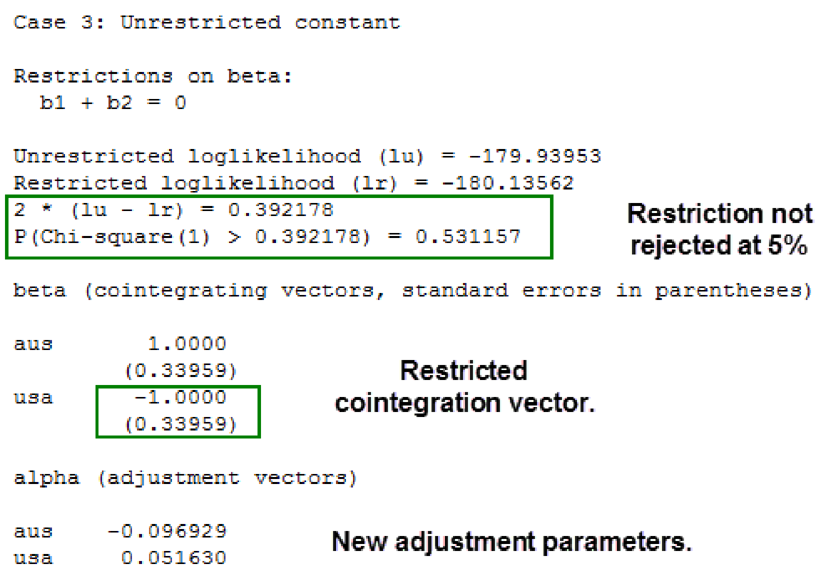
Figure 13.8: Plot of the error correction terms from the `vecm 3 1 aus usa` where the cointegrating vector is $aus = usa$.

After some experimentation, the decision was made to keep the trend in the ADF regresions. The term was significant for both series. The test-down procedure chose 3 lagged differences of c in the first model and 10 lagged differences of y in the second. In both cases, the unit root hypothesis could not be rejected at 10%. See Figures 13.11 and 13.12.

It is probably a good idea to confirm that the differences are stationary, since VAR in differences will require this.

If c and y are cointegrated then you would estimate a VECM. The Engle-Granger tests reveals that they are not.

```
Augmented Dickey-Fuller test for uhat
including 5 lags of (1-L)uhat (max was 12)
sample size 194
unit-root null hypothesis: a = 1

   model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: -0.008
   lagged differences: F(5, 188) = 5.028 [0.0002]
   estimated value of (a - 1): -0.0798819
```



```
Case 3: Unrestricted constant

Restrictions on beta:
  b1 + b2 = 0

Unrestricted loglikelihood (lu) = -179.93953
Restricted loglikelihood (lr) = -180.13562
2 * (lu - lr) = 0.392178
P(Chi-square(1) > 0.392178) = 0.531157

beta (cointegrating vectors, standard errors in parentheses)

aus       1.0000
        (0.33959)
usa      -1.0000
        (0.33959)

alpha (adjustment vectors)

aus    -0.096929
usa     0.051630
```

Restriction not rejected at 5%
Restricted cointegration vector.
New adjustment parameters.

Figure 13.9: Output from the restricted VECM model. The cointegrating relationship is A=U.

```
   test statistic: tau_c(2) = -2.39489
   asymptotic p-value 0.327

There is evidence for a cointegrating relationship if:
(a) The unit-root hypothesis is not rejected for the individual variables.
(b) The unit-root hypothesis is rejected for the residuals (uhat) from the
    cointegrating regression.
```

The p-value on the test statistic is 0.327. We cannot reject the unit root hypothesis for the residuals and therefore the series are not cointegrated. We are safe to estimate the VAR in differences.

The basic syntax for the `var` command appears below

```
var

Arguments:    order ylist [ ; xlist ]
Options:      --nc (do not include a constant)
              --trend (include a linear trend)
              --seasonals (include seasonal dummy variables)
              --robust (robust standard errors)
              --robust-hac (HAC standard errors)
              --impulse-responses (print impulse responses)
              --variance-decomp (print variance decompositions)
              --lagselect (show information criteria for lag selection)
```

You specify the lag order, the series to place in the VAR, and any options you want. You can choose HAC standard errors and ways to model deterministic trends in the model. Estimating the

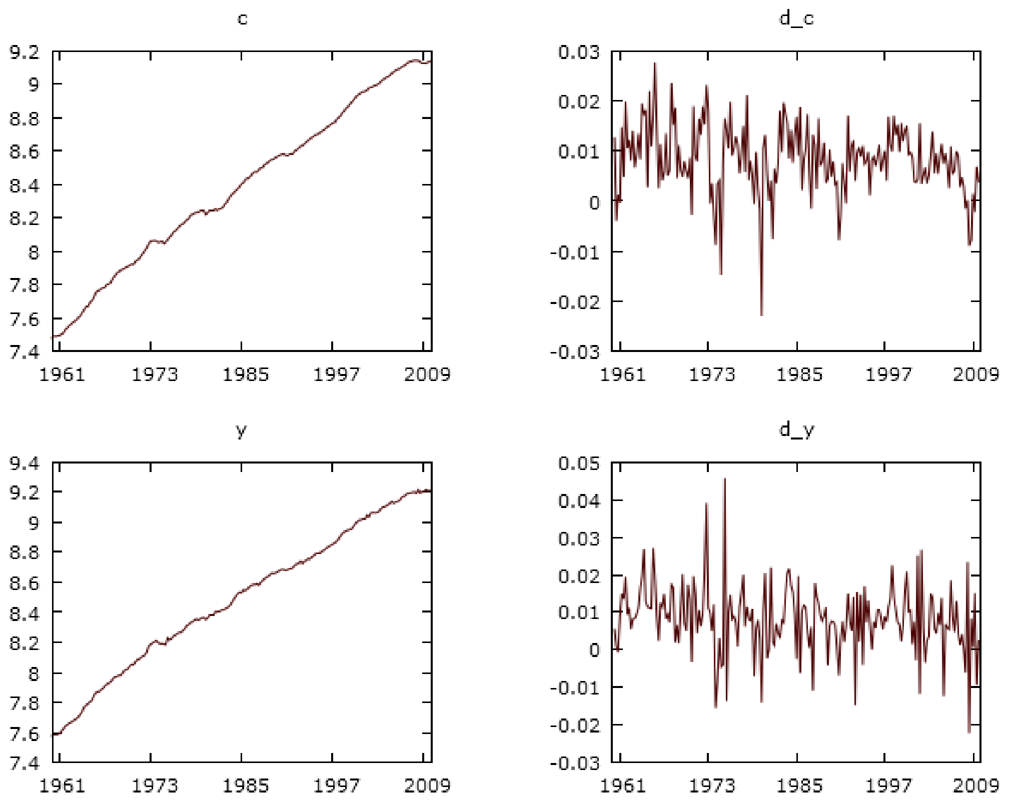


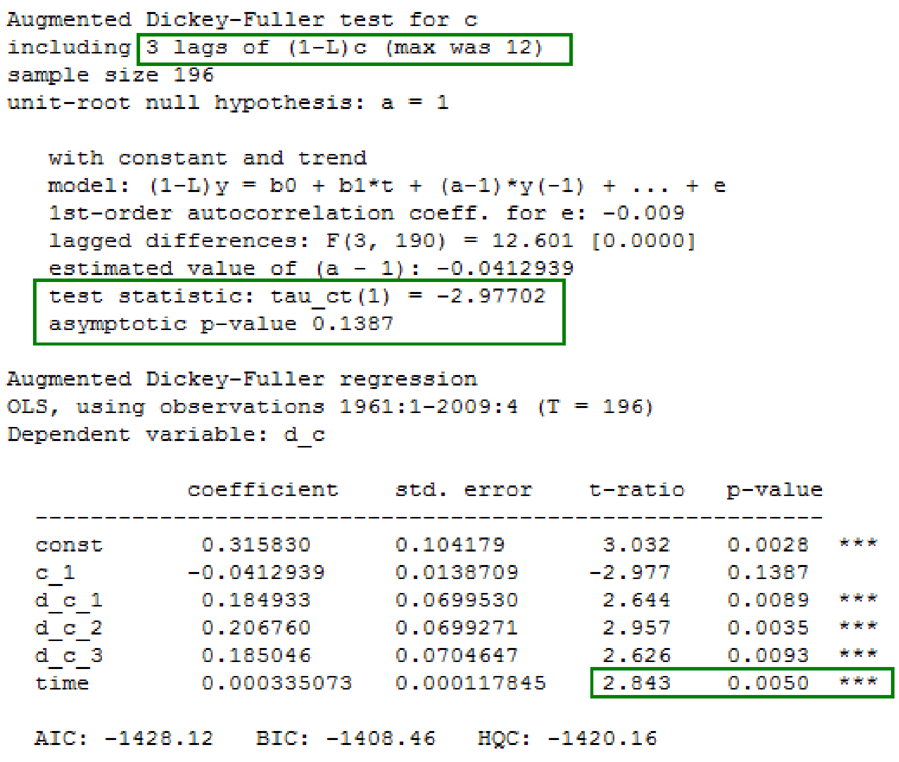
Figure 13.10: Natural logs of consumption and income and their differences.

VAR with the *–lagselect* option is useful in deciding how many lags of the two variables to add to the model.

```
var 12 diff(c) diff(y) --lagselect
```

We've chosen that option here with the first few lines of the result:

```
VAR system, maximum lag order 12

The asterisks below indicate the best (that is, minimized) values
of the respective information criteria, AIC = Akaike criterion,
BIC = Schwarz Bayesian criterion and HQC = Hannan-Quinn criterion.

lags        loglik    p(LR)       AIC          BIC          HQC

   1    1319.59415            -14.049135   -13.945463*  -14.007127*
   2    1323.61045  0.09039   -14.049310   -13.876523   -13.979296
   3    1329.48171  0.01937   -14.069323*  -13.827422   -13.971305
```



```
Augmented Dickey-Fuller test for c
including 3 lags of (1-L)c (max was 12)
sample size 196
unit-root null hypothesis: a = 1

   with constant and trend
   model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: -0.009
   lagged differences: F(3, 190) = 12.601 [0.0000]
   estimated value of (a - 1): -0.0412939
   test statistic: tau_ct(1) = -2.97702
   asymptotic p-value 0.1387

Augmented Dickey-Fuller regression
OLS, using observations 1961:1-2009:4 (T = 196)
Dependent variable: d_c

             coefficient    std. error    t-ratio   p-value
  ---------------------------------------------------------
  const        0.315830     0.104179        3.032    0.0028   ***
  c_1         -0.0412939    0.0138709      -2.977    0.1387
  d_c_1        0.184933     0.0699530       2.644    0.0089   ***
  d_c_2        0.206760     0.0699271       2.957    0.0035   ***
  d_c_3        0.185046     0.0704647       2.626    0.0093   ***
  time         0.000335073  0.000117845     2.843    0.0050   ***

  AIC: -1428.12   BIC: -1408.46   HQC: -1420.16
```

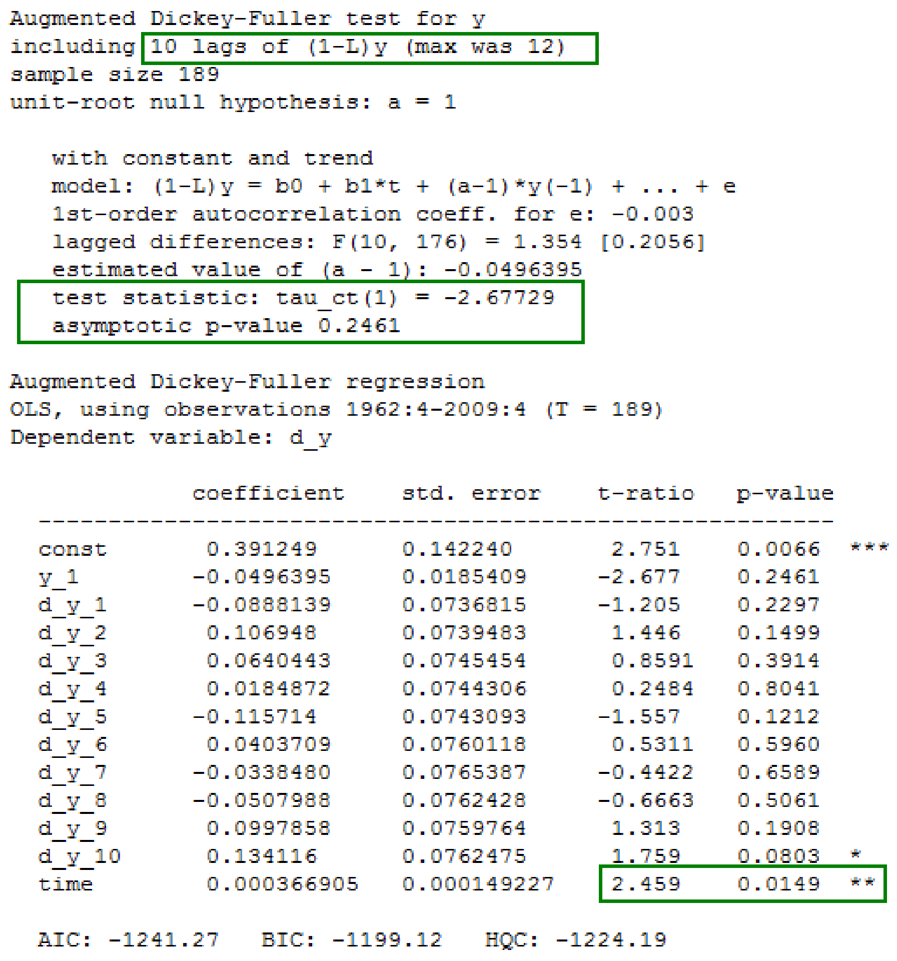
Figure 13.11: ADF tests of ln(RPCE)

```
   4    1333.38145   0.09921   -14.068251    -13.757235    -13.942227
```

The *BIC* (*SC*) and *HQC* pick the same number of lags, 1. That is what we've estimated so we are satisfied. You can also issue a model test command after the VAR to determine if there is any remaining autocorrelation in the residuals. If there is, you probably need to add additional lags to the VAR. When used here, the Ljung-Box Q statistics for both equations have p-values above 0.10 and the null hypothesis of no autocorrelation is not rejected.

The model output is found in Table 13.1

You can also get **gretl** to generate the VAR's lag selection command through the dialogs. Select `Model>Time series>VAR lag selection` from the pull-down menu. This reveals the VAR lag selection dialog box. You can choose the maximum lag to consider, the variables to include in the model, and whether the model should contain constant, trend, or seasonal dummies.



```
Augmented Dickey-Fuller test for y
including 10 lags of (1-L)y (max was 12)
sample size 189
unit-root null hypothesis: a = 1

   with constant and trend
   model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
   1st-order autocorrelation coeff. for e: -0.003
   lagged differences: F(10, 176) = 1.354 [0.2056]
   estimated value of (a - 1): -0.0496395
   test statistic: tau_ct(1) = -2.67729
   asymptotic p-value 0.2461

Augmented Dickey-Fuller regression
OLS, using observations 1962:4-2009:4 (T = 189)
Dependent variable: d_y

             coefficient    std. error    t-ratio   p-value
  ---------------------------------------------------------
  const       0.391249      0.142240       2.751    0.0066   ***
  y_1        -0.0496395     0.0185409     -2.677    0.2461
  d_y_1      -0.0888139     0.0736815     -1.205    0.2297
  d_y_2       0.106948      0.0739483      1.446    0.1499
  d_y_3       0.0640443     0.0745454      0.8591   0.3914
  d_y_4       0.0184872     0.0744306      0.2484   0.8041
  d_y_5      -0.115714      0.0743093     -1.557    0.1212
  d_y_6       0.0403709     0.0760118      0.5311   0.5960
  d_y_7      -0.0338480     0.0765387     -0.4422   0.6589
  d_y_8      -0.0507988     0.0762428     -0.6663   0.5061
  d_y_9       0.0997858     0.0759764      1.313    0.1908
  d_y_10      0.134116      0.0762475      1.759    0.0803   *
  time        0.000366905   0.000149227    2.459    0.0149   **

  AIC: -1241.27   BIC: -1199.12   HQC: -1224.19
```

Figure 13.12: ADF tests of ln(RPDI)

13.3 Impulse Response Functions and Variance Decompositions

Impulse response functions show the effects of shocks on the adjustment path of the variables. Forecast error variance decompositions measure the contribution of each type of shock to the forecast error variance. Both computations are useful in assessing how shocks to economic variables reverberate through a system.

Impulse response functions (IRFs) and forecast error variance decompositions (FEVD) can be produced after using the `var` or `vecm` commands. The results can be presented in a table or a graph.

Obtaining the impulse responses after estimating a VAR is easy in **gretl**. The first step is to estimate the VAR. From the main **gretl** window choose `Model>Time series>Vector Autoregression`. This brings up the dialog, shown in Figure 13.13. Set the lag order to 1, and add the differenced variables to the box labeled `Endogenous Variables`. Make sure the 'Include a constant' box is

VAR system, lag order 1
OLS estimates, observations 1960:3–2009:4 ($T = 198$)

Equation 1: d_c
Heteroskedasticity-robust standard errors, variant HC3

	Coefficient	Std. Error	t-ratio	p-value
const	0.00527761	0.000952508	5.5408	0.0000
d_c_1	0.215607	0.0903028	2.3876	0.0179
d_y_1	0.149380	0.0595427	2.5088	0.0129

Mean dependent var	0.008308	S.D. dependent var	0.006976
Sum squared resid	0.008431	S.E. of regression	0.006575
R^2	0.120487	Adjusted R^2	0.111466
$F(2, 195)$	10.39596	P-value(F)	0.000051
$\hat{\rho}$	−0.052639	Durbin–Watson	2.085697

Equation 2: d_y
Heteroskedasticity-robust standard errors, variant HC3

	Coefficient	Std. Error	t-ratio	p-value
const	0.00603667	0.00110476	5.4642	0.0000
d_c_1	0.475428	0.105260	4.5167	0.0000
d_y_1	−0.217168	0.0977454	−2.2218	0.0274

Mean dependent var	0.008219	S.D. dependent var	0.009038
Sum squared resid	0.014293	S.E. of regression	0.008562
R^2	0.111815	Adjusted R^2	0.102706
$F(2, 195)$	10.22543	P-value(F)	0.000060
$\hat{\rho}$	−0.003022	Durbin–Watson	1.993480

Table 13.1: Results from the VAR

checked and click **OK**. The results are shown in Table 13.1.

You can generate impulse responses by selecting `Analysis>Impulse responses` from the results window. An impulse response dialog appears that allows you to specify the forecast horizon and to change the ordering of the variables. Using 12 periods with `d_c` ordered first produces the results shown in Figure 13.2.

These can be graphed for easier interpretation from the results window by selecting `Graphs>Impulse responses (combined)` from the pull-down menu. This brings up a dialog that allows you to choose how the graph will be constructed. The dialog is shown in Figure 13.14.

This yields the graph shown in Figure 13.15. The forecast error variance decompositions (FEVD) are obtained similarly. Select `Analysis>Forecast variance decomposition` from the

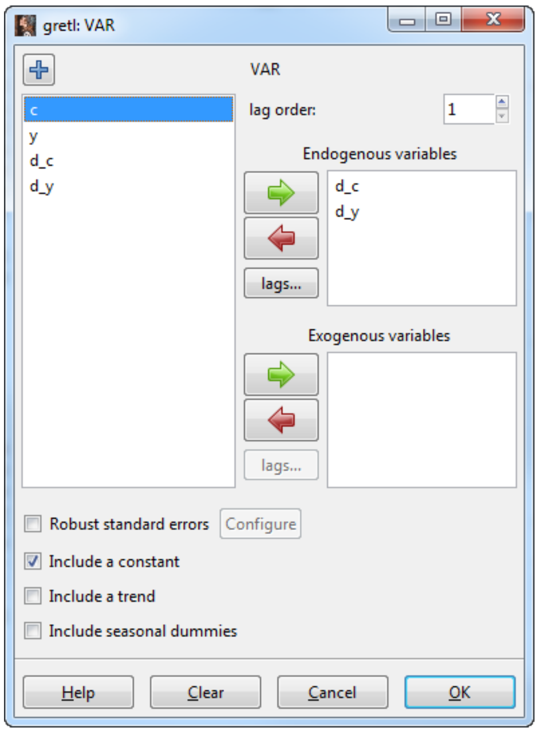
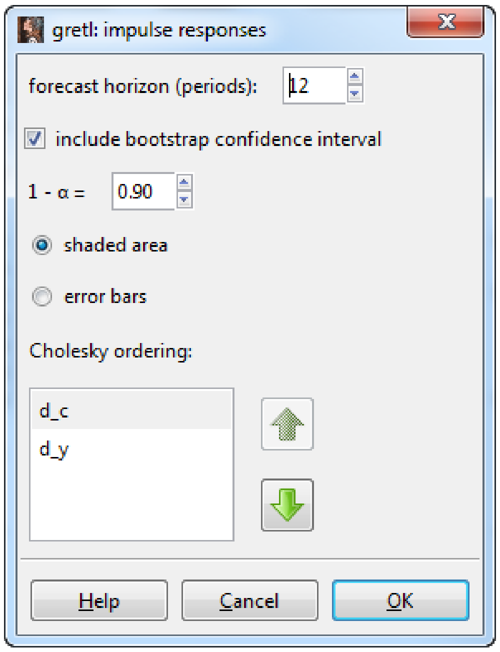


Figure 13.13: From the main **gretl** window, choose `Model>Time series>Vector Autogregression` to bring up the VAR dialog box.

vector autoregression model window to obtain the result shown in Table 13.3.

To generate the IRFs and the FEVDs using a script, simply employ the options `--impulse-responses` and `--variance-decomp`. These can be used with the `var` command as done here or the `vecm` command.

```
var 1 diff(c) diff(y) --impulse-responses --variance-decomp
```


gretl: impulse responses

forecast horizon (periods): 12

[✓] include bootstrap confidence interval

1 - α = 0.90

(●) shaded area

( ) error bars

Cholesky ordering:

d_c
d_y

Figure 13.14: Select `Graphs>Impulse responses (combined)` from the VAR results window brings up this dialog box.

13.4 Script

```
open "@gretldir\data\poe\gdp.gdt"
set echo off
setobs 4 1970:1 --time-series
# plot multiple time-series
scatters usa diff(usa) aus diff(aus)

# ADF tests with test down
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
adf mlag usa --ctt --test-down
adf mlag aus --ctt --test-down

adf mlag diff(usa) --ct --test-down
adf mlag diff(aus) --ct --test-down

# manually testing down based on LM tests
# USA
genr time
square time
diff usa aus
loop i=1..12
    ols d_usa(0 to -i) usa(-1)  const time sq_time --quiet
    printf "ADF lag order = %d\n",i
    modtest 1 --autocorr --quiet
```

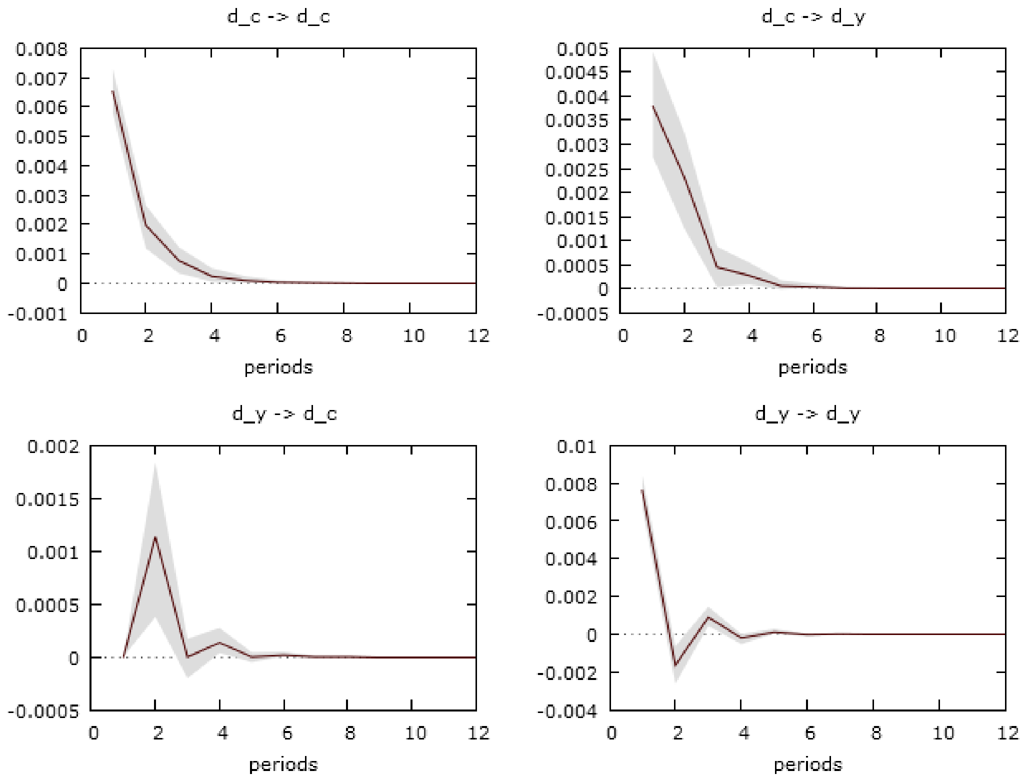


Figure 13.15: U.S. ln($RDPI$) and ln($RPCE$) impulse responses

```
end loop

# Australia
loop i=0..12
    if i = 0
        ols d_aus aus(-1)  const time sq_time --quiet
    else
        ols d_aus(0 to -i) aus(-1)  const time sq_time --quiet
    endif
    printf "ADF lag order = %d\n",i
    modtest 1 --autocorr --quiet
end loop

# Section 13.2 in POE4
ols aus usa
series uhat = $uhat
ols diff(uhat) uhat(-1)
ols diff(aus) const uhat(-1)
ols diff(usa) const uhat(-1)
modtest 1 --autocorr

# Engle-Granger test
coint 8 aus usa --test-down --nc
```

```

# restricted VECM
vecm 3 1 aus usa
restrict --full
    b[1]+b[2]=0
end restrict

# collecting error correction terms from restricted model
matrix a = $vecGamma
matrix b =$jbeta
series ec = aus + $jbeta[2,1]*usa
modtest 1 --autocorr

# VAR estimation
open "@gretldir\data\poe\fred.gdt"
scatters c diff(c) y diff(y)

adf 12 c --ct --test-down --verbose
adf 12 y --ct --test-down --verbose

adf 12 diff(c) --ct --test-down --verbose
adf 12 diff(y) --ct --test-down --verbose

var 12 diff(c) diff(y) --lagselect
var 1 diff(c) diff(y) --robust-hac
modtest 1 --autocorr

var 1 diff(c) diff(y) --impulse-responses --variance-decomp
```

Responses to a one-standard error shock in d_c

period	d_c	d_y
1	0.00652541	0.00378594
2	0.00197247	0.00228018
3	0.000765890	0.000442583
4	0.000231244	0.000268010
5	8.98931e–005	5.17366e–005
6	2.71100e–005	3.15021e–005
7	1.05509e–005	6.04757e–006
8	3.17823e–006	3.70284e–006
9	1.23838e–006	7.06878e–007
10	3.72596e–007	4.35247e–007
11	1.45351e–007	8.26206e–008
12	4.36806e–008	5.11615e–008

Responses to a one-standard error shock in d_y

period	d_c	d_y
1	0.000000	0.00760630
2	0.00113623	−0.00165185
3	−1.77382e–006	0.000898922
4	0.000133898	−0.000196060
5	−4.18065e–007	0.000106237
6	1.57795e–005	−2.32700e–005
7	−7.38999e–008	1.25555e–005
8	1.85961e–006	−2.76179e–006
9	−1.16116e–008	1.48388e–006
10	2.19159e–007	−3.27772e–007
11	−1.71048e–009	1.75376e–007
12	2.58288e–008	−3.88992e–008

Table 13.2: Impulse response functions (IRF)

Decomposition of variance for d_c

period	std. error	d_c	d_y
1	0.00652541	100.0000	0.0000
2	0.00691105	97.2970	2.7030
3	0.00695336	97.3298	2.6702
4	0.0069585	97.2967	2.7033
5	0.00695908	97.2972	2.7028
6	0.00695915	97.2967	2.7033
7	0.00695916	97.2967	2.7033
8	0.00695916	97.2967	2.7033
9	0.00695916	97.2967	2.7033
10	0.00695916	97.2967	2.7033
11	0.00695916	97.2967	2.7033
12	0.00695916	97.2967	2.7033

Decomposition of variance for d_y

period	std. error	d_c	d_y
1	0.00849642	19.8552	80.1448
2	0.00895081	24.3800	75.6200
3	0.00900671	24.3198	75.6802
4	0.00901283	24.3752	75.6248
5	0.00901361	24.3743	75.6257
6	0.00901369	24.3750	75.6250
7	0.0090137	24.3750	75.6250
8	0.00901371	24.3750	75.6250
9	0.00901371	24.3750	75.6250
10	0.00901371	24.3750	75.6250
11	0.00901371	24.3750	75.6250
12	0.00901371	24.3750	75.6250

Table 13.3: Forecast Error Variance Decompositions (FEVD)

Chapter 14

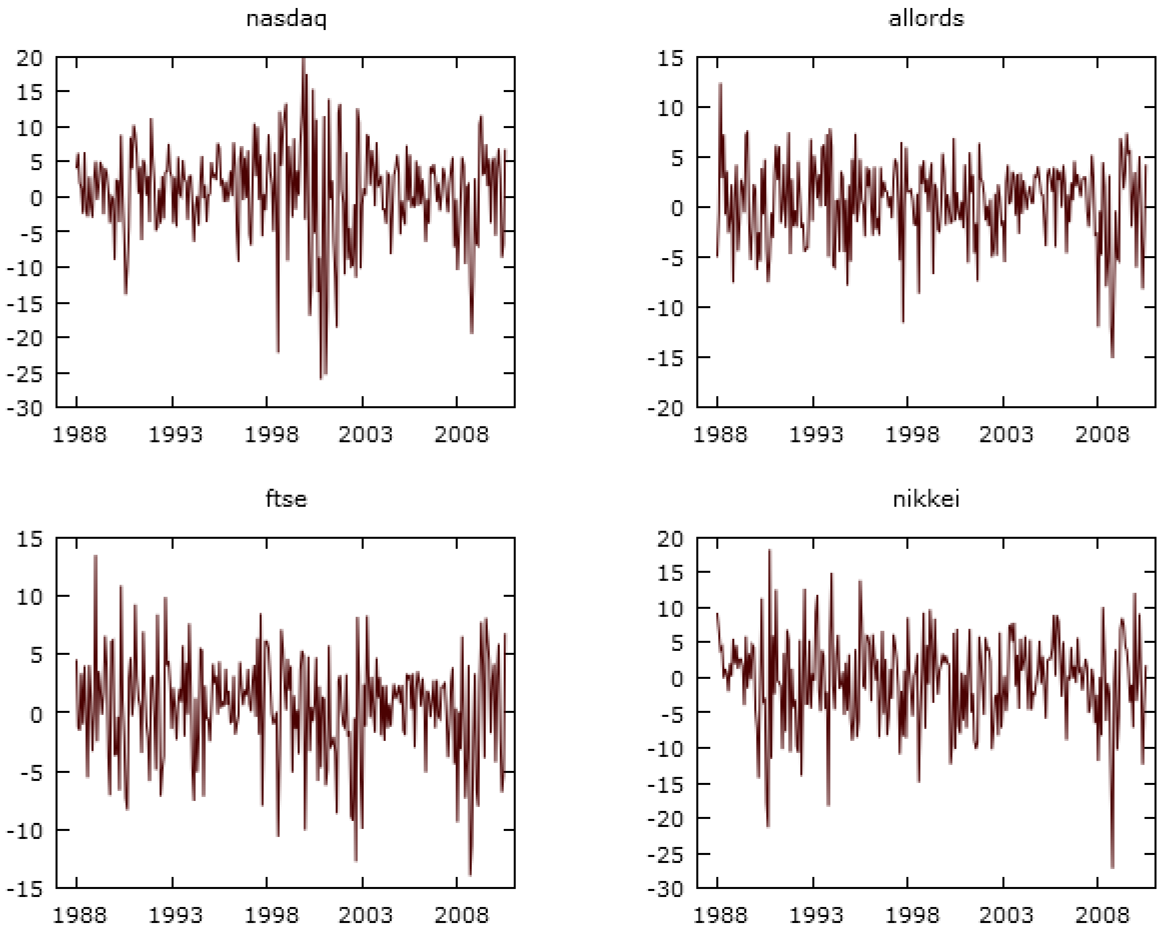
Time-Varying Volatility and ARCH Models: Introduction to Financial Econometrics

In this chapter we'll estimate several models in which the variance of the dependent variable changes over time. These are broadly referred to as ARCH (autoregressive conditional heteroskedasticity) models and there are many variations upon the theme.

The first thing to do is illustrate the problem graphically using data on stock returns. The data are stored in the **gretl** dataset *returns.gdt*. The data contain four monthly stock price indices: U.S. Nasdaq (nasdaq), the Australian All Ordinaries (allords), the Japanese Nikkei (nikkei) and the U.K. FTSE (ftse). The data are recorded monthly beginning in 1988:01 and ending in 2009:07. Notice that with monthly data, the suffix is two digits, that is 1988:01 is January (01) in the year 1988.

Simple scatter plots appear below. They can be generated using the GUI as described on page 278, or using the `scatters` command.

```
open "@gretldir\data\poe\returns.gdt"
scatters nasdaq allords ftse nikkei
```



14.1 ARCH and GARCH

The basic ARCH(1) model can be expressed as:

$$y_t = \beta + e_t \tag{14.1}$$
$$e_t|I_{t-1} \sim N(0, h_t) \tag{14.2}$$
$$h_t = \alpha_0 + \alpha_1 e_{t-1}^2 \tag{14.3}$$
$$\alpha_0 > 0,\ 0 \le \alpha_1 < 1$$

The first equation describes the behavior of the mean of your time-series. In this case, equation (14.1) indicates that we expect the time-series to vary randomly about its mean, β. If the mean of your time-series drifts over time or is explained by other variables, you'd add them to this equation just as you would a regular regression model. The second equation indicates that the error of the regression, e_t, are normally distributed and heteroskedastic. The variance of the current period's error depends on information that is revealed in the preceding period, i.e., I_{t-1}. The variance of e_t is given the symbol h_t. The final equation describes how the variance behaves. Notice that h_t

depends on the error in the preceding time period. The parameters in this equation have to be positive to ensure that the variance, h_t, is positive. Notice also that α cannot be greater than one; if it were, the variance would be unstable.

The ARCH(1) model can be extended to include more lags of the errors, e_{t-q}. In this case, q refers to the order of the ARCH model. For example, ARCH(2) replaces (14.3) with $h_t = \alpha_0 + \alpha_1 e^2_{t-1} + \alpha_2 e^2_{t-2}$. When estimating regression models that have ARCH errors in **gretl**, you'll have to specify this order.

ARCH is treated as a special case of a more general model in **gretl** called GARCH. GARCH stands for generalized autoregressive conditional heteroskedasticity and it adds lagged values of the variance itself, h_{t-p}, to (14.3). The GARCH(1,1) model is:

$$\begin{aligned} y_t &= \beta + e_t \\ e_t|I_{t-1} &\sim N(0, h_t) \\ h_t &= \delta + \alpha_1 e^2_{t-1} + \beta_1 h_{t-1} \end{aligned} \tag{14.4}$$

The difference between ARCH (14.3) and its generalization (14.4) is a term $\beta_1 h_{t-1}$, a function of the lagged variance. In higher order GARCH(p, q) model's, q refers to the number of lags of e_t and p refers to the number of lags of h_t to include in the model of the regression's variance.

To open the dialog for estimating ARCH and GARCH in **gretl** choose `Model>Time series>GARCH` from the main **gretl** window.[1] This reveals the dialog box where you specify the model (Figure 14.1). To estimate the ARCH(1) model, you'll place the time-series `r` into the dependent variable box and set `q=1` and `p=0`. This yields the results:

Model 1: GARCH, using observations 1–500
Dependent variable: r
Standard errors based on Hessian

	Coefficient	Std. Error	z	p-value
const	1.06394	0.0399241	26.6491	0.0000
α_0	0.642139	0.0648195	9.9066	0.0000
α_1	0.569347	0.0913142	6.2350	0.0000

Mean dependent var	1.078294	S.D. dependent var	1.185025
Log-likelihood	−740.7932	Akaike criterion	1489.586
Schwarz criterion	1506.445	Hannan–Quinn	1496.202

Unconditional error variance = 1.49108

[1]In a later version of **gretl** , an ARCH option has been added. You can use this as well, but the answer you get will be slightly different due to differences in the method used to estimate the model.

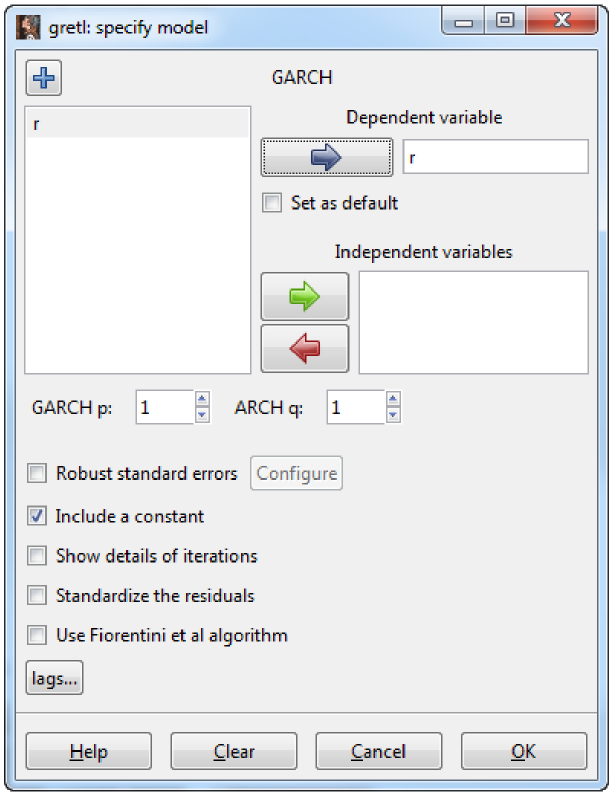
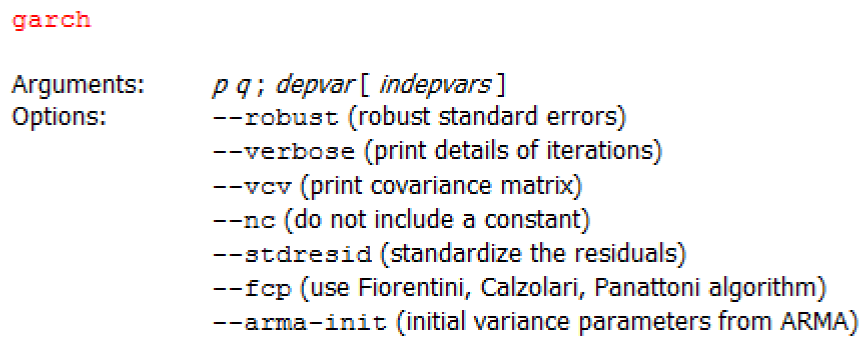


Figure 14.1: Estimating ARCH using the dialog box in **gretl** .

You will notice that the coefficient estimates and standard errors for the ARCH(1) and GARCH(1, 1) models are quite close to those in chapter 14 of your textbook. To obtain these, you will have to change the default variance-covariance computation using `set garch_vcv op` before running the script. Although this gets you close the the results in *POE4*, using the `garch_vcv op` is not usually recommended; just use the **gretl** default, `set garch_vcv unset`.

The standard errors and t-ratios often vary a bit, depending on which software and numerical techniques are used. This is the nature of maximum likelihood estimation of the model's parameters. With maximum likelihood, the model's parameters are estimated using numerical optimization techniques. All of the techniques usually get you to the same parameter estimates, i.e., those that maximize the likelihood function; but, they do so in different ways. Each numerical algorithm arrives at the solution iteratively based on reasonable starting values and the method used to measure the curvature of the likelihood function at each round of estimates. Once the algorithm finds the maximum of the function, the curvature measure is reused as an estimate of the variance covariance matrix. Since curvature can be measured in slightly different ways, the routine will produce slightly different estimates of standard errors.

Gretl gives you a way to choose which method you like use for estimating the variance-covariance matrix. And, as expected, this choice will produce different standard errors and t-ratios. The `set garch_vcv` command allows you to choose among five alternatives: `unset`–which restores the default, **hessian**, **im** (information matrix) , **op** (outer product matrix), **qml** (QML estimator), or **bw** (Bollerslev-Wooldridge). If unset is given the default is restored, which in this case is the Hessian; if the "robust" option is given for the garch command, QML is used.



```
garch

Arguments:     p q ; depvar [ indepvars ]
Options:       --robust (robust standard errors)
               --verbose (print details of iterations)
               --vcv (print covariance matrix)
               --nc (do not include a constant)
               --stdresid (standardize the residuals)
               --fcp (use Fiorentini, Calzolari, Panattoni algorithm)
               --arma-init (initial variance parameters from ARMA)
```

The series are characterized by random, rapid changes and are said to be volatile. The volatility seems to change over time as well. For instance the U.S. stock returns index (NASDAQ) experiences a relatively sedate period from 1992 to 1996. Then, stock returns become much more volatile until early 2004. Volatility increases again at the end of the sample. The other series exhibit similar periods of relative calm followed by increased volatility.

A histogram graphs of the empirical distribution of a variable. In **gretl** the `freq` command generates a histogram. A curve from a normal distribution is overlaid using the normal option and the Doornik-Hansen test for normality is performed. A histogram for the ALLORDS series appears below in Figure 14.1.

The series is leptokurtic. That means it has many observations around the average and a relatively large number of observations that are far from average; the center of the histogram has a high peak and the tails are relatively heavy compared to the normal. The normality test has a p-value of $0.0007 < 0.05$ and is significant at the 5% level.

14.2 Testing for ARCH

Testing for the presence of ARCH in the errors of your model is straightforward. In fact, there are at least two ways to proceed. The first is to estimate the regression portion of your model using least squares. Then choose the `Tests>ARCH` from the model's pull-down menu. This is illustrated in Figure 14.3 below.

This brings up the box where you tell **gretl** what order of ARCH(q) you want as your alternative hypothesis. In the example, $q = 1$ which leads to the result obtained in the text. The output is shown below in Figure 14.5. **Gretl** produces the LM statistic discussed in your text; the relevant part is highlighted in red.

The other way to conduct this test is manually. The first step is to estimate the regression

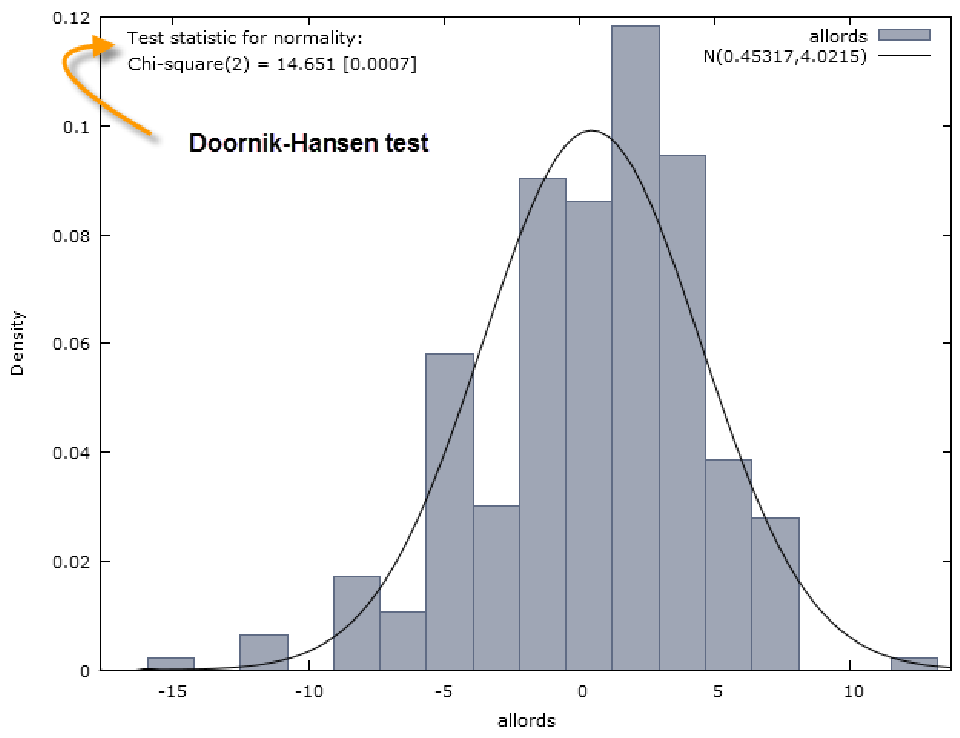


Figure 14.2: A histogram of the ALLORDS series is plotted using the normal option.

(14.1) using **gretl** . Save the squared residuals and then regress these on their lagged value. Take TR^2 from this regression as your test statistic. The script for this appears below:

```
open "c:\Program Files\gretl\data\poe\BYD.gdt"

ols r const
series ehat = $uhat
series ehat2 = ehat*ehat
ols ehat2 const ehat2(-1)
scalar tr2 = $trsq
```

The first line estimates the regression

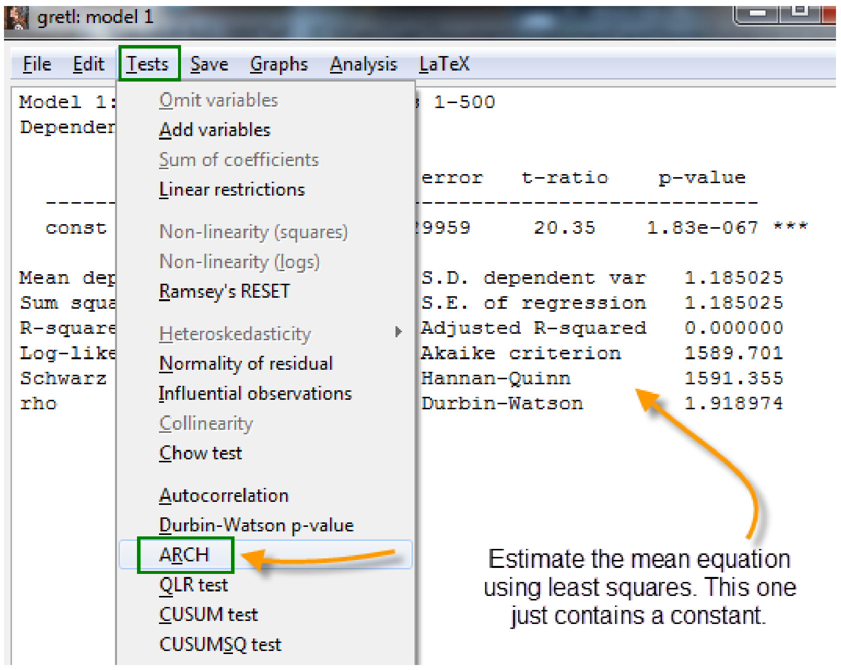
$$r_t = \beta + e_t \tag{14.5}$$

The residuals are saved in `ehat` and then squared as `ehat2`. The next line estimates the regression

$$\hat{e}_t = \alpha_1 + \alpha_2 \hat{e}_{t-1} + u_t \tag{14.6}$$

The notation `ehat2(-1)` takes the variable `ehat2` and offsets it in the dataset by the amount in parentheses. In this case, `ehat2(-1)` puts a minus one period lag of `ehat2` into your regression. The final line computes TR^2 from the regression.

Once you've estimated your ARCH or GARCH model, you can graph the behavior of the variance as done in the textbook. After estimating ARCH or GARCH, you can save the predicted vari-



```
Model 1:                  1-500
Depender

                   error    t-ratio    p-value
  ------             ----------------------------
  const            9959     20.35     1.83e-067 ***

Mean dep           S.D. dependent var   1.185025
Sum squa           S.E. of regression   1.185025
R-square           Adjusted R-squared   0.000000
Log-like           Akaike criterion     1589.701
Schwarz            Hannan-Quinn         1591.355
rho                Durbin-Watson        1.918974
```

Figure 14.3: Estimate the model using least squares. Then choose `Tests>ARCH` from the model's pull-down menu.

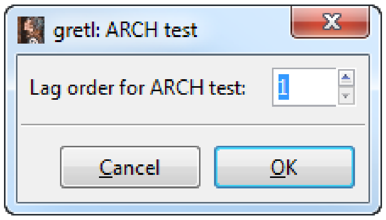


Figure 14.4: Testing for ARCH dialog box

ances using the command `series ht = $h`. Then plot them using `gnuplot ht --time-series`. The result is shown in Figure 14.2. A prettier plot can be obtained using the pull-down menus and editing the plot yourself using **gretl**'s Edit dialog box. To modify the graph, right click on the graph and choose `Edit`. From here you can add labels, change the colors or line style, and add titles. That's what I have done to produce the result in Figure 14.2.

14.3 Threshold ARCH

Threshold ARCH (TARCH) can also be estimated in **gretl**, though it requires a little programming; there aren't any pull-down menus for this estimator. Instead, we'll introduce **gretl**'s powerful `mle` command that allows user defined (log) likelihood functions to be maximized.

The threshold ARCH model replaces the variance equation (14.3) with

$$h_t = \delta + \alpha_1 e_{t-1}^2 + \gamma d_{t-1} e_{t-1}^2 + \beta_1 h_{t-1} \tag{14.7}$$

$$d_t = \begin{cases} 1 & if\ e_t < 0 \\ 0 & otherwise \end{cases} \tag{14.8}$$

The model's parameters are estimated by finding the values that maximize its likelihood. Maximum likelihood estimators are discussed in appendix C of Hill et al. (2011).

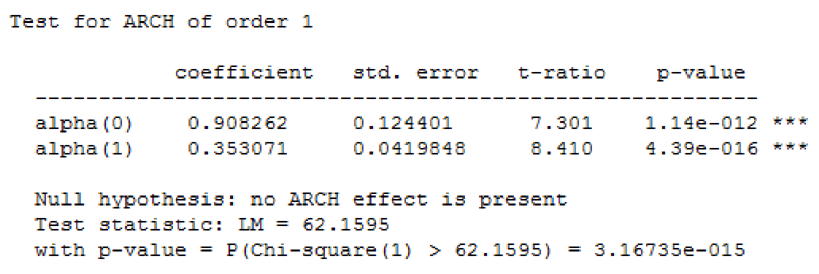
Gretl provides a fairly easy way to estimate via maximum likelihood that can be used for a wide range of estimation problems (see chapter 16 for other examples). To use **gretl**'s `mle` command, you must specify the log-likelihood function that is to be maximized. Any parameters contained in the function must be given reasonable starting values for the routine to work properly. Parameters can be declared and given starting values (using the scalar command).

Numerical optimization routines use the partial derivatives of the objective function to iteratively find the minimum or maximum of the function. If you want, you can specify the analytical derivatives of the log-likelihood function with respect to each of the parameters in **gretl**; if analytical derivatives are not supplied, **gretl** tries to compute a numerical approximation. The actual results you obtain will depend on many things, including whether analytical derivatives are used and the starting values.

For the threshold GARCH model, open a new script file and type in the program that appears in Figure 14.7.

Lines 3-7 of the script give starting values for the model's parameters. This is essential and picking good starting values increases the chances of success. You want to start the numerical optimization at a feasible point. In other words, you cannot start the model with a negative variance.

The second part of the script starting on line 9 contains the the algebraic expression of the log-likelihood function. Line 9 `ll = -0.5*(log(h) + (e^2)/h)` is what is called the *kernel* of the normal probability density function. Recall that the errors of the ARCH model are assumed to be normally distributed and this is reflected in the kernel.



```
Test for ARCH of order 1

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  alpha(0)     0.908262     0.124401      7.301    1.14e-012 ***
  alpha(1)     0.353071     0.0419848     8.410    4.39e-016 ***

  Null hypothesis: no ARCH effect is present
  Test statistic: LM = 62.1595
  with p-value = P(Chi-square(1) > 62.1595) = 3.16735e-015
```

Figure 14.5: ARCH test results

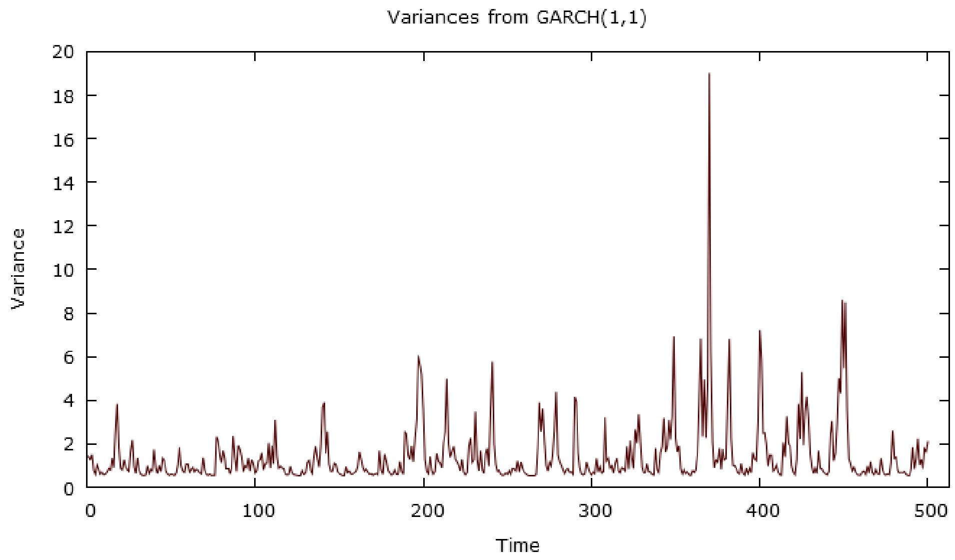


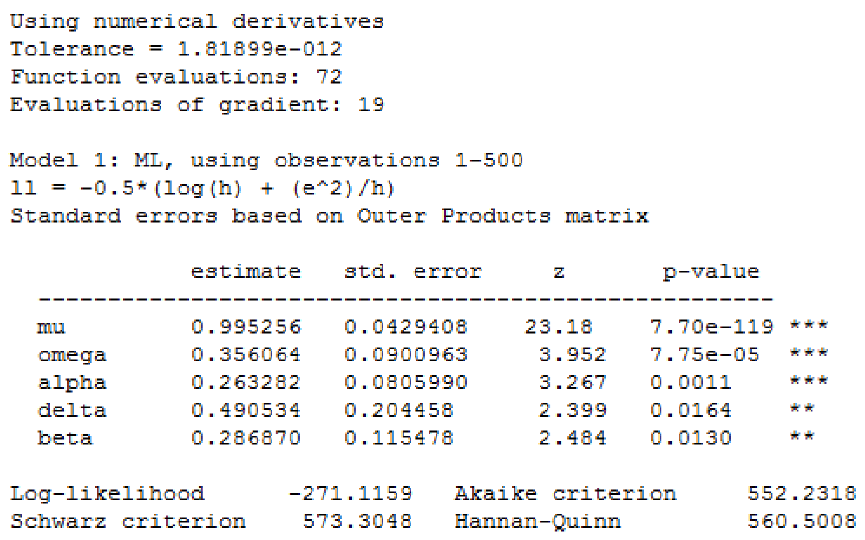
Figure 14.6: Plot of the variances after estimating the GARCH(1,1) using the BrightenYourDay returns. Right click on the graph to bring up the menu shown. Then choose `edit` to modify your graph.

Next, we have to specify an initial guess for the variances of the model, and these are set to the empirical variance of the series using `var(r)`. Then, the errors are generated, squared, and the threshold term is created using `series e2m = e2 * (e<0)`; the expression `(e<0)` takes the value of 1 for negative errors, `e`, and is zero otherwise. Then in line 14, the heteroskedastic function h_t is specified. The parameters of the model are given at the end using the `params` statement. This is required since we are going to let **gretl** try to maximize this function using numerical derivatives. The `mle` loop is ended with `end mle`. The output appears in Figure 14.8. The coefficient estimates are very close to those printed in *POE4*, and the standard errors and corresponding t-ratios are fairly similar as well. It is not unusual for estimates produced by different software to produce this kind of variation when estimating nonlinear models numerically. Different pieces of software that no doubt use different algorithms were used to numerically maximize the log-likelihood function. What is amazing here is that **gretl** does such a fine job without having to specify the analytic derivatives of the log-likelihood. Very impressive.

Gretl offers a new set of functions that estimate various kinds of GARCH models. Choose Models¿Time-series¿GARCH variants from the pull-down menu to reveal the following dialog box:

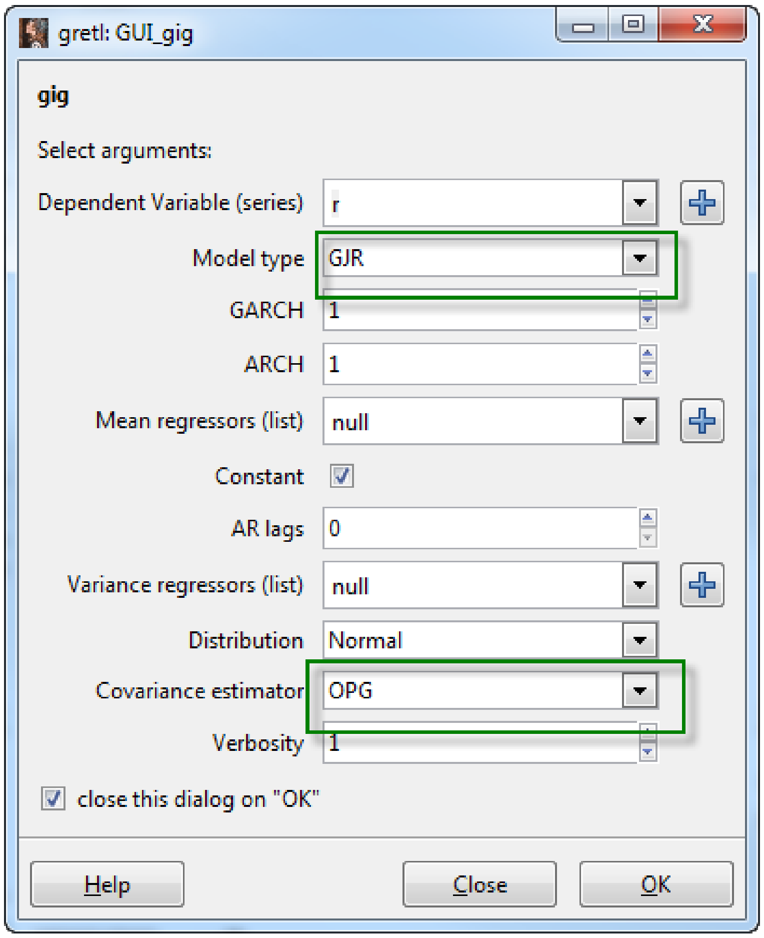
```
open "c:\Program Files\gretl\data\poe\BYD.gdt"

scalar mu = 0.5
scalar omega = .5
scalar alpha = 0.4
scalar delta = 0.1
scalar beta = 0

mle ll = -0.5*(log(h) + (e^2)/h)
   series h = var(r)
   series e = r - mu
   series e2 = e^2
   series e2m = e2 * (e<0)
   series h = omega + alpha*e2(-1) + delta*e2m(-1) + beta*h(-1)
   params mu omega alpha delta beta
end mle
```

Figure 14.7: Threshold GARCH script



```
Using numerical derivatives
Tolerance = 1.81899e-012
Function evaluations: 72
Evaluations of gradient: 19

Model 1: ML, using observations 1-500
ll = -0.5*(log(h) + (e^2)/h)
Standard errors based on Outer Products matrix

             estimate   std. error     z       p-value
  -----------------------------------------------------
  mu         0.995256   0.0429408    23.18    7.70e-119 ***
  omega      0.356064   0.0900963     3.952   7.75e-05  ***
  alpha      0.263282   0.0805990     3.267   0.0011    ***
  delta      0.490534   0.204458      2.399   0.0164    **
  beta       0.286870   0.115478      2.484   0.0130    **

Log-likelihood      -271.1159   Akaike criterion     552.2318
Schwarz criterion    573.3048   Hannan-Quinn         560.5008
```

Figure 14.8: Threshold ARCH results


gretl: GUI_gig

gig

Select arguments:

Dependent Variable (series): r

Model type: GJR

GARCH: 1

ARCH: 1

Mean regressors (list): null

Constant: ✓

AR lags: 0

Variance regressors (list): null

Distribution: Normal

Covariance estimator: OPG

Verbosity: 1

✓ close this dialog on "OK"

The GJR model type is actually equivalent to the TARCH discussed above. Estimating it with the OPG covariance estimator yields very similar results to the ones in *POE4*. This module offers several other variants of GARCH, but you will have to use the **gretl** documentation to be sure of what you are estimating. For instance, the TARCH option is not the same as the one in *POE4*.

```
Model: GJR(1,1) [Glosten et al.] (Normal)*
Dependent variable: r
Sample: 1-500 (T = 500), VCV method: OPG

    Conditional mean equation

             coefficient   std. error     z      p-value
  -------------------------------------------------------
  const       0.995450     0.0429312    23.19   6.15e-119 ***

    Conditional variance equation

             coefficient   std. error     z     p-value
  ------------------------------------------------------
```

```
omega       0.356106     0.0900902    3.953   7.73e-05 ***
alpha       0.476221     0.102614     4.641   3.47e-06 ***
gamma       0.256974     0.0873509    2.942   0.0033   ***
beta        0.286913     0.115495     2.484   0.0130   **


 (alt. parametrization)

           coefficient   std. error     z     p-value
-----------------------------------------------------
delta       0.356106     0.0900902    3.953   7.73e-05 ***
alpha       0.262915     0.0804612    3.268   0.0011   ***
gamma       0.489506     0.203966     2.400   0.0164   **
beta        0.286913     0.115495     2.484   0.0130   **

      Llik:   -730.58891        AIC:   1471.17783
      BIC:    1492.25087        HQC:   1479.44686
```

You notice that α and γ refer to two different ways to parameterize the model. The alternative refers to the TARCH model discussed in *POE4*.

14.4 Garch-in-Mean

The Garch-in-mean (MGARCH) model adds the equation's variance to the regression function. This allows the average value of the dependent variable to depend on volatility of the underlying asset. In this way, more risk (volatility) can lead to higher average return. The equations are listed below:

$$y_t = \beta_0 + \theta h_t + e_t \tag{14.9}$$
$$h_t = \delta + \alpha_1 e_{t-1}^2 + \gamma d_{t-1} e_{t-1}^2 + \beta_1 h_{t-1} \tag{14.10}$$

Notice that in this formulation we left the threshold term in the model. The errors are normally distributed with zero mean and variance h_t.

The parameters of this model can be estimated using **gretl**, though the recursive nature of the likelihood function makes it a bit more difficult. In the script below (Figure 14.9) you will notice that we've defined a function to compute the log-likelihood.[2] The function is called `gim_filter` and it contains eight arguments. The first argument is the time-series, `y`. Then, each of the parameters is listed (`mu`, `theta`, `delta`, `alpha`, `gam`, and `beta`) as scalars. The final argument is a placeholder for the variance, `h`, that is computed within the function.

Once the function is named and its arguments defined, you need to initiate series for the variances and the errors; these have been called `lh` and `le`, respectively. The log-likelihood function

[2]Actually, **gretl** genius Professor 'Jack' Lucchetti wrote the function and I'm very grateful!

is computed using a loop that runs from the second observation through the last. The length of the series can be obtained using the saved result `$nobs`, which is assigned to the variable `T`.

Gretl's loop syntax is very straightforward, though as we have shown in previous chapters, there are several variations. In this example the loop is controlled using the special index variable, `i`. In this case you specify starting and ending values for `i`, which is incremented by one each time round the loop. In the MGARCH example the loop syntax looks like this:

```
loop for i=2..T --quiet
    [Things to compute]
end loop
```

The first line start the loop using an index variable named `i`. The first value of i is set to `2`. The index `i` will increment by 1 until it reaches `T`, which has already been defined as being equal to `$nobs`. The `end loop` statement tells **gretl** the point at which to return to the top of the loop and advance the increment i. The `--quiet` option just reduces the amount of output that is written to the screen.

Within the loop itself, you'll want to lag the index and create an indicator variable that will take the value of 1 when the news is bad ($e_{t-1} < 0$). The next line squares the residual. `lh[i]` uses the loop index to place the variance computation from the iteration into the i^{th} row of `lh`. The line that begins `le[i]=` works similarly for the errors of the mean equation.

The variances are collected in `h` and the residuals in `le`, the latter of which is returned when the function is called. The function is closed using `end function`.

If this looks too complicated, you can simply highlight the code with your cursor, copy it using Ctrl-C, and paste it into a **gretl** script file (or use the scripts provided with this book).

Once the function is defined, you need to initialize each parameter just as you did in TGARCH. The series that will eventually hold the variances also must be initialized. The latter is done using `series h = NA`, which creates the series `h` and fills it with missing values (NA). The missing values for observations 2 through T are replaced as the function loops.

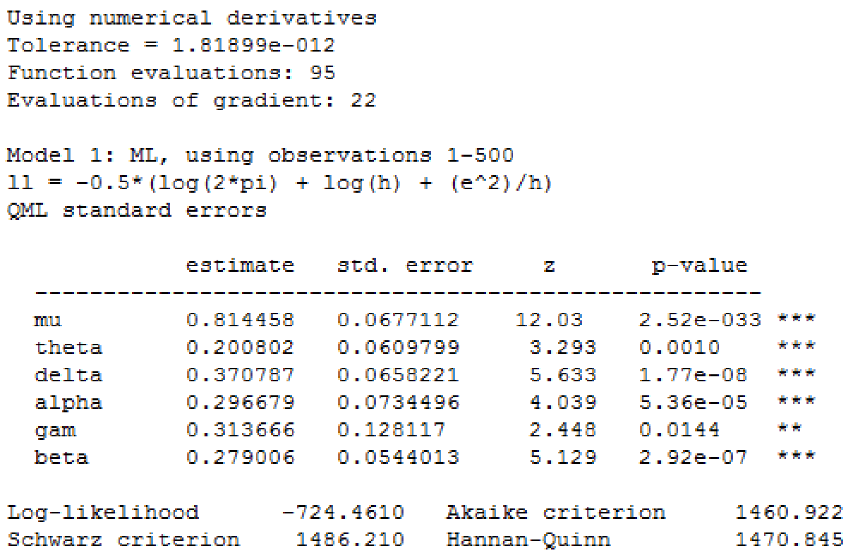
Next, the built-in `mle` command is issued and the normal density kernel is specified just as it was in the TGARCH example. Then, use the predefined `e=gim_filter( )` function, putting in the variable `r` for the time-series, the initialized parameters, and `&h` as a pointer to the variances that will be computed within the function. Issue the `params` statement to identify the parameters and have them print to the screen. Close the loop and run the script. The results appear in Figure 14.10 below. This is a difficult likelihood to maximize and **gretl** may take some time to compute the estimates. Still, it is quite remarkable that we get so close using a free piece of software and the numerical derivatives that it computes for us.

14.5 Script

```
open "@gretldir\data\poe\returns.gdt"
set echo off

scatters nasdaq allords ftse nikkei
freq nasdaq --normal
freq allords --normal
freq ftse --normal
freq nikkei --normal

open "@gretldir\data\poe\byd.gdt"

# arch(1) Using built in command for arch
arch 1 r const

# garch(0,1)=arch(1)
garch 0 1 ; r const

# garch(1,1)
garch 1 1 ; r const

# LM test for arch
ols r const
modtest 1 --arch

# LM test manually
ols r const
series ehat = $uhat
series ehat2 = ehat*ehat
ols ehat2 const ehat2(-1)
scalar tr2 = $trsq

# plotting garch variances
garch 1 1 ; r const
series ht = $h
gnuplot ht time

# threshold arch
open "@gretldir\data\poe\byd.gdt"

scalar mu = 0.5
scalar omega = .5
scalar alpha = 0.4
scalar delta = 0.1
scalar beta = 0

mle ll = -0.5*(log(h) + (e^2)/h)
   series h = var(r)
   series e = r - mu
   series e2 = e^2
   series e2m = e2 * (e<0)
   series h = omega + alpha*e2(-1) + delta*e2m(-1) + beta*h(-1)
   params mu omega alpha delta beta
end mle

# garch-in-mean -- the function
function gim_filter(series y, \
        scalar mu, scalar theta, scalar delta, scalar alpha, \
        scalar gam, scalar beta, series *h)

    series lh = var(y)
    series le = y - mu
    scalar T = $nobs
    loop for i=2..T --quiet
        scalar ilag = $i - 1
        scalar d = (le[ilag]<0)
        scalar e2lag = le[ilag]^2
        lh[i] = delta + alpha*e2lag + gam*e2lag*d + beta*lh[ilag]
        le[i] = le[i] - theta*lh[i]
    end loop

    series h = lh
    return series le

end function

# garch-in-mean
open "@gretldir\data\poe\byd.gdt"

scalar mu = 0.8
scalar gam = .1
scalar alpha = 0.4
scalar beta = 0
scalar delta = .5
scalar theta = 0.1

series h = NA

mle ll = -0.5*(log(2*pi) + log(h) + (e^2)/h)
   e = gim_filter(r, mu, theta, delta, alpha, gam, beta, &h)
   params mu theta delta alpha gam beta
end mle --robust
```

```
function gim_filter(series y, \
        scalar mu, scalar theta, scalar delta, scalar alpha, \
        scalar gam, scalar beta, series *h)

    series lh = var(y)
    series le = y - mu
    scalar T = $nobs
    loop for i=2..T --quiet
        scalar ilag = $i - 1
        scalar d = (le[ilag]<0)
        scalar e2lag = le[ilag]^2
        lh[i] = delta + alpha*e2lag + gam*e2lag*d + beta*lh[ilag]
        le[i] = le[i] - theta*lh[i]
    end loop

    series h = lh
    return series le

end function

open "c:\Program Files\gretl\data\poe\BYD.gdt"

scalar mu = 0.8
scalar gam = .1
scalar alpha = 0.4
scalar beta = 0
scalar delta = .5
scalar theta = 0.1

series h = NA

mle ll = -0.5*(log(2*pi) + log(h) + (e^2)/h)
   e = gim_filter(r, mu, theta, delta, alpha, gam, beta, &h)
   params mu theta delta alpha gam beta
end mle --robust
```

Figure 14.9: The MGARCH script includes a function to compute the log-likelihood.



```
Using numerical derivatives
Tolerance = 1.81899e-012
Function evaluations: 95
Evaluations of gradient: 22

Model 1: ML, using observations 1-500
ll = -0.5*(log(2*pi) + log(h) + (e^2)/h)
QML standard errors

             estimate   std. error     z       p-value
  -----------------------------------------------------
  mu         0.814458   0.0677112    12.03    2.52e-033 ***
  theta      0.200802   0.0609799     3.293   0.0010    ***
  delta      0.370787   0.0658221     5.633   1.77e-08  ***
  alpha      0.296679   0.0734496     4.039   5.36e-05  ***
  gam        0.313666   0.128117      2.448   0.0144    **
  beta       0.279006   0.0544013     5.129   2.92e-07  ***

Log-likelihood      -724.4610   Akaike criterion     1460.922
Schwarz criterion    1486.210   Hannan-Quinn         1470.845
```

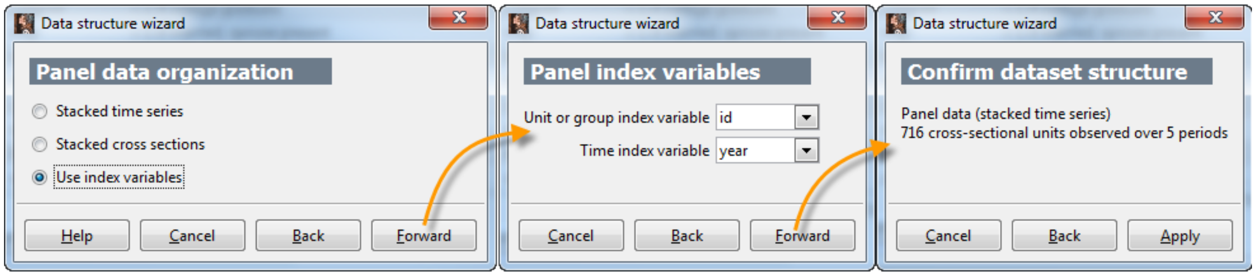
Figure 14.10: Garch-in-mean results

Chapter 15

Pooling Time-Series and Cross-Sectional Data

A panel of data consists of a group of cross-sectional units (people, firms, states or countries) that are observed over time. Following Hill et al. (2011) we will denote the number of cross-sectional units by N and the number of time periods we observe them as T.

In order to use the predefined procedures for estimating models using panel data in **gretl** you have to first make sure that your data have been properly structured in the program. The dialog boxes for assigning panel dataset structure using index variables is shown below:



( ) Stacked time series
( ) Stacked cross sections
[✓ Use index variables]

Unit or group index variable: id
Time index variable: year

Panel data (stacked time series)
716 cross-sectional units observed over 5 periods

To use this method, the data have to include variables that identify each individual and time period. Select the `Panel` option using the radio button and **gretl** will then be able to work behind the scenes to accurately account for the time and individual dimensions. The datasets that come with this manual have already been setup this way, but if you are using your own data you'll want to to assign the proper dataset structure to it so that all of the appropriate panel data procedures are available for use.

Gretl gives you easy access to a number of useful panel data sets via its database server.[1] These include the Penn World Table and the Barro and Lee (1996) data on international educational

[1]Your computer must have access to the internet to use this.

attainment. These data can be installed using `File>Databases>On database server` from the menu bar as shown in Figure 15.1 below. From here, select a database you want. In Figure 15.2

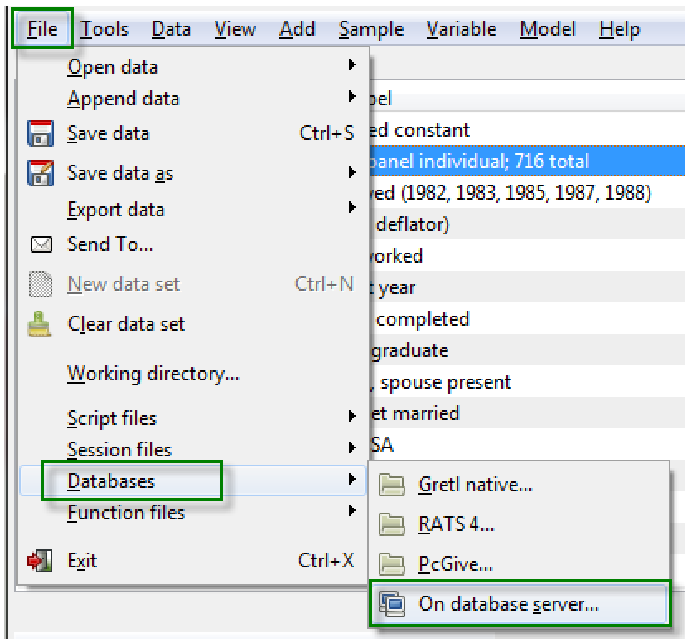
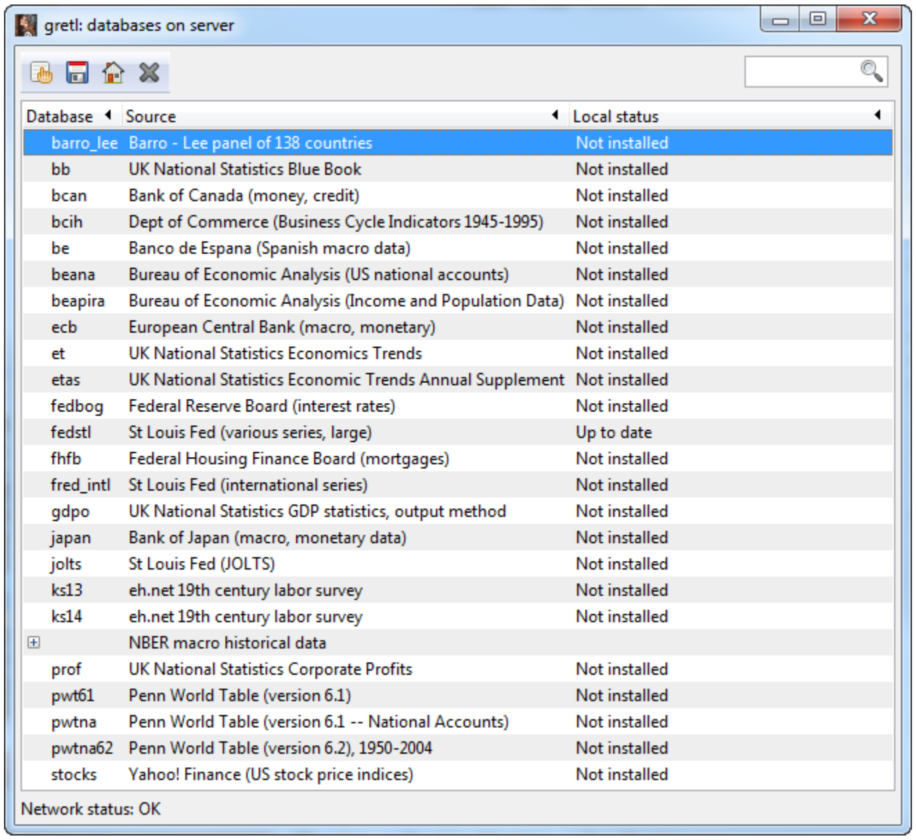


Figure 15.1: Accessing data from the database server via the pull-down menus

the entry for the Barro-Lee panel is highlighted. To its right, you are given information about whether that dataset is installed on your computer. Double click on `barro_lee` and a listing of the series in this database will appear in a new window. From that window you can search for a particular series, display observations, graph a series, or import it. This is a VERY useful utility, both for teaching and research and I encourage you to explore what is available on the **gretl** server. You will notice the 4 icons at the top of the window

The first icon from the left is the list series icon. Clicking it will bring up the list of series in a particular database as shown below in Figure 15.3. The icon that looks like a floppy disk (remember those?) will install the database. The clicking the next icon will show which databases are installed on your computer, and the 'X' closes the window.

Figure 15 shows a portion of the series list window for the Barro and Lee data from the database server. From here you can display the values contained in a series, plot the series, or add a series to your dataset. Highlight the particular series you want and click on the appropriate icon at the top.


gretl: databases on server

Database	Source	Local status
barro_lee	Barro - Lee panel of 138 countries	Not installed
bb	UK National Statistics Blue Book	Not installed
bcan	Bank of Canada (money, credit)	Not installed
bcih	Dept of Commerce (Business Cycle Indicators 1945-1995)	Not installed
be	Banco de Espana (Spanish macro data)	Not installed
beana	Bureau of Economic Analysis (US national accounts)	Not installed
beapira	Bureau of Economic Analysis (Income and Population Data)	Not installed
ecb	European Central Bank (macro, monetary)	Not installed
et	UK National Statistics Economics Trends	Not installed
etas	UK National Statistics Economic Trends Annual Supplement	Not installed
fedbog	Federal Reserve Board (interest rates)	Not installed
fedstl	St Louis Fed (various series, large)	Up to date
fhfb	Federal Housing Finance Board (mortgages)	Not installed
fred_intl	St Louis Fed (international series)	Not installed
gdpo	UK National Statistics GDP statistics, output method	Not installed
japan	Bank of Japan (macro, monetary data)	Not installed
jolts	St Louis Fed (JOLTS)	Not installed
ks13	eh.net 19th century labor survey	Not installed
ks14	eh.net 19th century labor survey	Not installed
⊞	NBER macro historical data	
prof	UK National Statistics Corporate Profits	Not installed
pwt61	Penn World Table (version 6.1)	Not installed
pwtna	Penn World Table (version 6.1 -- National Accounts)	Not installed
pwtna62	Penn World Table (version 6.2), 1950-2004	Not installed
stocks	Yahoo! Finance (US stock price indices)	Not installed

Network status: OK

Figure 15.2: Installing a data from the database server via the pull-down menus

15.1 A Basic Model

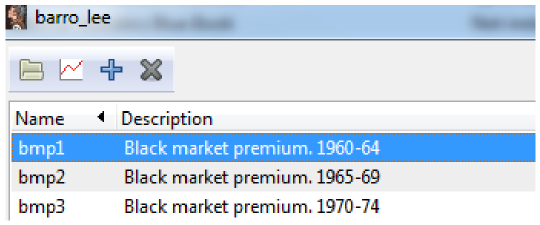
The most general expression of linear regression models that have both time and unit dimensions is seen in equation 15.1 below.

$$y_{it} = \beta_{1it} + \beta_{2it}x_{2it} + \beta_{3it}x_{3it} + e_{it} \tag{15.1}$$

where $i = 1, 2, \ldots, N$ and $t = 1, 2, \ldots, T$. If we have a full set of time observations for every individual then there will be NT total observations in the sample. The panel is said to be **balanced** in this case. It is not unusual to have some missing time observations for one or more individuals. When this happens, the total number of observation is less than NT and the panel is **unbalanced**.

The biggest problem with equation (15.1) is that even if the panel is complete (balanced), the model contains 3 times as many parameters as observations (NT)! To be able to estimate the model, some assumptions have to be made in order to reduce the number of parameters. One of the most common assumptions is that the slopes are constant for each individual and every time period; also, the intercepts vary only by individual. This model is shown in equation (15.2).

$$y_{it} = \beta_{1i} + \beta_2 x_{2it} + \beta_3 x_{3it} + e_{it} \tag{15.2}$$


barro_lee

Name	Description
bmp1	Black market premium. 1960-64
bmp2	Black market premium. 1965-69
bmp3	Black market premium. 1970-74

Figure 15.3: This shows a portion of the series list window for the Barro and Lee data from the database server. From here you can display the values contained in a series, plot the series, or add a series to your dataset. Highlight the particular series you want and click on the appropriate icon at the top.

This specification, which includes $N + 2$ parameters, includes dummy variables that allow the intercept to shift for each individual. By using such a model you are saying that over short time periods there are no substantive changes in the regression function. Obviously, the longer your time dimension, the more likely this assumption will be false.

In equation (15.2) the parameters that vary by individual are called **individual fixed effects** and the model is referred to as **one-way fixed effects**. The model is suitable when the individuals in the sample differ from one another in a way that does not vary over time. It is a useful way to avoid unobserved differences among the individuals in your sample that would otherwise have to be omitted from consideration. Remember, omitting relevant variables may cause least squares to be biased and inconsistent; a one-way fixed effects model, which requires the use of panel data, can be very useful in mitigating the bias associated with time invariant, unobservable effects.

If you have a longer panel and are concerned that the regression function is shifting over time, you can add $T - 1$ time dummy variables to the model. The model becomes

$$y_{it} = \beta_{1i} + \beta_{1t} + \beta_2 x_{2it} + \beta_3 x_{3it} + e_{it} \tag{15.3}$$

where either β_{1i} or β_{1t} have to be omitted in order to avoid perfect collinearity. This model contains $N + (T - 1) + 2$ parameters which is generally fewer than the NT observations in the sample. Equation (15.3) is called the **two-way fixed effects** model because it contains parameters that will be estimated for each individual and each time period.

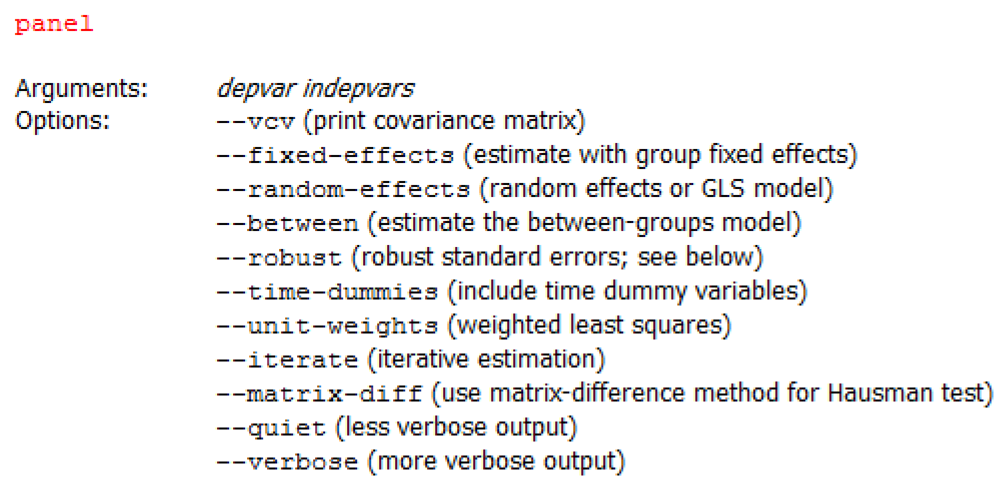
15.2 Estimation

Hill et al. (2011) provides a subset of National Longitudinal Survey which is conducted by the U.S. Department of Labor. The database includes observations on women, who in 1968, were between the ages of 14 and 24. It then follows them through time, recording various aspects of their lives annually until 1973 and bi-annually afterwards. Our sample consists of 716 women observed in 5 years (1982, 1983, 1985, 1987 and 1988). The panel is balanced and there are 3580 total observations.

Two model considered is found in equation (15.2) below.

$$\begin{aligned} ln(wage)_{it} &= \beta_{1i} + \beta_2 educ_{it} + \beta_3 exper_{it} + \beta_4 exper^2_{it} + \beta_5 tenure_{it} \\ &+\beta_6 tenure^2_{it} + \beta_7 south_{it} + \beta_8 union_{it} + \beta_9 black_{it} + e_{it} \end{aligned} \tag{15.4}$$

The main command used to estimate models with panel data in **gretl** is panel. The panel syntax is:



```
panel

Arguments:     depvar indepvars
Options:       --vcv (print covariance matrix)
               --fixed-effects (estimate with group fixed effects)
               --random-effects (random effects or GLS model)
               --between (estimate the between-groups model)
               --robust (robust standard errors; see below)
               --time-dummies (include time dummy variables)
               --unit-weights (weighted least squares)
               --iterate (iterative estimation)
               --matrix-diff (use matrix-difference method for Hausman test)
               --quiet (less verbose output)
               --verbose (more verbose output)
```

All of the basic panel data estimators are available. Fixed effects, two-way fixed effects, random effects, between estimation and (not shown) pooled least squares.

The first model to be estimated is referred to as pooled least squares. Basically, it imposes the restriction that $\beta_{1i} = \beta 1$ for all individuals. The individuals have the same intercepts. Applying pooled least squares in a panel is restrictive in a number of ways. First, to estimate the model using least squares violates at least one assumption that is used in the proof of the Gauss-Markov theorem. It is almost certain that errors for an individual will be correlated. If Johnny isn't the sharpest marble in the bag, it is likely that his earnings given equivalent education, experience, tenure and so on will be on the low side of average for each year. He has low ability and that affects each year's average wage similarly.

It is also possible that an individual may have smaller of larger earnings variance compared to others in the sample. The solution to these specification issues is to use robust estimates of the variance covariance matrix. Recall that least squares is consistent for the slopes and intercept (but not efficient) when errors are correlated or heteroskedastic, but that this changes the nature of the variance-covariance.

Robust covariances in panel data take into account the special nature of these data. Specifically they account for autocorrelation within the observations on each individual and they allow the variances for different individuals to vary. Since panel data have both a time-series and a cross-sectional dimension one might expect that, in general, robust estimation of the covariance matrix would require handling both heteroskedasticity and autocorrelation (the HAC approach).

Gretl currently offers two robust covariance matrix estimators specifically for panel data. These are available for models estimated via fixed effects, pooled OLS, and pooled two-stage least squares. The default robust estimator is that suggested by Arellano (2003), which is HAC provided the panel is of the "large n, small T" variety (that is, many units are observed in relatively few periods).

In cases where autocorrelation is not an issue, however, the estimator proposed by Beck and Katz (1995) and discussed by Greene (2003, chapter 13) may be appropriate. This estimator takes into account contemporaneous correlation across the units and heteroskedasticity by unit.

```
open "@gretldir\data\poe\nls_panel.gdt"
list xvars = const educ exper exper2 tenure tenure2 south black union
panel lwage xvars --pooled --robust
```

The first thing to notice is that even though the model is being estimated by least squares, the `panel` command is used with the `--pooled` option. The `--robust` option requests the default HCCME for panel data which is basically a special version of HAC (see section 9.6.1).

Pooled OLS, using 3580 observations
Included 716 cross-sectional units
Time-series length = 5
Dependent variable: lwage
Robust (HAC) standard errors

	Coefficient	Std. Error	t-ratio	p-value
const	0.476600	0.0844094	5.6463	0.0000
educ	0.0714488	0.00548952	13.0155	0.0000
exper	0.0556851	0.0112896	4.9324	0.0000
exper2	−0.00114754	0.000491577	−2.3344	0.0196
tenure	0.0149600	0.00711024	2.1040	0.0354
tenure2	−0.000486042	0.000409482	−1.1870	0.2353
south	−0.106003	0.0270124	−3.9242	0.0001
black	−0.116714	0.0280831	−4.1560	0.0000
union	0.132243	0.0270255	4.8933	0.0000

$\hat{\rho}$ 0.811231 Durbin–Watson 0.337344

As long as omitted effects (e.g., individual differences) are uncorrelated with any of the regressors, these estimates are consistent. If the individual differences are correlated with regressors, then you can estimate the model's parameters consistently using fixed effects.

For comparison purposes, the pooled least squares results without cluster standard errors is shown

Pooled OLS, using 3580 observations
Included 716 cross-sectional units
Time-series length = 5
Dependent variable: lwage

	Coefficient	Std. Error	t-ratio	p-value
const	0.476600	0.0561559	8.4871	0.0000
educ	0.0714488	0.00268939	26.5669	0.0000
exper	0.0556851	0.00860716	6.4696	0.0000
exper2	−0.00114754	0.000361287	−3.1763	0.0015
tenure	0.0149600	0.00440728	3.3944	0.0007
tenure2	−0.000486042	0.000257704	−1.8860	0.0594
south	−0.106003	0.0142008	−7.4645	0.0000
black	−0.116714	0.0157159	−7.4265	0.0000
union	0.132243	0.0149616	8.8388	0.0000

You can see that the estimates are the same those from the first set, but that the standard errors are much smaller. Given that the data come from a panel, and likely to suffer both within group autocorrelation and between group heteroskedasticity, the t-ratios could be highly misleading.

15.3 Fixed Effects

The model (15.2) is reestimated using fixed effects. Race and education do not change for individuals in the sample, and their influences cannot be estimated using fixed effects.

```
1 open "c:\Program Files\gretl\data\poe\nels_panel.gdt"
2 list xvars = const educ exper exper2 tenure tenure2 south union black
3 panel lwage xvars --fixed-effects
4
5 xvars -= educ black
6 panel lwage xvars --fixed-effects
```

Even though the parameters for `black` and `educ` are not identified in this model, we included them anyway in line 3 just to see how **gretl** handles this. The results are:

Fixed-effects, using 3580 observations
Included 716 cross-sectional units

Time-series length = 5
Dependent variable: lwage

	Coefficient	Std. Error	t-ratio	p-value
const	1.45003	0.0401400	36.1244	0.0000
exper	0.0410832	0.00662001	6.2059	0.0000
exper2	−0.000409052	0.000273333	−1.4965	0.1346
tenure	0.0139089	0.00327784	4.2433	0.0000
tenure2	−0.000896227	0.000205860	−4.3536	0.0000
south	−0.0163224	0.0361490	−0.4515	0.6516
union	0.0636972	0.0142538	4.4688	0.0000

Test for differing group intercepts –
Null hypothesis: The groups have a common intercept
Test statistic: $F(715, 2858) = 15.145$
with p-value = $P(F(715, 2858) > 15.145) = 0$

Cleverly, **gretl** has dropped `educ` and `black` from the model. It also reports a test of the hypothesis that the individual differences are jointly equal to zero. Failure to reject this hypothesis would lead to the pooled least squares estimates. The p-value is near zero and the equality of intercepts is rejected.

In line 5, we've used a special **gretl** trick that can be used to remove items from a list. The operator is `-=` and in this line the variables `educ` and `black` are removed from the `xvars` list. You can add things to a list using `+=`.

As pointed out in *POE4*, when N is small you can create a set of dummy variables for the fixed effects and estimate the model using least squares. This is equivalent to using the fixed effects estimator. The *nls_panel10.gdt* contains a subset of 10 individuals from the larger set of 716 and we can use it to demonstrate some features of **gretl** and the equivalence of the two procedures.

The first step is to create a set of indicator variables for each individual.

```
open "@gretldir\data\poe\nls_panel10.gdt"
setobs id year --panel-vars
genr unitdum
list x = exper exper2 tenure tenure2 union
ols lwage x du_*
panel lwage x --fixed-effects
```

Since the dataset has been declared to be a panel, **gretl** knows that the `id` variable identifies individuals. Hence, `genr unitdum` generates an indicator for each unique `id`. This is a special circumstance where the `genr` command must be used instead of `series`. The indicator variables are added to the dataset and are given names and variable ID numbers. The name of the first indicator is du_1 which takes a 1 if individual has id=1 and 0 otherwise. The remaining individuals also get an indicator variable, the last being du_10. The use of the wildcard * in line 5 reduces the amount of typing. The * will pick up every variable that begins du_. In this model du_* is equivalent to du_1 du_2 du_3 du_4 du_5 du_6 du_7 du_8 du_9 du_10.

The results from least squares dummy variable estimation and the equivalent fixed effects panel appear below in Table 15.1. The advantage of using the panel fixed effects version is that when there

```
Model 1: Pooled OLS, using 50 observations
Included 10 cross-sectional units
Time-series length = 5
Dependent variable: lwage

             coefficient   std. error   t-ratio    p-value
  --------------------------------------------------------
  exper       0.237999     0.187757      1.268     0.2133
  exper2     -0.00818817   0.00790482   -1.036     0.3074
  tenure     -0.0123500    0.0341433    -0.3617    0.7197
  tenure2     0.00229615   0.00268846    0.8541    0.3989
  union       0.113543     0.150863      0.7526    0.4567
  du_1        0.151905     1.09675       0.1385    0.8906
  du_2        0.186894     1.07148       0.1744    0.8625
  du_3       -0.0630423    1.35092      -0.04667   0.9630
  du_4        0.185626     1.34350       0.1382    0.8909
  du_5        0.938987     1.09778       0.8554    0.3982
  du_6        0.794485     1.11177       0.7146    0.4796
  du_7        0.581199     1.23591       0.4703    0.6411
  du_8        0.537925     1.09750       0.4901    0.6271
  du_9        0.418334     1.08405       0.3859    0.7019
  du_10       0.614558     1.09018       0.5637    0.5765

Model 2: Fixed-effects, using 50 observations
Included 10 cross-sectional units
Time-series length = 5
Dependent variable: lwage

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const       0.434687     1.14518       0.3796   0.7066
  exper       0.237999     0.187757      1.268    0.2133
  exper2     -0.00818817   0.00790482   -1.036    0.3074
  tenure     -0.0123500    0.0341433    -0.3617   0.7197
  tenure2     0.00229615   0.00268846    0.8541   0.3989
  union       0.113543     0.150863      0.7526   0.4567
```

Table 15.1: Comparison of fixed effects and least squares dummy variable estimators.

are many individuals, the output of the coefficients on the fixed effects themselves is suppressed. When N is large, you are seldom interested in the values of these parameters anyway.

15.4 Random Effects

The random effects estimator treats the individual differences as being randomly assigned to the individuals. Rather than estimate them as parameters as we did in the fixed effects model, here they are incorporated into the model's error, which in a panel will have a specific structure. The β_{1i} term in equation 15.3 is modeled:

$$\beta_{1i} = \bar{\beta}_1 + u_i \tag{15.5}$$

where the u_i are random individual differences that are the same in each time period.

$$\begin{align} y_{it} &= \bar{\beta}_1 + \beta_2 x_{2it} + \beta_3 x_{3it} + (e_{it} + u_i) \tag{15.6} \\ &= \bar{\beta}_1 + \beta_2 x_{2it} + \beta_3 x_{3it} + v_{it} \tag{15.7} \end{align}$$

where the combined error is

$$v_{it} = u_i + e_{it}$$

the key property of the new error term is that it is homoskedastic

$$\sigma_v^2 = \text{var}\,(v_{it}) = \text{var}\,(u_i + e_{it}) = \sigma_u^2 + \sigma_e^2 \tag{15.8}$$

and serially correlated. For individual i, that covariance among his errors is

$$\text{cov}\,(v_{it}, v_{is}) = \sigma_u^2$$

for $i \neq s$. The covariance between any two individuals is zero. One of the key advantages of the random effects model is that parameters on time invariant regressors can be estimated. That means that coefficients on *black* and *educ* can be estimated. Not so with fixed effects.

The parameter estimates are actually obtained through feasible generalized least squares. Equation 15.8 contains two parameters that describe the variances and covariances in the model. These are estimated and used to perform FGLS. The process is described in some detail in *POE4* and will not be discussed in much detail here. However, when **gretl** estimates the model as specified, it refers to the results as 'GLS'.

The transformation that is used on the variables of the model is sometimes referred to as quasi-demeaning. It is based on the computation of

$$\theta = 1 - \frac{\sigma_e}{\sqrt{T\sigma_u^2 + \sigma_e^2}} \tag{15.9}$$

This parameter θ is estimated from the data and the transformation are

$$y_{it}^* = y_{it} - \theta \bar{y}_i, \quad x_{1it}^* = 1 - \theta, \quad x_{2it}^* = x_{2it} - \theta \bar{x}_{2i}, \quad x_{3it}^* = x_{3it} - \theta \bar{x}_{3i} \tag{15.10}$$

The bars over the variables indicate means for the i^{th} individual taken over the available time periods. **Gretl** estimates θ and the variances. In the wage equation the estimate of σ_e^2, σ_u^2 and θ are, respectively:

```
'Within' variance = 0.0380681
'Between' variance = 0.115887
theta used for quasi-demeaning = 0.743683
```

These match the ones in *POE4* exactly.

If the random individual effects are correlated with regressors, then the random effects estimator will not be consistent. A statistical test of this proposition should be done whenever this estimator is used in order to reduce the chance of model misspecification.

To estimate the parameters of this model in **gretl** is easy. Simply specify the model you want to estimate and choose the random effects option.

```
open "@gretldir\data\poe\nls_panel.gdt"
setobs id year --panel-vars
list x1 = educ exper exper2 tenure tenure2 union black south
panel lwage x1 --random-effects
```

The results from FGLS estimation of the random effects model are shown in Table 15.3.

15.5 Between Estimator

Before discussing such tests, another estimator of the model's parameters deserves mention. The between estimator is also used in some circumstances. The between model is

$$\bar{y}_i = \bar{\beta}_1 + \beta_2 \bar{x}_{2i} + \beta_3 \bar{x}_{3i} + u_i + \bar{e}_i \tag{15.11}$$

where the $\bar{y}_i$ is the average value of y for individual i, and $\bar{x}_{ki}$ is the average value of the k^{th} regressor for individual i. Essentially, the observation in each group (or individual) are averaged over time. The parameters are then estimated by least squares. The variation between individuals is being used to estimate parameters. The errors are uncorrelated across individuals and homoskedastic and as long as individual differences are not correlated with regressors, the between estimator should be consistent for the parameters.

To obtain the between estimates, simply use the `--between` option of `panel` as shown below:

```
open "@gretldir\data\poe\nls_panel.gdt"
setobs id year --panel-vars
list x1 = educ exper exper2 tenure tenure2 union black south
panel lwage x1 --between
```

Dependent variable: lwage

	(1) Within	(2) FGLS	(3) Between	(4) Pooled OLS
const	1.45**	0.534**	0.417**	0.477**
	(36.1)	(6.68)	(3.07)	(5.65)
exper	0.0411**	0.0436**	0.0662**	0.0557**
	(6.21)	(6.86)	(2.82)	(4.93)
exper2	−0.000409	−0.000561**	−0.00161	−0.00115**
	(−1.50)	(−2.14)	(−1.61)	(−2.33)
tenure	0.0139**	0.0142**	0.0166	0.0150**
	(4.24)	(4.47)	(1.36)	(2.10)
tenure2	−0.000896**	−0.000755**	−0.000495	−0.000486
	(−4.35)	(−3.88)	(−0.704)	(−1.19)
south	−0.0163	−0.0818**	−0.105**	−0.106**
	(−0.452)	(−3.65)	(−3.62)	(−3.92)
union	0.0637**	0.0802**	0.156**	0.132**
	(4.47)	(6.07)	(4.39)	(4.89)
educ		0.0733**	0.0708**	0.0714**
		(13.7)	(13.1)	(13.0)
black		−0.117**	−0.122**	−0.117**
		(−3.86)	(−3.84)	(−4.16)
n	3580	3580	716	3580
$\bar{R}^2$	0.824		0.358	0.324
ℓ	1.17e+003	−1.65e+003	−240	−1.63e+003

t-statistics in parentheses
* indicates significance at the 10 percent level
** indicates significance at the 5 percent level

Table 15.3: Fixed Effects (Within), Random Effects (FGLS), Between, and Pooled OLS with robust standard errors.

The results for each of the estimators, in tabular form, are in Table 15.3. Wisely, **gretl** has omitted the R^2 for the random effects model. Recall that R^2 is only suitable for linear models estimated using OLS, which is the case for one-way fixed effects. There is not a lot of variation in these results, suggesting that perhaps the unobserved individual differences are not significantly correlated with the model's regressors.

15.6 Specification Tests

There are a couple of key specification tests one must do before relying on the between, random effects, or pooled least squares estimators. For consistency all require that the unobserved heterogeneity be uncorrelated with the model's regressors. This is tested using a version of a Hausman test. The other test is for the presence of random effects. This test is an LM test that is sometimes referred to as Breusch-Pagan, although there are tests of other hypotheses that go by the latter.

15.6.1 Breusch-Pagan Test

The Breusch Pagan test statistic is based on a Lagrange multiplier and is computed

$$LM = \sqrt{\frac{NT}{2\,(T-1)}}\left\{\frac{\sum\limits_{i=1}^{N}\left(\sum\limits_{t=1}^{T}\hat{e}_{it}\right)^2}{\sum\limits_{i=1}^{N}\sum\limits_{t=1}^{T}\hat{e}_{it}^2} - 1\right\} \tag{15.12}$$

The null hypothesis is $H_0 : \sigma_u^2 = 0$ against the alternative that $H_1 : \sigma_u^2 \geq 0$. Under the null, $LM \sim N(0,1)$ and the best idea would be to perform a one-sided test. Unfortunately, **gretl** and most other pieces of software report LM^2 and use the χ^2_1 which makes the alternative $H_1 : \sigma_u^2 \neq 0$.

The good news is that at least **gretl** computes LM^2 by default whenever you estimate a random effects model. Rejection of the null means that the individual (and in this model, random) differences have variance. If you fail to reject then you probably want to use pooled least squares.

For the wage model the result is

```
Breusch-Pagan test -
  Null hypothesis: Variance of the unit-specific error = 0
  Asymptotic test statistic: Chi-square(1) = 3859.28
  with p-value = 0
```

The statistic is 3859.28, which is very large and the no random effects hypothesis is rejected at the 5% level.

15.6.2 Hausman Test

The Hausman test probes the consistency of the random effects estimator. The null hypothesis is that these estimates are consistent–that is, that the requirement of orthogonality of the model's errors and the regressors is satisfied. The test is based on a measure, H, of the "distance" between the fixed-effects and random-effects estimates, constructed such that under the null it follows the χ^2 distribution with degrees of freedom equal to the number of time-varying regressors, J. If the value of H is "large" this suggests that the random effects estimator is not consistent and the fixed-effects model is preferable.

There are two ways of calculating H, the matrix-difference method and the regression method. The procedure for the matrix-difference method is this:

- Collect the fixed-effects estimates in a vector, $\tilde{\beta}$, and the corresponding random-effects estimates in $\hat{\beta}$, then form the difference vector $(\tilde{\beta} - \hat{\beta})$
- Form the covariance matrix of the difference vector as $var(\tilde{\beta} - \hat{\beta}) = var(\tilde{\beta}) - var(\hat{\beta}) = \Psi$. The two variance covariance matrices are estimated using the sample variance matrices of the fixed- and random-effects models respectively.
- Compute the quadratic form $H = (\tilde{\beta} - \hat{\beta})'\hat{\Psi}^{-1}(\tilde{\beta} - \hat{\beta}) \sim \chi^2_J$ if the errors and regressors are not correlated.

Given the relative efficiencies of $\tilde{\beta}$ and $\hat{\beta}$, the matrix $\hat{\Psi}$ "should be" positive definite, in which case H is positive, but in finite samples this is not guaranteed and of course a negative χ^2 value is not admissible.

The regression method avoids this potential problem. The procedure is:

- Treat the random-effects model as the restricted model, and record its sum of squared residuals as *SSRr*.
- Estimate via OLS an unrestricted model in which the dependent variable is quasi-demeaned y and the regressors include both quasi-demeaned X (as in the RE model) and the demeaned variants of all the time-varying variables (i.e. the fixed-effects regressors); record the sum of squared residuals from this model as $SSRu$.
- Compute $H = n(SSRr - SSRu)/SSRu$, where n is the total number of observations used. On this variant H cannot be negative, since adding additional regressors to the RE model cannot raise the SSR. See chapter 16 of the **Gretl** Users Guide for more details.

By default **gretl** computes the Hausman test via the regression method, but it uses the matrix difference method if you pass the option `--matrix-diff` to the `panel` command.

In the wage example, the Hausman test results are:

```
Hausman test -
  Null hypothesis: GLS estimates are consistent
  Asymptotic test statistic: Chi-square(6) = 20.5231
  with p-value = 0.00223382
```

The p-value is less than 5% which suggests that the random effects estimator is inconsistent. The conclusion from these tests is that even though there is evidence of random effects (LM rejects), the random effects are not independent of the regressors; the FGLS estimator will be inconsistent and you'll have to use the fixed effects estimator of a model that excludes education and race.

15.7 Seemingly Unrelated Regressions

The acronym SUR stands for **seemingly unrelated regression** equations. SUR is another way of estimating panel data models that are long (large T), but not wide (small N). More generally though, it is used to estimate systems of equations that do not necessarily have any parameters in common and whose regression functions do not appear to be related. In the SUR framework, each firm in your sample is parametrically different; each firm has its own regression function, i.e., different intercept and slopes. Firms are not totally unrelated, however. In this model the firms are linked by what is **not** included in the regression rather than by what is. The firms are thus related by unobserved factors and SUR requires us to specify how these omitted factors are linked in the system's error structure.

In the basic SUR model, the errors are assumed to be homoscedastic and linearly independent within each equation, or in our case, each firm. The error of each equation may have its own variance. Most importantly, each equation (firm) is correlated with the others in the same time period. The latter assumption is called **contemporaneous correlation**, and it is this property that sets SUR apart from other models.

Now consider the investment model suggested by Grunfeld (1958). Considering investment decisions of only two firms, General Electric (g) and Westinghouse (w), we have

$$inv_{gt} = \beta_{1g} + \beta_{2g}v_{gt} + \beta_{3g}k_{gt} + e_{gt} \tag{15.13}$$

$$inv_{wt} = \beta_{1w} + \beta_{2w}v_{wt} + \beta_{3w}k_{wt} + e_{wt} \tag{15.14}$$

where $t = 1, 2, \ldots, 20$, k is capital stock and v is value of the firm. In the context of the two firm Grunfeld model in (15.13) and (15.14) this would mean that $var[e_{gt}] = \sigma_g^2; var[e_{wt}] = \sigma_w^2$; $cov(e_{gt}, e_{wt}) = \sigma_{gw}$ for all time periods; and $cov(e_{it}, e_{is}) = 0$ for $t \neq s$ for each firm, $i = g, w$. So in the SUR model you essentially have to estimate a variance for each individual and a covariance between each pair of individuals. These are then used to construct a feasible generalized least squares estimator of the equations parameters.

Even though SUR requires a T and an N dimension, it is not specifically a panel technique. This is because the equations in an SUR system may be modeling different behaviors for a single individual rather than the same behavior for several individuals. As mentioned before, it is best used when panels are long and narrow since this gives you more observations to estimate the equations variances and the cross equation covariances. More time observations reduces the sampling variation associated with these estimates, which in turn improves the performance of the feasible generalized least squares estimator. If your panel dataset has a very large number of individuals and only a few years, then FGLS may not perform very well in a statistical sense. In the two firm Grunfeld example, N=2 and T=20 so we needn't worry about this warning too much, although the asymptotic inferences are based on T (and not N) being infinite.

The two firm example is from Hill et al. (2011) who have provided the data in the *grunfeld2.gdt* data set. The first model we estimate is the pooled model, estimated by least squares. This is done in lines 2 and 3.

```
1  open "@gretldir\data\poe\grunfeld2.gdt"
2  list xvars = const v k
3  ols inv xvars
4  modeltab add
```

The second model estimates the investment equations for each firm separately. There are a number of ways to do this and several are explored below. The first method uses interaction terms (see chapter 7) to estimate the two equations. Basically, an indicator variable is interacted with each regressor, including the constant. The two firm model would be:

$$inv = \beta_1 + \beta_2 k + \beta_3 v + \beta_4 d + \beta_5 (d \times k) + \beta_6 (d \times v) + e \tag{15.15}$$

where d is the firm indicator. In the script the interactions are created using a loop. In this way, you could automate the procedure for any number of explanatory variables. Line 5 generates the set of indicators for the units of the panel. There are only two units in this panel, so only two indicators are created named `du_1` and `du_2`. An empty list called `Z` is created. `Z` will be used to hold the variables created in the loop. The `foreach` loop is used in this example. The index is called `i` and it will loop over each element of the variable list, `X`. In line 8 the interaction term is assigned to a series. The name will be `d$i`, which as the loop proceeds will be `d`*`varname`*. The next line creates the variable list, adding the new interaction at each iteration. Finally, the model is estimated using the original regressors.

```
5  series unitdum
6  list dZ = null
7  loop foreach i X
8  series d$i = du_2 * $i
9     list dZ = dZ d$i
10 endloop
11 ols inv X dZ
12 modeltab add
13 modeltab show
```

The results appear below.

```
Pooled OLS estimates
Dependent variable: inv

                  (1)          (2)

const            17.87**      -9.956
                 (7.024)      (23.63)

v               0.01519**    0.02655**
              (0.006196)    (0.01172)

k                0.1436**     0.1517**
               (0.01860)    (0.01936)

dconst                         9.447
                              (28.81)

dv                            0.02634
                             (0.03435)

dk                           -0.05929
                              (0.1169)

       n            40           40
Adj. R**2       0.7995       0.8025
     lnL        -177.3       -175.3

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

They match those in Table 13.12 of *POE4*. One of the disadvantages of estimating the separate equations in this way is that it assumes that the error variances of the two firms are equal. If this is not true, then standard errors and t-ratios will not be valid. You could use a robust covariance estimator or estimate the model via groupwise heteroskedasticity. Also, the use of interaction terms complicates the interpretation of the coefficients a bit. The coefficients on the interaction terms are measuring the difference in effect between the interacted group and the reference group (GE). To get the marginal effect of an increase in k on average investment for Westinghouse we would add $\beta_2 + \beta_5$. The computation based on the least squares estimates is $0.1517 - 0.0593 = 0.0924$.

The next method of estimating the equations separately is better. It allows the variances of each subset to differ. The **gretl** script to estimate the two firm model using this data

```
wls du_1 inv v k const
wls du_2 inv v k const
```

This uses the trick explored earlier where observations can be included or excluded when weighted by 1 or 0, respectively. So, each of the firm indicator variables could be used to separate out that firm's observations. This allows use of `wls` to easily estimate the model. In fact, this could be automated for an arbitrary number of firms by putting it into a loop.

```
loop foreach i du_*
    wls $i inv v k const
endloop
```

Notice that the wildcard `du_*` is used again. The results from this exercise were added to the model table and appears below:

```
Dependent variable: inv

                 (W)          (GE)          (GE)          (W)
             Pooled OLS    Pooled OLS       WLS           WLS

const           17.87**      -9.956        -9.956       -0.5094
               (7.024)      (23.63)       (31.37)       (8.015)

v             0.01519**    0.02655**     0.02655      0.05289**
             (0.006196)    (0.01172)     (0.01557)     (0.01571)

k               0.1436**     0.1517**      0.1517**    0.09241
              (0.01860)    (0.01936)     (0.02570)     (0.05610)

dconst                        9.447
                             (28.81)

dv                          0.02634
                           (0.03435)

dk                         -0.05929
                            (0.1169)

         n         40           40            20            20
Adj. R**2      0.7995       0.8025        0.6706        0.7144
       lnL     -177.3       -175.3        -93.31        -73.23

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

Notice that the coefficients are actually equivalent in the two sets of regressions. The GE equations are put side-by-side to ease the comparison. The standard errors differ, as expected. The Westinghouse coefficients estimated by WLS are also the same as the ones from the pooled model, though it is less obvious. Recall that the implied marginal effect of a change in k on average investment was estimated to be $0.1517 - 0.0593 = 0.0924$, which matches the directly estimated result in the last column.

Collecting the results for large N would be somewhat of a problem, but remember, up to 6 models can be added to a model table in **gretl**.

Next, we will estimate the model using SUR via the `system` command. To do this, some rearranging of the data is required. The `system` estimator is **gretl** handles many different cases, but the observations have to be ordered in a particular way in order for it to work. Since each equation in a system may be estimated separately, each firm's observations must be given unique names and the observations must be aligned by time period. The *grunfeld2.gdt* has the data for each firm stacked on top of one another, making it 40×3. We need it to be 20×6. Ordinarily with SUR this is not a problem. Recall that SUR is for large T, small N models and the equations may not even contain the same variables. In our case, the data were ordered for use as panel and not as a system. This is easy enough to rectify.

The easiest way to do this in **gretl** is to use matrices. The data series will be converted to a matrix, reshaped, and then reloaded as data series. Unique names will have to be given the new series and you'll have to keep up with what gets placed where. It is slightly clumsy, but easy enough to do.

```
1 open "@gretldir\data\poe\grunfeld2.gdt"
2 list X = inv v k
3 matrix dat = { X  }
4 matrix Y = mshape(dat,20,6)
5 colnames(Y,"ge_i w_i ge_v  w_v ge_k w_k ")
```

In line 1 the data are opened and in line 2 the variable list created. In line 3 the data series listed in `X` are converted to a matrix called `dat`. Then, the matrix `Y` is converted from 40×3 to 20×6. The `mshape(`*`matrix,rows,columns`*`)` command essentially takes whatever is in the matrix and converts it to the new dimension. Elements are read from `X` and written to the target in column-major order. Thus, `y=mshape(X,2,4)`

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{pmatrix} = y \tag{15.16}$$

Finally, the proper column names were reassigned using the `colnames` command.

Next, the matrix needs to reenter **gretl** as a dataset. Begin by creating a new, empty dataset that contains the proper number of observations (T=20) using the `nulldata` command. You must use the `--preserve` option, otherwise the contents of your matrix will be deleted when you create the empty dataset!

```
6  nulldata 20 --preserve
7  list v = null
8  scalar n = cols(Y)
9  loop for i=1..n
10    series v$i = Y[,i]
11 endloop
12 rename v1 inv_g
13 rename v2 inv_w
```

An empty list called v is created in line 7. A scalar n is created that contains the number of columns of Y and then a loop from 1 to n is initiated. Inside the loop is a single statement that will assign each column of the matrix to a separate series. The series name will begin with v and the column number will be appended. You will end up with variables v1, v2, to v6. The choice of variable names is not informative. You need to verify that the new variables correspond to the correct variables from the original dataset. To help keep the regressions straight, the dependent variables were renamed. A clever programmer could probably figure out how to do this automatically, but for now let's move on.

Before pushing on to estimation of the SUR, there is one more way to estimate the model equation by equation using least squares. This can be done within the same system framework as SUR.

It consists of a block of code that starts with the system name="Grunfeld" line. One advantage naming your system is that results are attached to it, stored into the session, and are available for further analysis. For instance, with a saved set of equations you can impose restrictions on a single equation in the model or impose restrictions across equations.

```
system name="Grunfeld"
   equation inv_g const v_g k_g
   equation inv_w const v_w k_w
end system
estimate "Grunfeld" method=ols
```

Following the system name, each equation is put on a separate line. Notice that each equation is identified using equation which is followed by the dependent variable and then the independent variables which includes a constant. Close the system block using the end system command. The system is then estimated using the line estimate "Grunfeld" method=ols. Executing this script yields

Equation system, Grunfeld
Estimator: Ordinary Least Squares

Equation 1: OLS, using observations 1–20

```
                    Dependent variable: inv_g

             Coefficient    Std. Error    t-ratio   p-value

   const    −9.95631       31.3742       −0.3173   0.7548
   v_g       0.0265512      0.0155661     1.7057   0.1063
   k_g       0.151694       0.0257041     5.9015   0.0000

Mean dependent var   102.2900   S.D. dependent var   48.58450
Sum squared resid    13216.59   S.E. of regression   27.88272


              Equation 2: OLS, using observations 1–20
                    Dependent variable: inv_w

             Coefficient    Std. Error    t-ratio   p-value

   const    −0.509390      8.01529       −0.0636   0.9501
   v_w       0.0528941     0.0157065      3.3677   0.0037
   k_w       0.0924065     0.0560990      1.6472   0.1179

Mean dependent var   42.89150   S.D. dependent var   19.11019
Sum squared resid    1773.234   S.E. of regression   10.21312


                  Cross-equation VCV for residuals
                  (correlations above the diagonal)

             660.83   (0.729)
             176.45    88.662    log determinant = 10.2203


Breusch–Pagan test for diagonal covariance matrix:
χ2(1) = 10.6278 [0.0011]
```

Naming the system has many advantages. First, the specified model is saved to a **session** and an icon is added to the session icon view as shown below in Figure 15.4. Clicking on the model icon named "Grunfeld" opens the dialog shown in Figure 15.5. Do not worry about the code that appears in the box. It is not editable and is generated by **gretl**. You do have some choice as to how the particular system is estimated and whether iterations should be performed. These choices appear in Figure 15.6. As you can see, you may choose `sur`, `ols`, `tsls`, `wls`, and others. To reestimated a model, choose an estimator, and click **OK**.

A test can be used to determine whether there is sufficient contemporaneous correlation. The test is simple to do from the standard output or you can rely on **gretl**'s automatic result. Recall from *POE4* that the test is based on the squared correlation computed from least squares estimation

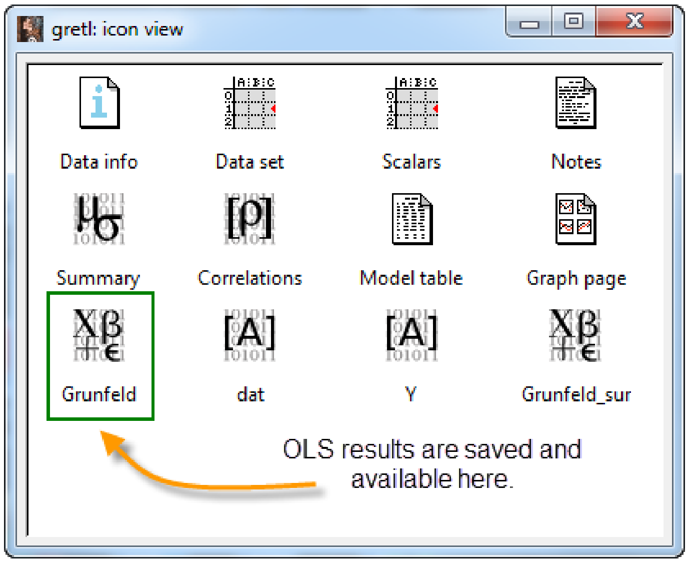


Figure 15.4: The session view window

of the system.

$$r^2_{g,w} = \frac{\hat{\sigma^2}_{g,w}}{\hat{\sigma^2}_g \hat{\sigma^2}_w} \tag{15.17}$$

A little caution is required here. The squared correlations are must be computed based on the residuals from the *least squares estimator*, not SUR. Since we've used the system command to estimate the model by OLS, the results above can be used directly.

The resulting cross-equation variance covariance for the residuals is

```
Cross-equation VCV for residuals
(correlations above the diagonal)

      777.45       (0.729)
      207.59        104.31
```

Then you compute

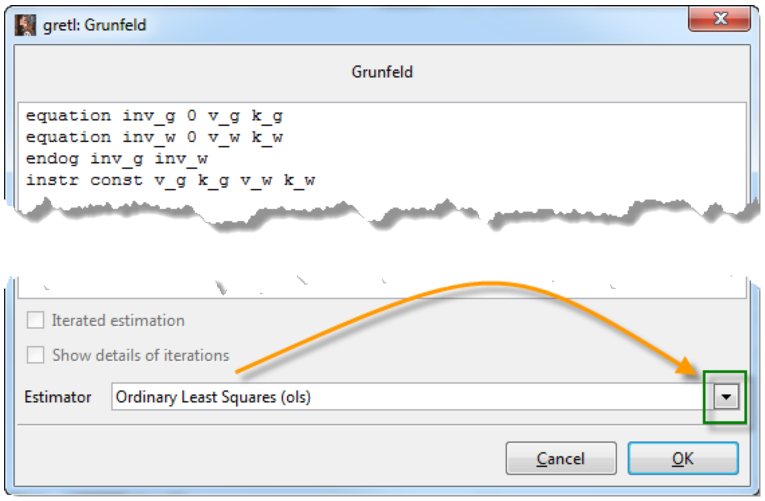
$$r^2_{ge,w} = \frac{207.59^2}{(777.45)(104.31)} = 0.729 \tag{15.18}$$

Gretl produces this number for you in the upper diagonal of the matrix and places it in parentheses. Using the given computation the test statistic is

$$LM = Tr^2_{g,w} \xrightarrow{d} \chi^2_{(1)} \tag{15.19}$$

provided the null hypothesis of no correlation is true. The arithmetic is $(20 * 0.729) = 14.58$.

Fortunately, **gretl** also produces this statistic as part of the standard output from system estimation by `method=ols`. It is referred to in the output as "Breusch–Pagan test for diagonal


gretl: Grunfeld

Grunfeld

```
equation inv_g 0 v_g k_g
equation inv_w 0 v_w k_w
endog inv_g inv_w
instr const v_g k_g v_w k_w
```

Figure 15.5: The system model choice dialog box. This system will be estimated by least squares. To reestimate using another model, click on the down arrow to reveal a set of estimator choices.

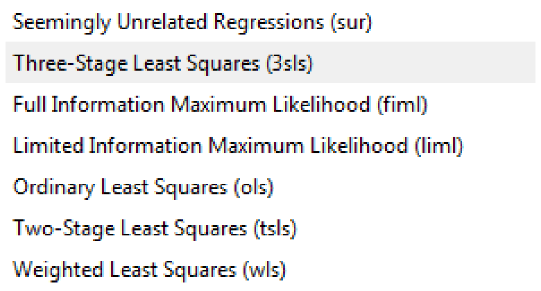

Seemingly Unrelated Regressions (sur)
Three-Stage Least Squares (3sls)
Full Information Maximum Likelihood (fiml)
Limited Information Maximum Likelihood (liml)
Ordinary Least Squares (ols)
Two-Stage Least Squares (tsls)
Weighted Least Squares (wls)

Figure 15.6: The estimator choices available from the system dialog.

covariance matrix" and its distributed $\chi^2(1)$ if there is no contemporaneous correlation among firms. The statistic is = 10.6278 with a p-value of 0.0011. The two firms appear to be contemporaneously correlated and SUR estimation may be more efficient.

To perform SUR, the only change is to rename the system (if desired) and to change `method=ols` to `method=sur`.

```
system name="Grunfeld_sur"
   equation inv_g const v_g k_g
   equation inv_w const v_w k_w
end system
estimate "Grunfeld_sur" method=sur
```

The results appear below:

Equation system, Seemingly Unrelated Regressions

Equation 1: SUR, using observations 1–20
Dependent variable: inv_g

	Coefficient	Std. Error	t-ratio	p-value
const	−27.7193	27.0328	−1.0254	0.3174
v_g	0.0383102	0.0132901	2.8826	0.0092
k_g	0.139036	0.0230356	6.0357	0.0000

Mean dependent var	102.2900	S.D. dependent var	48.58450
Sum squared resid	13788.38	S.E. of regression	26.25679

Equation 2: SUR, using observations 1–20
Dependent variable: inv_w

	Coefficient	Std. Error	t-ratio	p-value
const	−1.25199	6.95635	−0.1800	0.8590
v_w	0.0576298	0.0134110	4.2972	0.0004
k_w	0.0639781	0.0489010	1.3083	0.2056

Mean dependent var	42.89150	S.D. dependent var	19.11019
Sum squared resid	1801.301	S.E. of regression	9.490260

Once the system has been estimated, the `restrict` command can be used to impose the cross-equation restrictions on a system of equations that has been previously defined and named. The set of restrictions is starts with the keyword `restrict` and terminates with `end restrict`. Some additional details and examples of how to use the `restrict` command are given in section 6.1. Each restriction in the set is expressed as an equation. Put the linear combination of parameters to be tested on the left-hand-side of the equality and a numeric value on the right. Parameters are referenced using `b[i,j]` where i refers to the equation number in the system, and j the parameter number. So, to equate the intercepts in equations one and two use the statement

$$b[1,1] - b[2,1] = 0 \tag{15.20}$$

The full syntax for testing the full set of cross-equation restrictions

$$\beta_{1g} = \beta_{1w}, \quad \beta_{2g} = \beta_{2w}, \quad \beta_{3g} = \beta_{3w} \tag{15.21}$$

on equations (15.13) and (15.14) is shown

```
restrict "Grunfeld_sur"
b[1,1]-b[2,1]=0
```

```
b[1,2]-b[2,2]=0
b[1,3]-b[2,3]=0
end restrict
estimate "Grunfeld_sur" method=sur --geomean
```

Gretl estimates the two equation SUR subject to the restrictions.

Equation system, Grunfeld_sur
Estimator: Seemingly Unrelated Regressions

Equation 1: SUR, using observations 1–20
Dependent variable: inv_g

	Coefficient	Std. Error	t-ratio	p-value
const	19.1578	2.54265	7.5346	0.0000
v_g	0.0226805	0.00502650	4.5122	0.0002
k_g	0.109053	0.0190478	5.7252	0.0000

Mean dependent var	102.2900	S.D. dependent var	48.58450
Sum squared resid	15923.76	S.E. of regression	28.21681

Equation 2: SUR, using observations 1–20
Dependent variable: inv_w

	Coefficient	Std. Error	t-ratio	p-value
const	19.1578	2.54265	7.5346	0.0000
v_w	0.0226805	0.00502650	4.5122	0.0002
k_w	0.109053	0.0190478	5.7252	0.0000

Notice that the intercept and slopes are equal across firms now. The restrictions are imposed. It also computes an F-statistic of the null hypothesis that the restrictions are true versus the alternative that at least one of them is not true. It returns the computed F-statistic and its p-value. A p-value less than the desired level of significance leads to a rejection of the hypothesis.

The **gretl** output from this test procedure is

```
F test for the specified restrictions:
  F(3,34) = 3.43793 [0.0275]
```

which does not match the result in the text.[2] At the 5% level of significance, the equality of the two equations is rejected.

[2]Not sure why. To be determined.

15.8 Script

```
set echo off
open "@gretldir\data\poe\nls_panel.gdt"
# pooled least squares
list xvars = const educ exper exper2 tenure tenure2 south union black
panel lwage xvars --pooled --robust

# fixed effects
xvars -= const
panel lwage xvars const --fixed-effects

# fixed effects and lsdv
genr unitdum
xvars -= educ black
ols lwage xvars du_*
panel lwage xvars const --fixed-effects

# fe, re, between, and pooled comparison
open "@gretldir\data\poe\nls_panel.gdt"
list xvars = educ exper exper2 tenure tenure2 south union black
panel lwage xvars const --fixed-effects
modeltab add
panel lwage xvars const --random-effects
modeltab add
panel lwage xvars const --between
modeltab add
panel lwage xvars const --pooled --robust
modeltab add
modeltab show
modeltab clear

# Grunfeld example -- ols
open "@gretldir\data\poe\grunfeld2.gdt"
list X = const v k
ols inv X
modeltab add
genr unitdum
list dZ = null
loop foreach i X
series d$i = du_2 * $i
   list dZ = dZ d$i
endloop
ols inv X dZ
modeltab add
modeltab show

# using wls to estimate each equation separately
wls du_1 inv v k const
modeltab add
```

```
wls du_2 inv v k const
modeltab add
modeltab show

# repeat using wls with a loop
loop foreach i du_*
    wls $i inv v k const
endloop

# sur--reshaping the data
open "@gretldir\data\poe\grunfeld2.gdt"
list X = inv v k
matrix dat = { X  }
matrix Y = mshape(dat,20,6)
colnames(Y,"ge_i w_i ge_v  w_v ge_k w_k ")

nulldata 20 --preserve
list v = null
scalar n = cols(Y)
loop for i=1..n
   series v$i = Y[,i]
endloop

# rename the variables to improve output
rename v1 inv_g
rename v2 inv_w
rename v3 v_g
rename v4 v_w
rename v5 k_g
rename v6 k_w
setinfo inv_g -d "Investment GE" -n ""
setinfo inv_w -d "Investment Westinghouse" -n ""

# actual system estimation -- ols
system name="Grunfeld"
   equation inv_g const v_g k_g
   equation inv_w const v_w k_w
end system
estimate "Grunfeld" method=ols

# actual system estimation -- sur
system name="Grunfeld_sur"
   equation inv_g const v_g k_g
   equation inv_w const v_w k_w
end system
estimate "Grunfeld_sur" method=sur

# restricting sur
restrict "Grunfeld_sur"
b[1,1]-b[2,1]=0
b[1,2]-b[2,2]=0
```

```
b[1,3]-b[2,3]=0
end restrict
estimate "Grunfeld_sur" method=sur --geomean
```

Chapter 16

Qualitative and Limited Dependent Variable Models

16.1 Probit

There are many things in economics that cannot be meaningfully quantified. How you vote in an election, whether you go to graduate school, whether you work for pay, or what college major you choose has no natural way of being quantified. Each of these expresses a *quality* or *condition* that you possess. Models of how these decisions are determined by other variables are called **qualitative choice** or **qualitative variable** models.

In a **binary choice** model, the decision you wish to model has only two possible outcomes. You assign artificial numbers to each outcome so that you can do further analysis. In a binary choice model it is conventional to assign '1' to the variable if it possesses a particular quality or if a condition exists and '0' otherwise. Thus, your dependent variable is

$$y_i = \begin{cases} 1 & \text{if individual i has the quality} \\ 0 & \text{if not.} \end{cases}$$

The **probit** statistical model expresses the probability p that $y_i = 1$ as a function of your independent variables.

$$P[(y_i|x_{i2}, x_{i3}) = 1] = \Phi(\beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3}) \tag{16.1}$$

where Φ is the cumulative normal probability distribution (cdf). The argument inside Φ is linear in the parameters and is called the **index function**. Φ maps values of the index function to the 0 to 1 interval. Estimating this model using maximum likelihood is very simple since the MLE of the probit model is already programmed into **gretl**.

The syntax for a script is the same as for linear regression except you use the `probit` command in place of `ols`. The following script estimates how the difference in travel time between *bus* and *auto* affects the probability of driving a car. The dependent variable (*auto*) is equal to 1 if travel is by car, and *dtime* is (bus time - auto time).

```
open "@gretldir\data\poe\transport.gdt"
probit auto const dtime
scalar i_20 = $coeff(const)+$coeff(dtime)*20
scalar p_30 = cnorm($coeff(const)+$coeff(dtime)*30)
```

The third line computes the predicted value of the index ($\beta_1 + \beta_2 dtime$) when $dtime = 20$ using the estimates from the probit MLE. The last line computes the estimated probability of driving, given that it takes 30 minutes longer to ride the bus. This computation requires `cnorm`, which computes the cumulative normal cdf, $\Phi()$.

The results are:

Probit estimates using the 21 observations 1–21
Dependent variable: auto

	Coefficient	Std. Error	z-stat	Slope*
const	−0.0644342	0.399244	−0.1614	.
dtime	0.0299989	0.0102867	2.9163	0.0119068

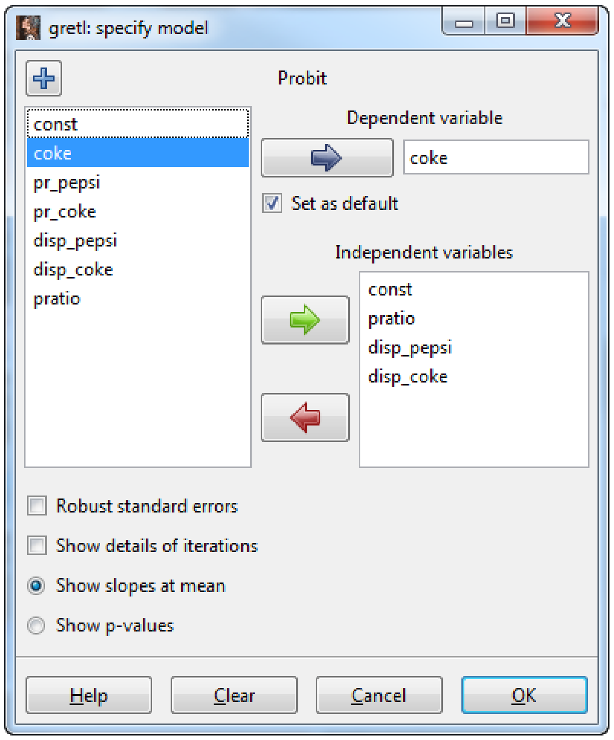
Mean dependent var	0.476190	S.D. dependent var	0.396907
McFadden R^2	0.575761	Adjusted R^2	0.438136
Log-likelihood	−6.165158	Akaike criterion	16.33032
Schwarz criterion	18.41936	Hannan–Quinn	16.78369

*Evaluated at the mean

Number of cases ‘correctly predicted’ = 19 (90.5 percent)
Likelihood ratio test: $\chi^2(1) = 16.734$ [0.0000]

```
i_20 =  0.535545
p_30 =  0.798292
```

Of course, you can also access the probit estimator from the pull-down menus using `Model>Nonlinear models>Probit>Binary`. The dialog box (Figure 16.1) looks very similar to the one for linear regression, except it gives you some additional options e.g., to view the details of the iterations.



const
coke
pr_pepsi
pr_coke
disp_pepsi
disp_coke
pratio

Dependent variable: coke

[✓] Set as default

Independent variables: const, pratio, disp_pepsi, disp_coke

[ ] Robust standard errors
[ ] Show details of iterations
(●) Show slopes at mean
( ) Show p-values

Figure 16.1: Use `Model>Nonlinear models>Probit>Binary` to open the Probit model's dialog box.

Several other statistics are computed. They include a measure of fit (McFadden's pseudo-R^2), the value of the standard normal pdf $\phi(\hat{\beta}^T\bar{x})$ at mean of independent variables, and a test statistic for the null hypothesis that the coefficients on the independent variables (but not the constant) are jointly zero; this corresponds to the overall F-statistic of regression significance in chapter 6.

16.1.1 Marginal Effects and Average Marginal Effects

The marginal effect of a change in x_{ij} on P_i is

$$\frac{\partial P_i}{\partial x_{ij}} = \phi(\beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3})\beta_j \tag{16.2}$$

where $\phi()$ is the standard normal probability density. That means that the marginal effect depends on all of the parameters of the model as well as the values of the variables themselves. In the travel example, suppose we want to estimate the marginal effect of increasing public transportation time. Given that travel via public transport currently takes 20 ($dtime$=2) minutes longer than auto, the marginal effect would be

$$\frac{\partial P_i}{\partial dtime_i} = \phi(\hat{\beta}_1 + \hat{\beta}_2 dtime_i) = \phi(-0.0644 + 0.3000 \times 2)(0.3000) = 0.1037 \tag{16.3}$$

This computation is easily done in a script:

```
open "@gretldir\data\poe\transport.gdt"
probit auto const dtime
scalar i_20 = $coeff(const)+$coeff(dtime)*2
scalar d_20 = dnorm(i_20)*$coeff(dtime)
printf "\nThe value of the index for dtime=20 is %8.4f\n\
The probability of choosing auto is %8.4f \n", i_20, d_20
```

which produces

```
The value of the index for dtime=20 is 0.5355
The probability of choosing auto is 0.1037
```

Rather than evaluate marginal effects at specific points in the data, some authors report **average marginal effects**. The average marginal effect of a change in x_{ij} on P_i is

$$\widehat{AME}_j = \frac{1}{N}\sum_{i=1}^{N}\phi(\hat{\beta}_1 + \hat{\beta}_2 x_{i2} + \hat{\beta}_3 x_{i3})\hat{\beta}_j \tag{16.4}$$

With these, you compute the marginal effect at each point in your dataset and average them.

There is one other way that people use to estimate marginal effects, and that is to evaluate the marginal effect at the means of the data. That would be

$$\widehat{ME}_j = \phi(\hat{\beta}_1 + \hat{\beta}_2 \bar{x}_2 + \hat{\beta}_3 \bar{x}_3)\hat{\beta}_j \tag{16.5}$$

These are computed and reported by **gretl** and labeled 'slope' in the output. The biggest disadvantage of using these is that the average values of the variables may not be representative of anyone in your sample. This is especially true if one or more of the variables is an indicator. For this reason, I generally favor the use of the AME, unless there are specific cases that I want to consider. You can actually get a pretty good idea of what the (average) marginal effects will be by looking at the estimated slopes from a linear probability model.

Below is a simple script to compute the average marginal effects (AME) for the travel example. The model has only one regressor and a constant. We will compute the AME for an increase in travel time.

```
series me = dnorm($coeff(const)+$coeff(dtime)*dtime)*$coeff(dtime)
scalar amf = mean(me)
summary me --simple
```

The first line computes a new variable, `me`, that holds the marginal effect for each individual. The mean function, which takes the average of a series) finds the average marginal effect. The simple summary statistis also reveal the AME and give us an idea of how variable it is (minimum, maximum, and standard deviation). The result is:

```
Summary statistics, using the observations 1 - 21
for the variable 'me' (21 valid observations)

  Mean                        0.048407
  Minimum                    0.0024738
  Maximum                      0.11526
  Standard deviation          0.036457
```

The average marginal effect in the sample is 0.0484. The smallest is 0.0024738 and the largest 0.11526. That is a fairly large range, actually.

To facilitate the computation of AME, I have written a function that will compute them for an arbitrary probit model. The function is called `ame` and it needs two pieces of information to compute the marginal effects. First, it needs the parameters from estimation of probit. Then, it needs the list of explanatory variables that are in the model. The function will print the average marginal effects and output a $N \times K$ matrix that contains each of the marginal effects for every observation and variable. This matrix is handy to have for further computations.

```
1 function matrix ame(matrix *param, list x)
2     matrix p = lincomb(x, param)
3     matrix me_matrix = dnorm(p)*param'
4     matrix amfx = meanc(me_matrix)
5     printf "\nThe average marginal effects are %8.4f \n", amfx
6     return me_matrix
7 end function
```

The function is quite simple, but uses what is referred to as a **pointer**. The `*param` is the pointer. This is technically not necessary, but saves some computer overhead. It also requires special handling when the function is called. More on that detail in a moment.

The second line uses the `lincomb` function, which takes a linear combination of its arguments. The first argument should be a **list** that contains the desired series, the second argument is a vector of coefficients to use with the series. The result can be a series, or in this case, a matrix. So for instance, suppose X is $T \times K$ and contains variables and β is $K \times 1$ parameters. The linear combination $X\beta$ is $T \times 1$. Line 3 computes the matrix that contains all of the marginal effects. The `meanc` function in line 4 computes the column means of the matrix (AME), which gets printed in line 5. The entire matrix of marginal effects is returned when the function is called.

Once the function is loaded (highlight it and run it) it is ready to be used. Create the variable list, estimate the probit model, and save the coefficients using `matrix coef = $coeff`. Given the variable list and the parameter estimates, you can call the function as in line 4 of the script below.

```
1 list x = const dtime
2 probit auto x
```

```
3 matrix coef = $coeff
4 ame(&coef, x)
```

Since I used a pointer to identify the parameter vector in the function (`matrix *param`), you have to use the ampersand (`&`) in front of the parameter matrix being passed to the function, i.e., `ame(&coef, x)`. Thus, pointers require a pair of markers, `*` and `&`, when used. The `*` tells **gretl** to use the memory address of what follows rather than make a copy of the object to pass through the function. The `&` tells **gretl** to go get that object from the memory address when called. Basically, the use of pointers means that copies of objects do not have to be made, and it also means that whatever is getting passed around in this fashion can be modified in the process. That may not sound like a great idea, but it can make your programs more modular. See section 10.4 of the **Gretl** Users Guide for more details.

The use of `ame(&coef, x)` in line 4 will print the AME to the screen. If you want to save the matrix output from the function, use `matrix me_probit = ame(&coef, x)` and the result will be saved to `me_probit`.

The result for the travel time example is:

```
The average marginal effects are  -0.0104  0.0484
```

The average marginal effect of a 10 minute ($dtime = 1$) increase in travel time is 0.0484.

16.1.2 Standard Errors and Confidence Intervals for Marginal Effects

Obtaining confidence intervals for the marginal effects (and the AME) is relatively straightforward as well. To estimate the standard error of the marginal effect, we resort to the Delta method. This method of finding the variance of functions of parameters was discussed in section 5.3.2. You may want to take a look at this section again (page 99), before proceeding.

Using the Delta method means taking analytic or numerical derivatives of the marginal effect or AME to be used in the computation of the standard error of the AME. The analytic derivatives are not that hard to take, but why bother when numerical ones are available. This is the approach taken in commercial software that includes the ability to estimate nonlinear combinations of parameters and their standard errors.

The function in **gretl** that takes numeric derivatives is `fdjac`, which stands for **first difference Jacobian**. To use the Delta method, you want to take the partial derivatives of a function with respect to the model's parameters. Not surprisingly, the `fdjac` function requires two arguments: a function and a vector of parameters. To illustrate its use, consider the new function for marginal effects below.

```
function matrix me1(matrix *param, list x, scalar *q)
    matrix p = lincomb(x, param)
    return dnorm(p)*param[q]
end function
```

It is very similar to the `ame` function, except in this instance we have added an extra scalar input. The `scalar *q` will be used to locate the desired parameter in the model. So, if we want to compute the marginal effect for `x2`, and `list x = const x2 x3`, then `q` will be set to 2; `x2` is the second variable in the list. Rather than return all of the marginal effects, this function only returns the set for a given variable. This makes using `fdjac` easier to use and to explain.

Once the function is defined, you are ready to use `fdjac` in the Delta method.

```
1  open "@gretldir\data\poe\transport.gdt"
2  list x = const dtime
3  probit auto x
4  matrix coef = $coeff
5  matrix covmat = $vcv
6  scalar q = 2
7  series mfx = me1(&coef, x, &q)
8  matrix amfx = mean(mfx)
9  matrix jac = fdjac(coef, me1(&coef, x, &q))
10 matrix mjac = meanc(jac)
11 matrix variance = qform(mjac,covmat)
12 matrix se = sqrt(variance)
13 printf "\nThe average marginal effect of dtime = %6.4f with\
14  standard error %6.4f \n", amfx, se
15
16 scalar ub = amfx + critical(z,0.025)*se
17 scalar lb = amfx + critical(z,0.025)*se
18 printf "\nThe 95%% confidence interval for the AME is (%6.4f, %6.4f) \n",lb,ub
```

The first six lines of the script are standard. Open the data, create the variable `list`, estimate the model by `probit`, create the matrix to hold the coefficients, create the matrix to hold the estimated covariance matrix, and choose the scalar `q` that identifies the location of the desired marginal effect in the variable list, which in this case is the second one.

Lines 7 and 8 get the sample marginal effects for the second coefficient and their mean. The next line takes the derivative of the marginal effect function using `fdjac`. Since we used pointers in the function, the ampersand needs to precede the coefficient and scalar inputs.

As shown in the appendix of chapter 16 in *POE4*, taking the average of the derivatives is what is needed for the variance calculation. In line 11 the quadratic form is taken. Basically, `qform(x,A)` computes xAx^T; this is exactly what is involved in the variance expression in equation (5.10). The square root provides the standard error of the AME. The average marginal effect = 0.0484 with standard error 0.0034. The 95% confidence interval for the AME is (0.0413, 0.0556).

Finally, we can automate the evaluation of marginal effects at certain points. Recall that we computed the marginal effect for someone who is currently taking 20 (*dtime*=2) minutes longer to travel by bus than auto. The marginal effect was easy enough to compute, but now we want to add a confidence interval. The function will be modified slightly to accept a row vector that contains the values of the explanatory variables at the desired evaluation point. The function is:

```
1 function matrix me_at(matrix *param, matrix *xx, scalar *q)
2     scalar p = xx*param
3     return dnorm(p)*param[q]
4 end function
```

Now instead of using `lincomb`, we use a matrix calculation in line 2. This requires a matrix input in the function definition in line 1, which we have marked with a pointer.

The script to execute this is

```
1  open "@gretldir\data\poe\transport.gdt"
2  list x = const dtime
3  probit auto x
4  matrix coef = $coeff
5  matrix covmat = $vcv
6  scalar q = 2
7  matrix xx = { 1, 2 }
8  matrix mfx = me_at(&coef, &xx, &q)
9  matrix jac = fdjac(coef, me_at(&coef, &xx, &q))
10 matrix variance = qform(jac,covmat)
11 matrix se = sqrt(variance)
12 printf "\nThe marginal effect of dtime when dtime=2 is %6.4f with \
13 standard error %6.4f \n", mfx, se
14
15 scalar ub = mfx + critical(z,0.025)*se
16 scalar lb = mfx + critical(z,0.025)*se
17 printf "\nThe 95%% confidence interval the ME with dtime=2 is\
18  (%6.4f, %6.4f) \n", lb, ub
```

Lines 3-5 set the inputs to the function. Line 5 is a row vector that has a 1 in the first element for the constant and 2 in the second for *dtime*. This is sent to the `me_at` function to get the marginal effect and the `fdjac` evaluates the derivative with respect to coefficients, `coef`. Variance is computed using `qform` and then the standard error is taken from this. The result is:

```
The marginal effect of dtime when dtime=2 is 0.1037 with standard error 0.0326
The 95% confidence interval the ME with dtime=2 is (0.0397, 0.1677).
```

A perfect match for the result in *POE4*. Excellent!

Finally, the predicted probability that $auto = 1$ given a commuting time difference of 30 minutes is calculated and a confidence interval obtained using the Delta method. The function is very similar to the last one and it was used as a template.

```
function matrix mep(matrix *param, matrix *xx, scalar *q)
    scalar p = xx*param
    return cnorm(p)
end function
```

Notice that `cnorm` is used to get the cumulative probability.

To use the function,

```
scalar q = 2
matrix coef = $coeff
matrix xx = { 1, 3 }
matrix mp = mep(&coef, &xx, &q)
matrix jac_3 = fdjac(coef, mep(&coef, &xx, &q))
matrix variance = qform(jac_3,covmat)
matrix se = sqrt(variance)
printf "\nThe probability of driving when dtime=3 is %6.4f with standard \
error %6.4f \n", mp, se

scalar ub = mp + critical(z,0.025)*se
scalar lb = mp - critical(z,0.025)*se
printf "\nThe 95%% confidence interval the Prob with dtime=3 is \
(%6.4f, %6.4f) \n",lb,ub
```

The probability of driving when *dtime*=3 is 0.7983 with standard error 0.1425. The 95% confidence interval with *dtime*=3 is (0.5189, 1.0777). Obviously, the upper bound is not feasible since probabilities cannot exceed 1.

16.1.3 Hypothesis Tests

Based on the soft drink example explored in section 8.7, suppose you want to test the hypothesis that the Coke and Pepsi displays have an equal but opposite effect on the probability of buying Coke. If a store has both displays, the net effect on Coke purchases is zero.

The model is:

$$\Pr(Coke_i = 1) = \Phi(\beta_1 + \beta_2 pratio + \beta_3 disp\_coke + \beta_4 disp\_pepsi) \tag{16.6}$$

The null and alternative hypotheses are:

$$H_0: \beta_3 - \beta_4 = 0 \quad H_1: \beta_3 - \beta_4 \neq 0 \tag{16.7}$$

The simplest thing to do is use the `restrict` statement as shown below:

```
open "@gretldir\data\poe\coke.gdt"
list x = pratio disp_coke disp_pepsi const
probit coke x
restrict
   b[3]+b[4]=0
end restrict
```

This works exactly as it did in linear regression. The outcome in **gretl** is:

```
Restriction:
 b[disp_pepsi] + b[disp_coke] = 0

Test statistic: chi^2(1) = 5.40401, with p-value = 0.0200905
```

The p-value is less than 5% and the hypothesis is rejected at this level.

Another hypothesis to consider is that the displays have no effect. The null and alternative hypotheses are:

$$H_0: \beta_3 = 0 \text{ and } \beta_4 = 0 \quad H_1: \beta_3 \neq 0 \text{ or } \beta_4 \neq 0 \tag{16.8}$$

The **gretl** code is

```
open "@gretldir\data\poe\coke.gdt"
list x = pratio disp_coke disp_pepsi const
probit coke x
restrict
   b[3]=0
   b[4]=0
end restrict
```

This statistic will have an χ^2_2 distribution if the null hypothesis is true. The outcome in **gretl** is:

```
Restriction set
 1: b[disp_pepsi] = 0
 2: b[disp_coke] = 0

Test statistic: chi^2(2) = 19.4594, with p-value = 5.9489e-005
```

Again, this hypothesis is rejected at any reasonable level of significance.

Since probit and logit are estimated via maximum likelihood, you can also perform a **likelihood ratio test**. The likelihood ratio is

$$LR = 2(\ln L_U - \ln L_R) \sim \chi^2_J \tag{16.9}$$

if the null is true. The parameter J is the degrees of freedom for the χ^2 and it equals the number of hypotheses you are testing, in this case 2. It has the same approximate distribution as the preceding test. L_U and L_R are the maximized log-likelihoods from an unrestricted and restricted model, respectively. The key is to estimate restricted and unrestricted models and collect the log-likelihood from each. For the first hypothesis, the restriction implies that

$$P_{coke} = \Phi(\beta_1 + \beta_2 pratio + \beta_4(disp\_pepsi - disp\_coke) \tag{16.10}$$

To estimate the restricted model, just form the new variable ($disp\_pepsi - disp\_coke$) and use it in the model. The script to compute and evaluate the LR is:

```
scalar c_p = disp_pepsi-disp_coke
probit coke x                # unrestricted model
scalar llu = $lnl
probit coke const pratio c_p # restricted model
scalar llr = $lnl
scalar lr = 2*(llu-llr)
pvalue X 1 lr
```

The result is

```
Generated scalar lr = 5.42183

Chi-square(1): area to the right of 5.42183 = 0.0198865
(to the left: 0.980114)
```

The statistic is 5.42, which is very close to the value from the previous test of this hypothesis.

16.2 Logit

The logit model is very similar to probit. Rather than the probability of an event being described by a normal distribution, it is modeled using a logistic distribution. The logistic and normal have very similar shapes and the outcomes from the logit estimation are usually very similar to those in probit. The probability that individual i chooses the alternative is

$$P_i = F(z_i) = \Lambda(z_i) = \frac{1}{1+e^{-z_i}} \tag{16.11}$$

$$z_i = \sum_{j=1}^{k} x_{ij}\beta_j \tag{16.12}$$

In logit the probability is modeled using $\Lambda(z_i)$ rather than $\Phi(z_i)$ as in the probit model.

Below we estimate the probability of purchasing Coca-Cola rather than Pepsi using probit, logit, and the linear probability model. The data are contained in *coke.gdt* and consist of 1140 individuals who purchased either Coke or Pepsi. The **gretl** script to estimate the models and put the results in a model table is

```
open "@gretldir\data\poe\coke.gdt"
list x = pratio disp_coke disp_pepsi const
probit coke x --quiet
modeltab add
logit coke x --quiet
modeltab add
ols coke x --robust
modeltab add
modeltab show
```

The result obtained is:

Dependent variable: coke

	(1) Probit	(2) Logit	(3) OLS
const	1.11**	1.92**	0.890**
	(5.83)	(5.90)	(13.6)
pratio	−1.15**	−2.00**	−0.401**
	(−6.34)	(−6.34)	(−6.60)
disp_pepsi	−0.447**	−0.731**	−0.166**
	(−4.41)	(−4.36)	(−4.81)
disp_coke	0.217**	0.352**	0.0772**
	(2.25)	(2.22)	(2.27)
n	1140	1140	1140
R^2	0.093	0.095	0.120
ℓ	−711	−709	−748

t-statistics in parentheses
** indicates significance at the 5 percent level
For logit and probit, R^2 is McFadden's pseudo-R^2

The signs and the t-ratios are approximately equal across the model. In logit and probit, the coefficient's sign indicates the sign of the marginal effect. Coefficient magnitudes differ only because of the implicit differences in how the coefficients are normalized. Although it is not obvious, there is an approximate relationship among the 'slope' coefficients of the three sets of estimates.

$$\tilde{\gamma}_{Logit} \cong 4\hat{\beta}_{LPM}$$
$$\tilde{\beta}_{Probit} \cong 2.5\hat{\beta}_{LPM}$$
$$\tilde{\gamma}_{Logit} \cong 1.6\hat{\beta}_{Probit}$$

So, $4(-0.4009) = -1.6036$, which is fairly close to the estimate -1.996 for the `pratio` coefficient in the logit column. More importantly, there are even closer similarities between the marginal effects implies by logit and probit. Their averages (AME) are usually very close to the corresponding coefficient in the linear probability model.

To get the AMEs in the logit model, we need a function to evaluate the logistic probability density, which is

$$\lambda(z_i) = \frac{e^{-z_i}}{(1+e^{-z_i})^2}, \quad -\infty < z_i < \infty \tag{16.13}$$

My function is

```
function matrix dlogist(matrix *param, list x)
    matrix p = lincomb(x, param)
    matrix d = exp(-p)./(1.+exp(-p)).^2
    return d
end function
```

It uses the 'dot' operators for division, multiplication, and exponentiation. These work element-by-element. Vectorizing this as done here may or may not be a good idea, but the syntax is straightforward. A more versatile approach would probably be to loop over the available observations.

Now we need a function that computes and average the marginal effects. Minor modification of the `ame` function that was used for the probit model yields the average marginal effect for the logit function `ame_l` below:

```
function matrix ame_l(matrix *param, list x)
    matrix p = lincomb(x, param)
    matrix me_matrix = dlogist(&param,x)*param'
```

```
4     matrix amfx = meanc(me_matrix)
5     printf "\nThe average marginal effects are %8.4f \n", amfx
6     return me_matrix
7 end function
```

The only change to the original probit `ame` function comes in line 3 where the `dlogist` function replaces `dnorm`.

Estimating the model by logit and getting the AME is done using:

```
list x = const pratio disp_coke disp_pepsi
logit coke x
matrix coef = $coeff
ame_l(&coef,x)
```

which produces

```
The average marginal effects are   0.4175 -0.4333  0.0763 -0.1587
```

The average marginal effect for a increase in the price ratio is -0.4333. That compares to -0.4097 in probit and -0.4009 in the linear probability model. It would certainly be easy at this point to compute standard errors for these marginal effects, but we will save that as an exercise.

The models can also be compared based on predictions. **Gretl** produces a table in the standard probit and logit outputs that facilitates this. The table is 2×2 and compares predictions from the model to actual choices. The table for the beverage choice model is:

```
Number of cases 'correctly predicted' = 754 (66.1%)
f(beta'x) at mean of independent vars = 0.394
Likelihood ratio test: Chi-square(3) = 145.823 [0.0000]

           Predicted
             0     1
  Actual 0  507   123
         1  263   247
```

The table reveals that with probit, of the $(507+123) = 630$ consumers that chose Pepsi (Pepsi=0), the model predicted 507 of these correctly (80.48% correct for Pepsi). It predicted $247/(263 + 247) = 247/510 = 48.43\%$ correct for Coke. The overall percentage that was correctly predicted is $754/1140 = 66.1\%$. The table for logit is exactly the same, so there is no reason to prefer one over the other for their predictive accuracy.

In fact, look at the correlations between the predictions of the three estimators:

```
Correlation Coefficients for model predictions, using the observations 1 - 1140
5\% critical value (two-tailed) = 0.0581 for n = 1140

        probit        logit          ols
        1.0000        0.9996        0.9950  probit
                      1.0000        0.9924  logit
                                    1.0000  ols
```

As you can see, these are VERY highly correlated, all over 0.99 and significant at 5%.

16.3 Multinomial Logit

Starting with version 1.8.1, **Gretl** includes a routine to estimate multinomial logit (MNL) using maximum likelihood. In versions before 1.8.1 the alternatives were either (1) use **gretl**'s maximum likelihood module to estimate your own or (2) use another piece of software! In this section we'll estimate the multinomial logit model using the native **gretl** function and I'll relegate the other methods to a separate (optional) section 16.3.1. The other methods serve as good examples of how to use **gretl**'s scripting language and how to use it in conjunction with **R**.

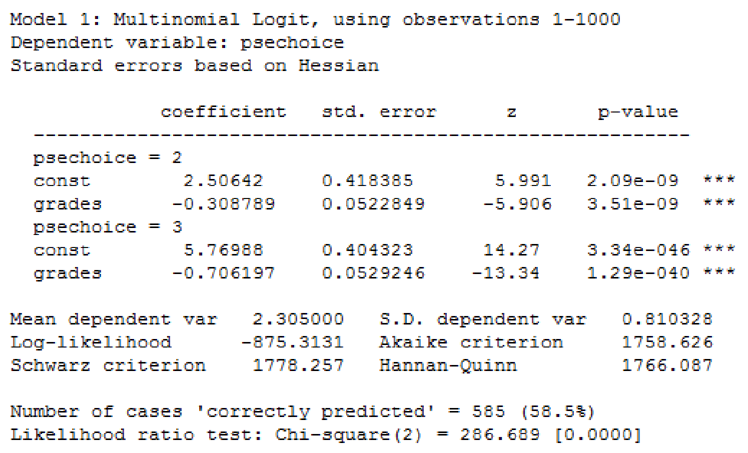
In this model the dependent variable is categorical and is coded in the following way. A student graduating from high school chooses among three alternatives: attend no college `psechoice=1`, enroll in a 2-year college `psechoice=2`, or enroll in a 4-year college `psechoice=3`. The explanatory variable is `grades`, which is an index ranging from 1.0 (highest level, A+ grade) to 13.0 (lowest level, F grade) and represents combined performance in English, Math and Social Studies. For this example, the choices are treated as being unordered. There are 1000 observations.

To estimate the model of school choice as a function of grades and a constant open the *nels_small.gdt* dataset and use the `logit` command with the `--multinomial` option as shown:

```
open "c:\Program Files\gretl\data\poe\nels_small.gdt"
logit psechoice const grades --multinomial
```

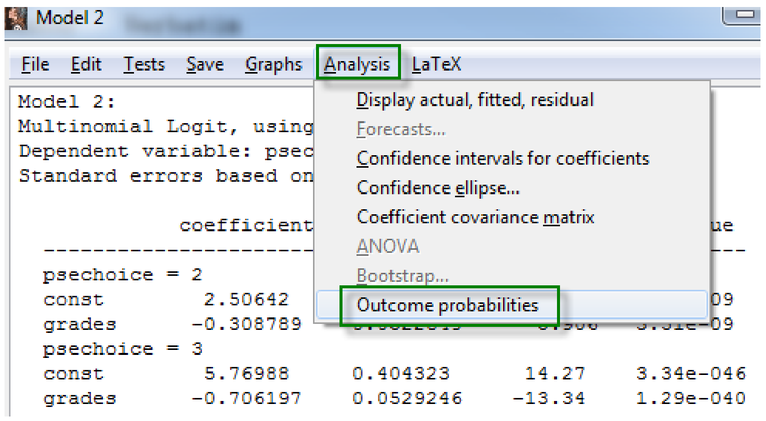
This yields the output shown in Figure 16.2:

The coefficients appear in sets. The first set are the coefficients that go with `psechoice=2` and the second set go with `psechoice=3`; this implies that **gretl** chose `psechoice=1` used as the base.



```
Model 1: Multinomial Logit, using observations 1-1000
Dependent variable: psechoice
Standard errors based on Hessian

             coefficient   std. error      z       p-value
  ---------------------------------------------------------
  psechoice = 2
  const        2.50642     0.418385      5.991    2.09e-09  ***
  grades      -0.308789    0.0522849    -5.906    3.51e-09  ***
  psechoice = 3
  const        5.76988     0.404323     14.27     3.34e-046 ***
  grades      -0.706197    0.0529246   -13.34     1.29e-040 ***

Mean dependent var   2.305000   S.D. dependent var   0.810328
Log-likelihood      -875.3131   Akaike criterion     1758.626
Schwarz criterion    1778.257   Hannan-Quinn         1766.087

Number of cases 'correctly predicted' = 585 (58.5%)
Likelihood ratio test: Chi-square(2) = 286.689 [0.0000]
```

Figure 16.2: Output from multinomial logit



```
Model 2:
Multinomial Logit, using
Dependent variable: psec
Standard errors based on

             coefficient
  ----------------------
  psechoice = 2
  const        2.50642
  grades      -0.308789
  psechoice = 3
  const        5.76988     0.404323     14.27     3.34e-046
  grades      -0.706197    0.0529246   -13.34     1.29e-040
```

Figure 16.3: You can obtain the outcome probabilities from the multinomial logit model window. These are also available after estimation in the `$mnlprobs` accessor.

The probability of choosing an alternative in multinomial logit is

$$p_{i1} = \frac{1}{1+\sum_{j=2}^{J}\exp(\beta_{1j}+\beta_{2j}x_{i2}+\cdots+\beta_{kj}x_{ik})} \qquad j=1 \tag{16.14}$$

$$p_{ij} = \frac{\exp(\beta_{1j}+\beta_{2j}x_{i2}+\cdots+\beta_{kj}x_{ik})}{1+\sum_{j=2}^{J}\exp(\beta_{1j}+\beta_{2j}x_{i2}+\cdots+\beta_{kj}x_{ik})} \qquad j\neq 1 \tag{16.15}$$

Obtaining the probabilities is simple. If you estimate the model via the GUI (`Model>Nonlinear models>Logit>Multinomial`) then you will have an option in the model window to produce the predicted probabilities for each case in the sample. In Figure 16.3 you will see that `Analysis>Outcome probabilities` can be selected. The first few of these are shown:

```
    Estimated outcome probabilities for psechoice
```

```
          1        2        3
  1   0.4408   0.3274   0.2319
  2   0.3511   0.3308   0.3181
  3   0.2539   0.3148   0.4313
  4   0.2539   0.3148   0.4313
  5   0.2539   0.3148   0.4313
  ....
1000   0.0339   0.1351   0.8310
```

A script can be written to obtain predicted probabilities that shows off a few more tricks. The proposed function is called mlogitprob and the script for it is:

```
1  function list mlogitprob(series y, list x, matrix theta)
2      list probs = null
3      matrix X = { x }
4      scalar j = max(y)
5      scalar k = cols(X)
6      matrix b = mshape(theta,k,j-1)
7      matrix tmp = X*b
8      series den = (1 + sumr(exp(tmp)))
9  
10     loop for i=1..j --quiet
11         if i = 1
12             series p$i = 1/den
13         else
14             scalar q = i - 1
15             series num = exp(X[q,]*b[,q])
16             series p$i=num/den
17         endif
18         list probs += p$i
19     endloop
20     return probs
21 end function
```

The inputs are the dependent variable, y, a list of independent variables, x, and the coefficients from multinomial logit estimation, theta. The function will return a list that contains the computed probabilites. These will be added to the dataset.

An empty list must be created, which is done using list null. In line 3 the independent variables are converted into a matrix called X. The fourth line finds the maximum category in the coding of the dependent variable. Ours, psechoice, takes values 1, 2, and 3 in the data so this will return the value 3. If your data are coded 0, 1, 2 like they sometimes are, you will have to alter your script to account for that. The scalar k counts the number of independent variables. In MNL there are J choices and $J-1$ sets of k parameters. The matrix b reshapes the $(J-1)k \times 1$ vector of coefficients produced by logit --multinomial into a $k \times (J-1)$ matrix. Each column of this matrix contains the coefficients for the $(j-1)^{th}$ choice. The matrix labeled tmp computes the indexes for each choice. The matrix `den` computes the row sums of these to produce the denominator found in the MNL probabilities.

The loop is required because of the way MNL probabilities are computed. For the normalized choice, the numerator is 1. For the others it is $\exp(index_j)$. The computed probabilities are added to the list `probs` using the operator (`+=`), which is a fancy way of appending new results to existing results. The loop ends and you must return the `list probs` in order for the computed series to be passed out of the function and added to the dataset. If you have been working through the simpler functions we've considered up to this point, then the added complexity of this one will not be bothersome. If this is your first function in **gretl**, then you are no doubt lost. Again, it is not required to get probabilities from MNL.

To use the function, create the variable list, estimate the model and save the coefficients to a matrix. Finally, create a list and print it by observation as in:

```
open "@gretldir\data\poe\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
matrix theta = $coeff
list n = mlogitprob(psechoice, x, theta)
smpl 1 12
print n --byobs
smpl full
```

Of course, you could make things easy on yourself and just use the accessor, `$mnlprobs`. This gives you access to the probabilities from the multinomial logit that we obtained using the GUI. Not much fun in that, but it is easy. However, with our homemade function we can compute marginal effects.

To get average marginal effects is a snap at this point. We will simply add 1 to the value of *grades*, recompute the probabilities, and average the difference between the two. This will require renaming the predicted probabilities, but that is easily done.

```
rename p1 p01
rename p2 p02
rename p3 p03

series grade1 = grades+1
list x1 = const grade1
list n1 = mlogitprob(psechoice, x1, theta)
series d1 = p1-p01
series d2 = p2-p02
series d3 = p3-p03
summary d* --simple
```

The script yields:

```
Summary statistics, using the observations 1 - 1000

                    Mean         Minimum         Maximum      Std. Dev.
d1              0.080044       0.0092216         0.11644       0.034329
d2           -0.00014717        -0.11560        0.017795       0.023719
d3             -0.066086        -0.31899     -0.00037743       0.069045
```

As a student's performance gets worse (*grades* increases by 1), the average probability of not attending college goes up by 0.08. The probability of attending 4-year school declines by -0.066.

Finding marginal effects at specific points requires another function, but it is very similar to the one used above. The only substantive change is feeding the function a matrix rather than the list of variables and changing series computations within the function to either scalars or matrix. Here is the new function called

```
function matrix mlogitprob_at(series y, matrix X, matrix theta)
    matrix probs = {}
    scalar j = max(y)
    scalar k = cols(X)
    matrix b = mshape(theta,k,j-1)
    matrix tmp = X*b
    scalar den = (1 + sumr(exp(tmp)))

    loop for i=1..j --quiet
        if i = 1
            scalar  p$i = 1/den
        else
            scalar q = i - 1
            scalar num = exp(X*b[,q])
            scalar p$i=num/den
        endif
        matrix probs = probs ~ p$i
    endloop
    return probs
end function
```

The function is quite easy to use and reproduces the results in *POE4* Table 16.3.

```
open "@gretldir\data\poe\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
matrix theta = $coeff
matrix Xm = {1 , quantile(grades,.50)}
matrix p50 = mlogitprob_at(psechoice, Xm, theta)
matrix Xm = {1 , quantile(grades,.05)}
matrix p05 = mlogitprob_at(psechoice, Xm, theta)
printf "\nThe predicted probabilities for student\
grades=%6.2f is %8.4f\n",quantile(grades,.05), p05
printf "\nThe predicted probabilities for student\
grades=%6.2f is %8.4f\n",quantile(grades,.50), p50
```

To use the function to get marginal effects of 1 unit change in grades for median and 95th percentile students we create quantiles based on the series grades and use these in our new function. Taking the difference in probabilities will give us an approximate marginal effect at those quantiles.

```
open "@gretldir\data\poe\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
matrix theta = $coeff
scalar q50 = quantile(grades,.50)
matrix Xm = {1 , q50-0.5}
matrix p0 = mlogitprob_at(psechoice, Xm, theta)
matrix Xm = {1 , q50+0.5}
matrix p1 = mlogitprob_at(psechoice, Xm, theta)
matrix me = p1-p0
printf "\nThe marginal effect of grades for student\
grades =%6.2f is %8.4f\n",median(grades), me

scalar q05 = quantile(grades,.05)
matrix Xm = {1 , q05-0.5}
matrix p0 = mlogitprob_at(psechoice, Xm, theta)
matrix Xm = {1 , q05+0.5}
matrix p1 = mlogitprob_at(psechoice, Xm, theta)
matrix me = p1-p0
printf "\nThe marginal effect of grades for student\
  grades=%6.2f is %8.4f\n", q05, me
```

Notice that the script returns the predicted probabilities for these students and the change in those probabilities resulting from a 1 unit change in grades. The total probabilities should sum to 1 and the marginal effects should sum to zero. This script also uses a common trick. The one unit change is evaluated at ± 0.5 on either side of each quantile; then the discrete difference is taken. The results match those in *POE4* reasonably well.

The option `--multinomial` is used when the choices are unordered. For **ordered logit**, this option is omitted. **Gretl** takes a look at the dependent variable, in this case `psechoice`, to make sure that it is actually discrete. Ours takes on three possible values (1, 2, or 3) and the `logit` function in **gretl** will handle this automatically.

16.3.1 Using a Script for MNL

In this section I'll give you an idea of how to estimate this model using another **gretl** script and in section 16.10 I'll show you how to estimate the model in another free software called **R**.

Although versions of **Gretl** prior to 1.8.1 did not include a specific function for estimating MNL, it can be estimated with a little effort. **Gretl** contains two things that make this reasonably easy to do. First, it includes a module for maximizing user specified likelihood functions (see chapter 14 for other examples). To use the `mle` function, the user has to write a program in **hansl** to compute a model's log-likelihood given the data. The parameters of the log-likelihood must be declared and given starting values (using the `scalar` command). If you want, you can specify the derivatives of the log-likelihood function with respect to each of the parameters; if analytical derivatives are not supplied, a numerical approximation is computed. In many instances, the numerical approximations work quite well. In the event that the computations based on numerical derivatives fail, you may have to specify analytical ones to make the program work.

What appears below is taken from the **Gretl** Users Guide. The example for MNL for *POE4* requires only a slight modification in order for the program to run with our dataset.

The multinomial logit function, which can be found in the **Gretl** User's Guide, is defined

```
function mlogitlogprobs(series y, matrix X, matrix theta)
  scalar n = max(y)
  scalar k = cols(X)
  matrix b = mshape(theta,k,n)
  matrix tmp = X*b
  series ret = -ln(1 + sumr(exp(tmp)))
  loop for i=1..n --quiet
    series x = tmp[,i]
    ret += (y=$i) ? x : 0
  end loop
return series ret
end function
```

The function is named `mlogitlogprobs` and has three arguments. The first is the dependent variable, `series y`, the second is set of independent variables contained in `matrix X`, and the last is the matrix of parameters, called `theta`. Scalars in the function are defined for sample size, number of regressors, and the coefficients are placed in an $n \times k$ array in order to match the dimensionality of the data. The index `tmp=X*b` is created and `ret` returns the log-likelihood function. Don't worry if you can't make sense of this because you should not have to change any of this to estimate MNL with another dataset. That is one of the beauties of defining and using a function.

To use the `mlogitlogprobs` function, you need to know a little about how it works. You will have to get your data into the right form in order for the function to work properly. After loading the data, make sure that the dependent choice variable is in the correct format for the function.

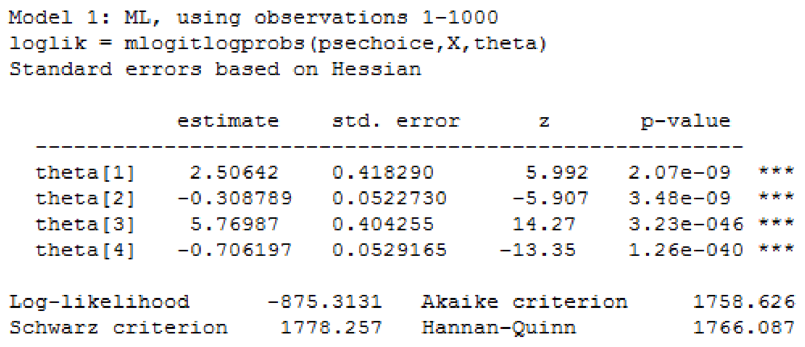
The function requires the choices to start at 0. If you list the data, you'll find that psechoice is coded 1, 2, 3 instead of the required 0, 1, 2. So the next step is to subtract 1 from psechoice.

Create the matrix of regressors, define the number of regressors and use these to initialize the matrix of coefficients, theta. Then list the dependent variable, matrix of independent variables, and the initialized parameter matrix as arguments in the function. Click the run button and wait for the result.

```
open "@gretldir\data\poe\nels_small.gdt"
psechoice = psechoice-1  # dep. var. must be 0-based
list x = const grades
smpl full
matrix X = { x }
scalar k = cols(X)
matrix theta = zeros(2*k, 1)
mle loglik = mlogitlogprobs(psechoice,X,theta)
    params theta
end mle --hessian
```

The only changes I had to make to the original example in the **Gretl** User Guide are (1) change the dataset (2) change the dependent variable to psechoice (3) put the desired regressors into X and (4) make sure the function contains the desired variables.

The results from the program appear below in Figure 16.4. Wow! They match those in *POE4* and are dirt simple to obtain. Finally, if you want to produce the probabilities and marginal effects,



```
Model 1: ML, using observations 1-1000
loglik = mlogitlogprobs(psechoice,X,theta)
Standard errors based on Hessian

             estimate    std. error      z       p-value
  ------------------------------------------------------
  theta[1]    2.50642    0.418290       5.992   2.07e-09  ***
  theta[2]   -0.308789   0.0522730     -5.907   3.48e-09  ***
  theta[3]    5.76987    0.404255      14.27    3.23e-046 ***
  theta[4]   -0.706197   0.0529165    -13.35    1.26e-040 ***

Log-likelihood      -875.3131   Akaike criterion     1758.626
Schwarz criterion    1778.257   Hannan-Quinn         1766.087
```

Figure 16.4: These results are from a **gretl** function taken from the **Gretl** Users Guide.

you can use the estimates **gretl** has stored in the 4×1 vector called theta. This was the approach taken in the preceding section and I won't repeat the details here.

16.4 Conditional Logit

Conditional logit is used to model choices when there is alternative specific information available. When choosing among brands of soft-drinks, you have information on the choice that an individual makes as well as the price of each available alternative. This kind of data differs from the scanner data used in multinomial logit because in that example we only had information on the grade earned by an individual; there were no alternative grades for those choosing what kind of school to attend. The grade was specific to the individual, not the choices. In conditional logit there is information about each alternative. Models that combine individual specific information and choice specific information are referred to as **mixed**. Such data are somewhat rare. Usually you either have information on the individual (income or race) or the choices (prices and advertising), but not both.

The following example should make this more clear. We are studying choices among three soft-drinks: Pepsi, Coke, and Seven-up. Each may sell for a different price. Each individual purchases one of the brands. The probability that individual i chooses j is

$$p_{ij} = \frac{\exp(\beta_{1j} + \beta_2 price_{ij})}{\exp(\beta_{11} + \beta_2 price_{i1}) + \exp(\beta_{12} + \beta_2 price_{i2}) + \exp(\beta_{13} + \beta_2 price_{i3})} \tag{16.16}$$

Now there is only 1 parameter that relates to *price*, but there are J=3 constants. One of these is not identified and is set to zero. This is referred to as **normalization** and in our case we set $\beta_{13} = 0$.

Below is a function and a script that will estimate the conditional logit model for the soft drink example by maximum likelihood. The function is not general in the sense that it will work with another model, but the basic idea could be used to generalize it to do so. The MCMC method discussed below is an alternative that is more ready for general use, but the results will differ somewhat from maximum likelihood estimation.

The function computes the value of the log-likelihood for the conditional logit model. The inputs consist of two lists and a vector of starting values. The first list contains indicator variables identifying which choice was made (*pepsi*, *7up* or *coke*). The second list contains the regressors.

Conditional Logit script

```
function matrix clprobs(list y, list x, matrix theta)
matrix Y = { y }
matrix p = { x }
scalar n = $nobs
matrix P = {}
loop i=1..n --quiet
    scalar i1 = exp(theta[1]+theta[3]*p[i,1])
    scalar i2 = exp(theta[2]+theta[3]*p[i,2])
    scalar i3 = exp(theta[3]*p[i,3])
    scalar  d = i1+i2+i3
    matrix pp = (Y[i,1]=1)*i1/d + (Y[i,2]=1)* i2/d + (Y[i,3]=1)* i3/d
    matrix P = P | pp
```

```
13 endloop
14 return sumc(ln(P))
15 end function
16
17 open "@gretldir\data\poe\cola2.gdt"
18 list y = pepsi sevenup coke
19 list x =  pr_pepsi pr_7up pr_coke
20 matrix theta = {-1.19, .283, .1038}
21
22 mle lln = clprobs(y, x, theta)
23     params theta
24 end mle
```

Lines 2 and 3 convert the lists to matrices. The number of observations is counted in line 4 and an empty matrix is created to hold the result in 5. The loop that starts in line 6 just computes the probabilities for each observed choice. The scalars `i1`, `i2` and `i3` are added together for the denominator of equation (16.16); each of these scalars is divided by the denominator term. The logical statements, i.e., (`Y[i,1]=1`) is multiplied by the probability. If the person chooses the first alternative, this `i1/d` is set to `pp`. The other logicals are false at this point and are zero. The vector `pp` contains the probabilities of making the choice for the alternative actually chosen. The return is the sum of the logs of the probabilities, which is just the log-likelihood.

The results from this function and MLE estimation are found below:

```
Using numerical derivatives
Tolerance = 1.81899e-012
Function evaluations: 41
Evaluations of gradient: 12

Model 2: ML, using observations 1-1822
lln = clprobs(y, x, theta)
Standard errors based on Hessian

             estimate    std. error      z       p-value
  ----------------------------------------------------------
  theta[1]    0.283166   0.0623772      4.540   5.64e-06  ***
  theta[2]    0.103833   0.0624595      1.662   0.0964    *
  theta[3]   -2.29637    0.137655     -16.68    1.77e-062 ***

Log-likelihood      -1824.562   Akaike criterion     3655.124
Schwarz criterion    3671.647   Hannan-Quinn         3661.220
```

These match the results in *POE4*. Even the estimated standard errors are the same out to 4 decimal places. Very good indeed. Substantively, the price coefficient is -2.296 and is significantly different from zero at any reasonable level of significance.

16.5 Ordered Probit

In this example, the probabilities of attending no college, a 2 year college, and a 4 year college after graduation are modeled as a function of a student's grades. In principle, we would expect that those with higher grades to be more likely to attend a 4 year college and less likely to skip college altogether. In the dataset, grades are measured on a scale of 1 to 13, with 1 being the highest. That means that if higher grades increase the probability of going to a 4 year college, the coefficient on grades will be *negative*. The probabilities are modeled using the normal distribution in this model where the outcomes represent increasing levels of difficulty.

We can use **gretl** to estimate the ordered probit model because its `probit` command actually handles multinomial ordered choices as well as binomial choice. Open the *nels_small.gdt* data

```
open "@gretldir\data\poe\nels_small.gdt"
probit psechoice const grades
```

The results in below are very much like the ones in *POE4* and those produced by Bayesian estimation provided by **MCMCpack**, the generation of which is discussed in section 16.10.3.

Model 2: Ordered Probit, using observations 1–1000
Dependent variable: psechoice

	Coefficient	Std. Error	z	p-value
grades	−0.306624	0.0191735	−15.9921	0.0000
cut1	−2.94559	0.146828	−20.0615	0.0000
cut2	−2.08999	0.135768	−15.3938	0.0000

Mean dependent var	2.305000	S.D. dependent var	0.810328
Log-likelihood	−875.8217	Akaike criterion	1757.643
Schwarz criterion	1772.367	Hannan–Quinn	1763.239

Number of cases 'correctly predicted' = 587 (58.7 percent)
Likelihood ratio test: $\chi^2(1)$ = 285.672 [0.0000]

Test for normality of residual –
Null hypothesis: error is normally distributed
Test statistic: $\chi^2(2)$ = 2.96329
with p-value = 0.227264

The coefficient on *grades* is negative and significant at 5%. This means that as the *grades* variable gets larger (grades get worse), the index is getting smaller and at the margins 2-year college attendees are being pushed towards no college and the 4-year college attendees are being pushed toward the 2-year option. We know for sure that the probability of being in the lowest category goes up and of being in the highest category goes down. Whatever happens in the middle depends on net effects of people being pushed out below and pulled in from above.

The other two parameters are the cut-off points that determine the boundaries between categories. The parameter $\mu_1 < \mu_2$.

The algebraic expressions for the marginal effects are:

$$\frac{\partial P(y=1)}{\partial grades} = -\phi(\mu_1 - \beta grades)\beta$$
$$\frac{\partial P(y=2)}{\partial grades} = [\phi(\mu_1 - \beta grades) - \phi(\mu_2 - \beta grades)]\beta$$
$$\frac{\partial P(y=3)}{\partial grades} = \phi(\mu_2 - \beta grades)\beta$$

where ϕ is the probability density function of a standard normal distribution. The parameters μ_1 and μ_2 are the thresholds (or cut-off points) and β is the coefficient on `grades`. So, for example if you want to calculate the marginal effect on the probability of attending a 4-year college ($y = 3$) for a student having grades at the median (6.64) and 5th percentile (2.635) use:

```
probit psechoice grades
k = $ncoeff
matrix b = $coeff[1:k-2]
mu1 = $coeff[k-1]
mu2 = $coeff[k]

matrix X = {6.64}
scalar Xb = X*b
P3a = pdf(N,mu2-Xb)*b

matrix X = 2.635
scalar Xb = X*b
P3b = pdf(N,mu2-Xb)*b

printf "\nFor the median grade of 6.64, the marginal\
effect is %.4f\n", P3a
printf "\nFor the 5th percentile grade of 2.635, the\
marginal effect is %.4f\n", P3b
```

This yields

```
For the median grade of 6.64, the marginal effect is -0.1221

For the 5th percentile grade of 2.635, the marginal effect is -0.0538
```

16.6 Poisson Regression

When the dependent variable in a regression model is a count of the number of occurrences of an event you may want to use the poisson regression model. In these models, the dependent variable is a nonnegative integer, (i.e., $y = 0, 1, \ldots$), which represent the number of occurrences of a particular event. The probability of a given number of occurrences is modeled as a function of independent variables.

$$P(Y = y|x) = \frac{e^{-\lambda}\lambda^y}{y!} \quad y = 0, 1, 2, \ldots \tag{16.17}$$

where $\lambda = \beta_1 + \beta_2 x$ is the regression function.

Estimating this model using maximum likelihood is very simple since the MLE of the poisson regression model is already programmed into **gretl**. The syntax for a script is the same as for linear regression except you use the `possion` command in place of `ols`. This is shown in the following script which replicates the example from your textbook.

A country's total number of medals (medaltot) in the 1988 olympics is modeled as a function of ln(*gdp*) and ln(*pop*). Of course, you can also access the poisson regression estimator from the pull-down menus using `Model>Nonlinear models>Possion`. To replicate the example in *POE4* be sure to restrict the sample to 1988 before estimating the model.

```
open "@gretldir\data\poe\olympics.gdt"
set echo off
set messages off
smpl year = 88 --restrict
logs pop gdp
poisson medaltot const l_pop l_gdp
scalar m1 = exp($coeff(const)+$coeff(l_pop)*ln(quantile(pop,0.5)) \
                   +$coeff(l_gdp)*ln(quantile(gdp,0.5)))
scalar m2 = exp($coeff(const)+$coeff(l_pop)*ln(quantile(pop,0.75)) \
                   +$coeff(l_gdp)*ln(quantile(gdp,0.5)))
scalar mft = exp($coeff(const)+$coeff(l_pop)*ln(quantile(pop,0.5)) \
                   +$coeff(l_gdp)*ln(quantile(gdp,0.5)))*$coeff(l_gdp)
printf "\nMarginal Effect at the medians is %.3f\n",mft
printf "\nAverage medal total for median gdp and pop is %.3f\n",m1
printf "\nAverage medal total for median gdp and 75th quantile pop is %.3f\n",m2
```

In the script, we have also computed a marginal effect and two means for representative countries. The algebraic justification for these can be found below.

The results for poisson model are:

Poisson, using observations 1–205 ($n = 151$)
Missing or incomplete observations dropped: 54

Dependent variable: medaltot

	Coefficient	Std. Error	z	p-value
const	−15.8875	0.511805	−31.0420	0.0000
l_pop	0.180038	0.0322801	5.5773	0.0000
l_gdp	0.576603	0.0247217	23.3238	0.0000

Mean dependent var	4.887417	S.D. dependent var	16.62670
Sum squared resid	25165.58	S.E. of regression	13.03985
McFadden R^2	0.546854	Adjusted R^2	0.544972
Log-likelihood	−722.3365	Akaike criterion	1450.673
Schwarz criterion	1459.725	Hannan–Quinn	1454.350

Overdispersion test: $\chi^2(1) = 9.20208$ [0.0024]

Both the size and the wealth of a country contribute to higher medal counts. The probability that the random variable Y is equal to an outcome y can be obtained based on the formula:

$$\widehat{P(Y = y)} = \frac{\exp(-\tilde{\lambda}_0)\tilde{\lambda}_0^y}{y!} \quad y = 0, 1, 2, \ldots \tag{16.18}$$

where $\widehat{E(y_0)} = \tilde{\lambda}_0 = \exp(\tilde{\beta}_1 + \tilde{\beta}_2 x_0)$. The marginal effects are simple

$$\frac{\partial E(y_i)}{\partial x_i} = \lambda_i \beta_2 \tag{16.19}$$

The marginal effect at the medians and the average medal totals for two hypothetical countries are:

```
Marginal Effect at the medians is 0.498

Average medal total for median gdp and pop is 0.863

Average medal total for median gdp and 75th quantile pop is 1.051
```

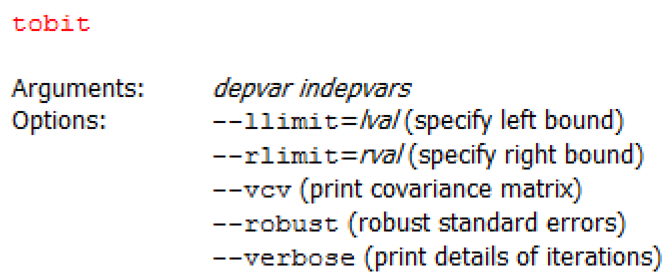
16.7 Tobit

The tobit model is essentially just a linear regression where some of the observations on your dependent variable have been censored. A **censored** variable is one that, once it reaches a limit, it is recorded at that limit no matter what it's actual value might be. For instance, anyone earning \$1 million or more per year might be recorded in your dataset at the upper limit of \$1 million. That means that Bill Gates and the authors of your textbook earn the same amount in the eyes of your dataset (just kidding, gang). Least squares can be seriously biased in this case and it is wise to use a censored regression model (tobit) to estimate the parameters of the regression when a portion of your sample is censored.

Hill et al. (2011) consider the following model of hours worked for a sample of women. equation (16.20).

$$hours_i = \beta_1 + \beta_2 educ_i + \beta_3 exper_i + \beta_4 age_i + \beta_5 kidsl6_i + e_i \tag{16.20}$$

They estimate the model as a censored regression since a number of people in the sample are found to work zero hours. The command for censored regression in **gretl** is tobit, the syntax for which is shown below



```
tobit

Arguments:    depvar indepvars
Options:      --llimit=lval (specify left bound)
              --rlimit=rval (specify right bound)
              --vcv (print covariance matrix)
              --robust (robust standard errors)
              --verbose (print details of iterations)
```

The routine allows you to specify the left and right points at which censoring occurs. You also can choose a robust covariance that is robust with respect to the normality assumption used to obtain the MLE (not heteroskedasticity).

Estimation of this model in **gretl** is shown in the following script which replicates the example from *POE4*. The script estimates a tobit model of hours worked and generates the marginal effect of another year of schooling on the average hours worked. Hours are assumed to be censored at zero and no lower limit need be specified.

```
open "@gretldir\data\poe\mroz.gdt"
list xvars = const educ exper age kidsl6
tobit hours xvars
```

The results from the basic tobit estimation of the hours worked equation are:

Tobit, using observations 1–753
Dependent variable: hours
Standard errors based on Hessian

	Coefficient	Std. Error	z	p-value
const	1349.88	386.298	3.4944	0.0005
educ	73.2910	20.4698	3.5804	0.0003
age	−60.7678	6.88310	−8.8286	0.0000
exper	80.5353	6.28051	12.8231	0.0000
kidsl6	−918.918	111.588	−8.2349	0.0000

Chi-square(4)	244.2972	p-value	1.10e–51
Log-likelihood	−3827.143	Akaike criterion	7666.287
Schwarz criterion	7694.031	Hannan–Quinn	7676.975

$\hat{\sigma}$ = 1133.7 (40.8769)
Left-censored observations: 325
Right-censored observations: 0

Test for normality of residual –
Null hypothesis: error is normally distributed
Test statistic: $\chi^2(2) = 6.31679$
with p-value = 0.0424938

The marginal effect of another year of schooling on hours worked is

$$\frac{\partial E(hours_i)}{\partial educ_i} = \Phi(\widehat{hours_i})\hat{\beta}_2, \tag{16.21}$$

where $\widehat{hours_i}$ is the estimated regression function evaluated at the mean levels of education, experience, and age for a person with one child under the age of six. Then, the `cnorm` function is used to compute the normal cdf, $\Phi(\widehat{hours_i})$, evaluated at the prediction.

```
matrix beta = $coeff
scalar H_hat = beta[1]+beta[2]*mean(educ)+beta[3]*mean(exper) \
       +beta[4]*mean(age)+beta[5]*1
scalar z = cnorm(h_hat/$sigma)
scalar me_educ = z*$coeff(educ)

printf "\nThe computed scale factor = %6.5g\nand marginal effect of\
another year of schooling = %5.5g.\n", z, me_educ
```

This produces

```
The computed scale factor =  0.363
and marginal effect of another year of schooling = 26.605.
```

Note, the backward slashes (\) used at the end of the first two lines in the generation of `H_hat` are continuation commands. The backslash at the end of a line tells **gretl** that the next line is a continuation of the current line. This helps keep your programs looking good (and in this case, fitting within the margins of the page!).

A slightly easier way to evaluate the index, $\widehat{hours_0}$, is to use matrices. In the alternative version we convert the data to a matrix and create a vector of the variable means. The average number of children (0.24), is replaced with a 1 and the index is computed using vector algebra.

```
tobit hours xvars
matrix beta = $coeff
matrix X = { xvars }
matrix meanx = meanc(X)
matrix meanx[1,5]=1
scalar h_hat=meanx*beta
printf "\nTwo ways to compute a prediction get %8.4f and %8.4f\n", h_hat, H_hat
```

This produces

```
Two ways to compute a prediction get -397.3022 and -397.3022
```

Finally, estimates of the restricted sample using least squares and the full sample that includes the zeros for hours worked follow.

```
list xvars = const educ exper age kidsl6
smpl hours > 0 --restrict
ols hours xvars
smpl --full
ols hours xvars
```

Notice that the sample is restricted to the positive observations using the `smpl hours > 0 --restrict` statement. To estimate the model using the entire sample the full range is restored using `smpl full`.

These were added to a model table and the result appears below:

Dependent variable: hours

	(1)	(2)	(3)
	Tobit	OLS	OLS
const	1350^{**}	1830^{**}	1335^{**}
	(386.3)	(292.5)	(235.6)
educ	73.29^{**}	-16.46	27.09^{**}
	(20.47)	(15.58)	(12.24)
exper	80.54^{**}	33.94^{**}	48.04^{**}
	(6.281)	(5.009)	(3.642)
age	-60.77^{**}	-17.11^{**}	-31.31^{**}
	(6.883)	(5.458)	(3.961)
kidsl6	-918.9^{**}	-305.3^{**}	-447.9^{**}
	(111.6)	(96.45)	(58.41)

n	753	428	753
$\bar{R}^2$		0.1168	0.2531
ℓ	-3827	-3426	-6054

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

You can see that the tobit regression in column (1) and the OLS regression in column (3) use the entire sample. Estimating the model by OLS with the zero observations in the model reduces all of the slope coefficients by a substantial amount. Tossing out the zero observations as in the OLS regression in column (2) reverses the sign on years of schooling (though it is not significant). For only women in the labor force, more schooling has no effect on hours worked. If you consider the entire population of women, more schooling does increase hours, presumably by enticing more women into the labor force.

16.8 Simulation

You can use **gretl** to show that least squares is biased when the sample is censored using a Monte Carlo simulation. The simulated data are generated

$$y_i^* = -9 + 1x_i + e_i \tag{16.22}$$

where $e_i \sim N(0, 16)$. Then,

$$y_i = \begin{cases} y_i^* & \text{if } y_i^* > 0 \\ 0 & \text{if } y_i^* \leq 0 \end{cases}$$

The $x_i \sim U(0, 20)$, which are held constant in the simulated samples.

The following script demonstrates that least squares is indeed biased when all observations, including the zero ones, are included in the sample. The line `series yc = (y > 0) ? y : 0` is a logical statement that generates 'y' or '0' depending on whether the statement in parentheses is true. Thus, a new variable, `yc`, is created that takes the value `y` if `y>0` and is zero if not. Another logical is used in line 10 to generate an indicator variable, `w`. The variable `w=1` when the statement in the parentheses `(y>0)` is true. Otherwise it is equal to zero. The variable `w` is used in `wls` to exclude the observations that have zero hours of work.

```
1 nulldata 200
2 series xs = 20*uniform()
3 list x = const xs
4 series ys = -9 + 1*xs
```

```
loop 1000 --progressive --quiet
   series y = ys + normal(0,4)
   series yc = (y > 0) ? y : 0
   ols y x
   ols yc x
   series w = (yc>0)
   wls w yc x
   tobit yc x
endloop
```

Because the tobit estimator is iterative, there is a lot of output generated to the screen. However, if you scroll down you will find the results from this simulation. Recall, the value of the constant was set at -9 and the slope to 1. The column labeled 'mean of the estimated coefficients' is the average value of the estimator in 1000 iterations of the Monte Carlo. When the estimator is unbiased, this number should be 'close' to the true value in the statistical sense. You can use the next column (std. dev. of estimated coefficients) to compute a Monte Carlo standard error to perform a test.

Since the coefficients are being averaged over the number, *NMC*, of simulated samples, the central limit theorem should apply; the mean should be approximately normally distributed and the variance of the mean equal to $\sigma/\sqrt{NMC}$. The result in the column labeled 'std. dev. of estimated coefficients' is an estimate of σ. To test for unbiasedness of the tobit estimator of the slope ($H_o : b_2 = 1$ against the two-sided alternative) compute:

$$\sqrt{NMC}(\bar{b}_2 - 1)/\hat{\sigma} = \sqrt{1000}(1.00384 - 1)/0.0737160 = 1.647 \sim N(0,1) \tag{16.23}$$

if the estimator is unbiased. The 5% critical value is 1.96 and the unbiasedness of the tobit estimator cannot be rejected at this level of significance. See Adkins (2011*b*) for more examples and details.

```
OLS estimates using the 200 observations 1-200
Statistics for 1000 repetitions
Dependent variable: y

                   mean of      std. dev. of     mean of     std. dev. of
                  estimated      estimated      estimated     estimated
      Variable   coefficients   coefficients   std. errors   std. errors

        const      -9.00646       0.548514       0.562873      0.0283463
           xs      0.999336       0.0494064      0.0500999     0.00252303


OLS estimates using the 200 observations 1-200
Statistics for 1000 repetitions
Dependent variable: yc

                   mean of      std. dev. of     mean of     std. dev. of
                  estimated      estimated      estimated     estimated
      Variable   coefficients   coefficients   std. errors   std. errors
```

```
        const      -2.20798       0.232987       0.405670       0.0226162
           xs      0.558122      0.0351037      0.0361076      0.00201301


WLS estimates using the 108 observations 1-200
Statistics for 1000 repetitions
Dependent variable: yc

                   mean of      std. dev. of     mean of     std. dev. of
                  estimated      estimated      estimated      estimated
      Variable   coefficients   coefficients   std. errors    std. errors

        const      -2.09574       0.960994        1.09869       0.118095
           xs      0.602659      0.0743574      0.0774449     0.00757796


Tobit estimates using the 200 observations 1-200
Standard errors based on Hessian
Statistics for 1000 repetitions
Dependent variable: yc

                   mean of      std. dev. of     mean of     std. dev. of
                  estimated      estimated      estimated      estimated
      Variable   coefficients   coefficients   std. errors    std. errors

        const      -9.07517       0.988720       0.994815      0.0954671
           xs       1.00384      0.0737160      0.0742580     0.00629653
```

The estimators in the first set and last are unbiased. OLS in the first instance uses the full sample that has not been censored. In reality, the censored observations won't be available so this estimator is not really feasible outside of the Monte Carlo. The tobit estimator in the last set is feasible, however. Clearly it is working pretty well with this data generation process. The second set of results estimates the model using the entire sample with 0 recorded for the censored observations. It is not performing well at all and is no better than the third set of results that discards the zero hours observations. It does reveal what happens, conditional on being in the labor force though. So, it is not without its uses.

16.9 Selection Bias

Selection bias occurs when your sample is truncated and the cause of that truncation is correlated with your dependent variable. Ignoring the correlation, the model could be estimated using least squares or truncated least squares. In either case, obtaining consistent estimates of the regression parameters is not possible. In this section the basic features of the this model will be presented.

Consider a model consisting of two equations. The first is the **selection equation**, defined

$$z_i^* = \gamma_1 + \gamma_2 w_i + u_i, \quad i = 1, \ldots, N \tag{16.24}$$

where z_i^* is a **latent variable**, γ_1 and γ_2 are parameters, w_i is an explanatory variable, and u_i is a random disturbance. A latent variable is unobservable, but we do observe the dichotomous variable

$$z_i = \begin{cases} 1 & z_i^* > 0 \\ 0 & \text{otherwise} \end{cases} \tag{16.25}$$

The second equation, called the **regression equation**, is the linear model of interest. It is

$$y_i = \beta_1 + \beta_2 x_i + e_i, \quad i = 1, \ldots, n, \quad N > n \tag{16.26}$$

where y_i is an observable random variable, β_1 and β_2 are parameters, x_i is an exogenous variable, and e_i is a random disturbance. It is assumed that the random disturbances of the two equations are distributed as

$$\begin{bmatrix} u_i \\ e_i \end{bmatrix} \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & \sigma_e^2 \end{pmatrix}\right] \tag{16.27}$$

A selectivity problem arises when y_i is observed only when $z_i = 1$ and $\rho \neq 0$. In this case the ordinary least squares estimator of β in (16.26) is biased and inconsistent. A consistent estimator has been suggested by Heckman (1979) and is commonly referred to as **Heckman's two-step estimator**, or more simply, **Heckit**. Because the errors are normally distributed, there is also a maximum likelihood estimator of the parameters. **Gretl** includes routines for both.

The two-step (Heckit) estimator is based on conditional mean of y_i given that it is observed:

$$E[y_i|z_i > 0] = \beta_1 + \beta_2 x_i + \beta_\lambda \lambda_i \tag{16.28}$$

where

$$\lambda_i = \frac{\phi(\gamma_1 + \gamma_2 w_i)}{\Phi(\gamma_1 + \gamma_2 w_i)} \tag{16.29}$$

is the *inverse Mill's ratio*;$(\gamma_1 + \gamma_2 w_i)$ is the **index function**; $\phi(\cdot)$ is the standard normal probability density function evaluated at the index; and $\Phi(\cdot)$ is the standard normal cumulative density function evaluated at the index. Adding a random disturbance yields:

$$y_i = \beta_1 + \beta_2 x_i + \beta_\lambda \lambda_i + v_i \tag{16.30}$$

It can be shown that (16.30) is heteroskedastic and if λ_i were known (and nonstochastic), then the selectivity corrected model (16.30) could be estimated by generalized least squares. Alternately, the heteroskedastic model (16.30) could be estimated by ordinary least squares, using White's heteroskedasticity consistent covariance estimator (HCCME) for hypothesis testing and the construction of confidence intervals. Unfortunately, λ_i is not known and must be estimated using the sample. The stochastic nature of λ_i in (16.30) makes the automatic use of HCCME in this context inappropriate.

The two-steps of the Heckit estimator consist of

1. Estimate the selection equation to obtain $\hat{\gamma}_1$ and $\hat{\gamma}_2$. Use these in equation (16.29) to estimate the inverse Mill's ratio, $\hat{\lambda}_i$.
2. Add $\hat{\lambda}_i$ to the regression model as in equation (16.30) and estimate it using least squares.

This ignores the problem of properly estimating the standard errors, which requires an additional step. **Gretl** takes care of this automatically when you use the `heckit` command.

The example from *POE4* uses the *mroz.gdt* data. The first thing we'll do is to estimate the model ignoring selection bias using least squares on the nonzero observations. Load the data and generate the natural logarithm of wages. Since wages are zero for a portion of the sample, **gretl** will generate an error when you take the natural logs. You can safely ignore it as **gretl** will simply create missing values for the variables that cannot be transformed. Then use the `ols` command to estimate a linear regression on the truncated subset.

```
open "@gretldir\data\poe\mroz.gdt"
logs wage
ols l_wage const educ exper
```

The results are:

Model 1: OLS estimates using the 428 observations 1–428
Dependent variable: l_wage

	Coefficient	Std. Error	t-ratio	p-value
const	−0.400174	0.190368	−2.1021	0.0361
educ	0.109489	0.0141672	7.7283	0.0000
exper	0.0156736	0.00401907	3.8998	0.0001

Mean dependent var	1.190173	S.D. dependent var	0.723198
Sum squared resid	190.1950	S.E. of regression	0.668968
R^2	0.148358	Adjusted R^2	0.144350
$F(2,425)$	37.01805	P-value(F)	1.51e–15
Log-likelihood	−433.7360	Akaike criterion	873.4720
Schwarz criterion	885.6493	Hannan–Quinn	878.2814

Notice that the sample has been truncated to include only 428 observations for which hours worked are actually observed. The estimated return to education is about 11%, and the estimated coefficients of both education and experience are statistically significant.

The Heckit procedure takes into account that the decision to work for pay may be correlated with the wage a person earns. It starts by modeling the decision to work and estimating the resulting selection equation using a probit model. The model can contain more than one explanatory variable, w_i, and in this example we have four: a womans age, her years of education, a dummy variable for whether she has children and the marginal tax rate that she would pay upon earnings if employed. Generate a new variable `kids` which is a dummy variable indicating the presence of any kids in the household. Estimate the probit model, generate the index function, and use it to compute the inverse Mill's ratio variable. Finally, estimate the regression including the IMR as an explanatory variable.

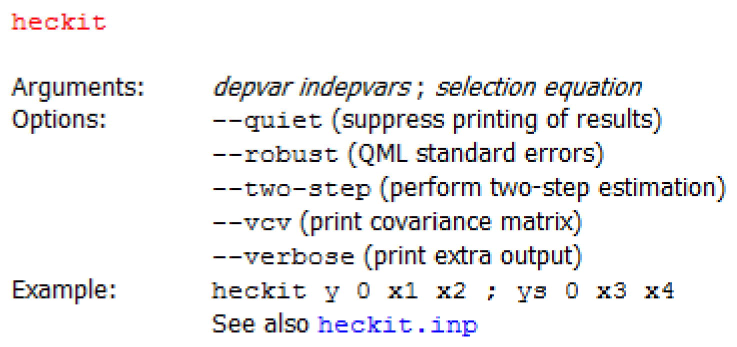
```
open "@gretldir\data\poe\mroz.gdt"
series kids = (kidsl6+kids618>0)
logs wage
series kids = (kidsl6+kids618>0)
list X = const educ exper
list W = const mtr age kids educ probit lfp W
series ind = $coeff(const) + $coeff(age)*age + \
          $coeff(educ)*educ + $coeff(kids)*kids + $coeff(mtr)*mtr
series lambda = dnorm(ind)/cnorm(ind)
ols l_wage X lambda
```

The variables for the regression are put into the list `X` and those for the selection equation are put into `W`. The `dnorm` and `cnorm` functions return the normal density and normal cumulative density evaluated at the argument, respectively. The results are:

```
OLS estimates using the 428 observations 1–428
Dependent variable: l_wage

             Coefficient   Std. Error    t-ratio   p-value

const         0.810542     0.494472       1.6392   0.1019
educ          0.0584579    0.0238495      2.4511   0.0146
exper         0.0163202    0.00399836     4.0817   0.0001
lambda       −0.866439     0.326986      −2.6498   0.0084

Mean dependent var     1.190173   S.D. dependent var   0.723198
Sum squared resid      187.0967   S.E. of regression   0.664278
R^2                    0.162231   Adjusted R^2         0.156304
F(3, 424)              27.36878   P-value(F)           3.38e–16
Log-likelihood        −430.2212   Akaike criterion     868.4424
Schwarz criterion      884.6789   Hannan–Quinn         874.8550
```

Notice that the estimated coefficient of the inverse Mill's ratio is statistically significant, implying that there is a selection bias in the least squares estimator. Also, the estimated return to education has fallen from approximately 11% (which is inconsistently estimated) to approximately 6%. Unfortunately, the usual standard errors do not account for the fact that the inverse Mills ratio is itself an estimated value and so they are not technically correct. To obtain the correct standard errors, you will use **gretl**'s built-in `heckit` command.

The `heckit` command syntax is



```
heckit

Arguments:    depvar indepvars ; selection equation
Options:      --quiet (suppress printing of results)
              --robust (QML standard errors)
              --two-step (perform two-step estimation)
              --vcv (print covariance matrix)
              --verbose (print extra output)
Example:      heckit y 0 x1 x2 ; ys 0 x3 x4
              See also heckit.inp
```

In terms of our example the generic syntax will be

```
heckit y const x2 x3 ... xk; z const w2 w3 ... ws --two-step
```

where `const x2 ... xk` are the k independent variables for the regression and `const w2 .... ws` are the s independent variables for the selection equation. In this example, we've used the two-step option which mimics the manual procedure employed above, but returns the correct standard errors.

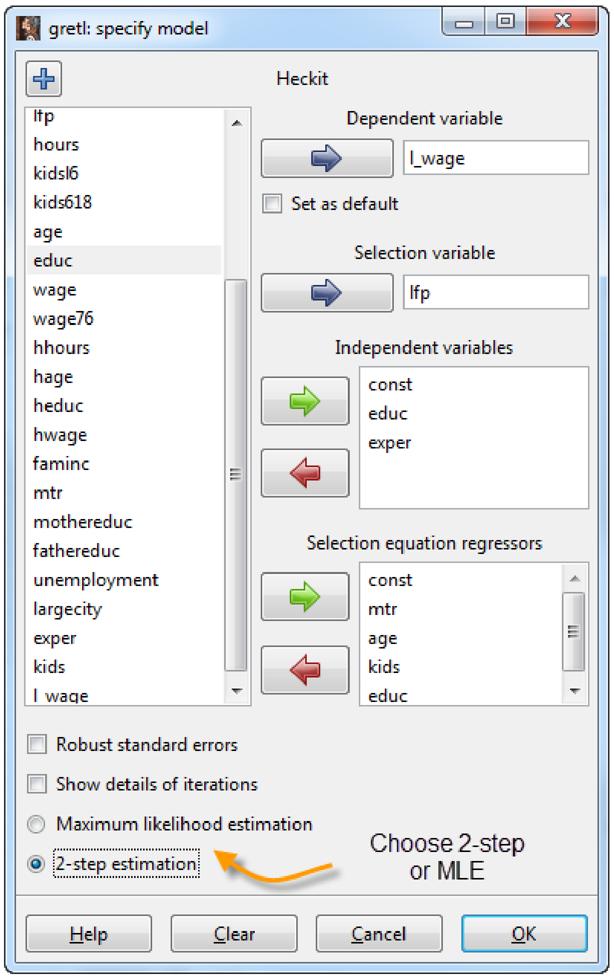
```
heckit l_wage X ; lfp W --two-step
```

Again, we've used the results from the `list` function, which put the independent variables for the regression into `X` and the variables for the selection equation into `W`.

The results appear below in Table 16.3. Notice that in this model, the return to another year of schooling is about 5.8%. The parameter on the inverse Mills ratio is significant, which is evidence of selection bias.

To use the pull-down menus, select `Model>Nonlinear models>Heckit` from **gretl**'s main window. This will reveal the dialog shown in figure 16.5. Enter `lwage` as the dependent variable and the indicator variable `lfp` as the selection variable. Then enter the desired independent variables for the regression and selections equations. Finally, select the *2-step estimation* button at the bottom of the dialog box and click **OK**.

You will notice that the coefficient estimates are identical to the ones produced manually above. However, the standard errors, which are now consistently estimated, have changed. The t-ratio



gretl: specify model

Heckit

Dependent variable: l_wage

[ ] Set as default

Selection variable: lfp

Independent variables: const, educ, exper

Selection equation regressors: const, mtr, age, kids, educ

[ ] Robust standard errors

[ ] Show details of iterations

( ) Maximum likelihood estimation

[✓ 2-step estimation]

Figure 16.5: Choose `Model>Nonlinear models>Heckit` from **gretl**'s main window to reveal the dialog box for Heckit.

Two-step Heckit estimates using the 428 observations 1–428
Dependent variable: l_wage
Selection variable: lfp

	Coefficient	Std. Error	z-stat	p-value
const	0.810542	0.610798	1.3270	0.1845
educ	0.0584579	0.0296354	1.9726	0.0485
exper	0.0163202	0.00420215	3.8838	0.0001
lambda	−0.866439	0.399284	−2.1700	0.0300
Selection equation				
const	1.19230	0.720544	1.6547	0.0980
mtr	−1.39385	0.616575	−2.2606	0.0238
age	−0.0206155	0.00704470	−2.9264	0.0034
kids	−0.313885	0.123711	−2.5372	0.0112
educ	0.0837753	0.0232050	3.6102	0.0003

Mean dependent var	1.190173	S.D. dependent var	0.723198
$\hat{\sigma}$	0.932559	$\hat{\rho}$	−0.929098

Total observations: 753
Censored observations: 325 (43.2%)

Table 16.3: Two-step Heckit results.

of the coefficient on the inverse Mills ratio, $\hat{\lambda}$, has fallen to −2.17, but it is still significant at the 5% level. **Gretl** also produces the estimates of the selection equation, which appear directly below those for the regression.

16.10 Using R for Qualitative Choice Models

R is a programming language that can be very useful for estimating sophisticated econometric models. In fact, many statistical procedures have been written for **R**. Although **gretl** is very powerful, there are still many things that it won't do out of the box. The ability to export **gretl** data into **R** makes it possible to do some very fancy econometrics with relative ease. The proliferation of new procedures in **R** comes as some cost though. Although the packages that are published at CRAN (http://cran.r-project.org/) have met certain standards, there is no assurance that any of them do what they intend correctly. **Gretl**, though open source, is more controlled in its approach. There are two major programmers that vet what **gretl** does and an active community of users that experiment and test the outcomes. **Gretl** users can add functions, just as **R** users do, but the basic set of **gretl** routines have been tested against known results. Mistakes occur, but they are usually rooted out and fixed by Professor Cottrell or Lucchetti before most people notice.

To use any of the **R** packages, you'll need a copy of **R**, internet access, and the ability to install these to a local drive. A **package** is just a collection of programs and documentation written in **R** that make it easier to use for specific tasks. In the appendix D we use a package to read in data saved in Stata's format and below another to estimate qualitative choice models using a Bayesian approach.

The **R** software package that is used to estimate qualitative choice models is called **MCMC-pack**. **MCMCpack** stands for Markov Chain Monte Carlo package and it can be used to estimate every qualitative choice model in this chapter. We will just use it to estimate multinomial logit, conditional logit, and ordered probit. So, let's take a quick look at **MCMCpack** and what it does.

The Markov chain Monte Carlo (MCMC) methods are basic numerical tools that are often used to compute Bayesian estimators. In Bayesian analysis one combines what one already knows (called the *prior*) with what is observed through the sample (the likelihood function) to estimate the parameters of a model. The information available from the sample information is contained in the likelihood function; this is the same likelihood function discussed in your book. If we tell the Bayesian estimator that everything we know is contained in the sample, then the two estimators are essentially the same. That is what happens with **MCMCpack** under its defaults.

The biggest difference is in how the two estimators are computed. The MLE is computed using numerical optimization of the likelihood function, whereas **MCMCpack** uses simulation to accomplish virtually the same thing. See Lancaster (2004) or Koop (2003) for an introduction to Bayesian methods and its relationship to maximum likelihood.

The MCMC creates a series of estimates–called a (Markov) chain–and that series of estimates has an empirical probability distribution. Under the proper circumstances the probability distribution of the chain will mimic that of the MLE. Various features of the chain can be used as estimates. For instance, the sample mean is used by **MCMCpack** to estimate the parameters of the multinomial logit model. **MCMCpack** uses variation within the chain to compute the MLE variance covariance matrix, which is produced using the `summary` command.

One piece of information that you must give to **MCMCpack** is the desired length of your Markov chain. In the examples here, I chose 20,000, which is the number used in the sample programs included in **MCMCpack**. Longer chains tend to be more accurate, but take longer to compute. This number gets us pretty close to the MLEs produced by **gretl** and by Stata.

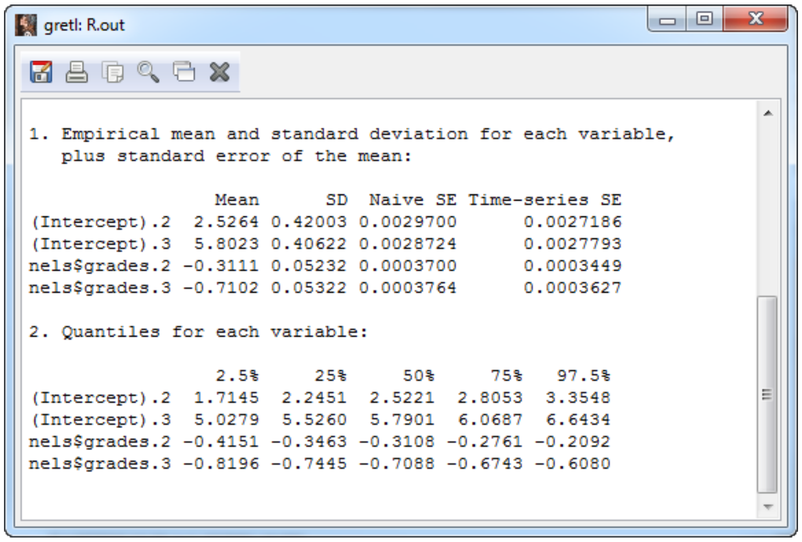
16.10.1 Multinomial Logit

Open the *nels_small.gdt* data set and then open a new R script. The latter is done using `File>Script files>New script>R script`. This opens a window called *edit R commands* as is shown in Figure D.2. In the box, type in the following program The program code to estimate the multinomial logit example is shown below:

```
nels <- gretldata
library(MCMCpack)
posterior <- MCMCmnl(nels$psechoice ~ nels$grades, mcmc=20000)
summary(posterior)
```

The first line converts the data contained in `gretldata`, which is what **gretl** loads into **R** by default, to `nels`. Then load the **MCMCpack** using the library command. A warning is in order. If you have not installed **MCMCpack**, then this will cause **gretl** to crash. Be sure to save anything of importance in **gretl** before trying this. Refer to sections D.3 and D.4 for a brief introduction to packages and reading Stata datasets in **R**.

The next line calls the multinomial logit estimator (`MCMCmnl`). The first argument of `MCMCmnl` is the dependent variable `nels$psechoice`, followed by a ~, and then the independent variable `nels$grades`. The last argument tells **R** how many simulated values to compute, in this case 20,000. The results of the simulation are stored in the object called `posterior`. Posterior is the name given in the Bayesian literature to the probability distribution of the estimates. The mean or median of this distribution is used as a point estimate (vis-a-vis the MLE). The last line of the program requests the summary statistics from the Markov chain. The results appear in Figure 16.6 In the MNL model, the estimates from **MCMCpack** are a little different from those produced by



```
1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                  Mean      SD  Naive SE Time-series SE
(Intercept).2   2.5264 0.42003 0.0029700      0.0027186
(Intercept).3   5.8023 0.40622 0.0028724      0.0027793
nels$grades.2  -0.3111 0.05232 0.0003700      0.0003449
nels$grades.3  -0.7102 0.05322 0.0003764      0.0003627

2. Quantiles for each variable:

                  2.5%     25%     50%     75%   97.5%
(Intercept).2   1.7145  2.2451  2.5221  2.8053  3.3548
(Intercept).3   5.0279  5.5260  5.7901  6.0687  6.6434
nels$grades.2  -0.4151 -0.3463 -0.3108 -0.2761 -0.2092
nels$grades.3  -0.8196 -0.7445 -0.7088 -0.6743 -0.6080
```

Figure 16.6: Multinomial logit results from the MCMCmnl estimator in **R**

gretl, but they are reasonably close. The quantiles are useful for several reasons. As you can see, the median is actually closer to the MLE than the mean of the posterior distribution. Also, 95% confidence sets can be gleaned from the 2.5% and 97.5% quantiles.

16.10.2 Conditional Logit

In this example I'll show you how to use **MCMCpack** in **R** to estimate the conditional logit model.

The first order of business is to get the data into a format that suits **R**. This part is not too pretty, but it works. The data are read into **gretl** from the *cola.gdt* data. Launching **R** from within **gretl** transfers the data into **R**, where it is referred to as `gretldata`. It is renamed `cola` and then `attach(cola)` is used to make referencing the variables easier to do. The `attach(cola)` statement is not necessary, but including it will enable you to call each of the variables in the object `cola` by name. For example, `cola$price` refers to the variable named `price` in the object named `cola`. Once `cola` is attached, `cola$price` can be referred to simply as `price`.

The data in the original *cola.gdt* dataset are arranged

```
> cola[1:12,]
   id choice price feature display
1   1      0  1.79       0       0
2   1      0  1.79       0       0
3   1      1  1.79       0       0
4   2      0  1.79       0       0
5   2      0  1.79       0       0
6   2      1  0.89       1       1
7   3      0  1.41       0       0
8   3      0  0.84       0       1
9   3      1  0.89       1       0
10  4      0  1.79       0       0
11  4      0  1.79       0       0
12  4      1  1.33       1       0
```

The **MCMCpack** routine in **R** wants to see it as

```
id  bev.choice pepsi.price sevenup.price  coke.price
 1      3         1.79          1.79        1.79
 2      3         1.79          1.79        0.89
 3      3         1.41          0.84        0.89
 4      3         1.79          1.79        1.33
```

where each line represents an individual, recording his choice of beverage and each of the three prices he faces. The goal then is to reorganize the original dataset so that the relevant information for each individual, which is contained in 3 lines, is condensed into a single row. To simplify the example, I dropped the variables not being used.

Most of the program below is devoted to getting the data into the proper format. The line

```
pepsi.price <- price[seq(1,nrow(cola),by=3)]
```

creates an object called `pepsi.price`. The new object consists of every third observation in `price`, starting with observation 1. The square brackets [ ] are used to take advantage of **R**'s powerful indexing ability. The function `seq(1,nrow(cola),by=3)` creates a seqence of numbers that start at 1, increment by 3, and extends until the last row of `cola` i.e., [1 3 6 9 ... 5466]. When used inside the square brackets, these numbers constitute an index of the object's elements that you want to grab. In this case the object is `price`. The `sevenup.price` and `coke.price` lines do the same thing, except their sequences start at 2 and 3, respectively.

The next task is to recode the alternatives to a single variable that takes the value of 1, 2 or 3 depending on a person's choice. For this I used the same technique.

```
pepsi <- choice[seq(1,nrow(cola),by=3)]
sevenup <- 2*choice[seq(2,nrow(cola),by=3)]
coke <- 3*choice[seq(3,nrow(cola),by=3)]
```

The first variable, `pepsi`, takes every third observation of `choice` starting at the first row. The variable will contain a one if the person chooses Pepsi and a zero otherwise since this is how the variable `choice` is coded in the data file. The next variable for Seven-Up starts at 2 and the sequence again increments by 3. Since Seven-Up codes as a 2 the ones and zeros generated by the sequence get multiplied by 2 (to become 2 or 0). Coke is coded as a 3 and its sequence of ones and zeros is multiplied by 3. The three variables are combined into a new one called bev.choice that takes the value of 1,2, or 3 depending on a person's choice of Pepsi, Seven-Up, or Coke.

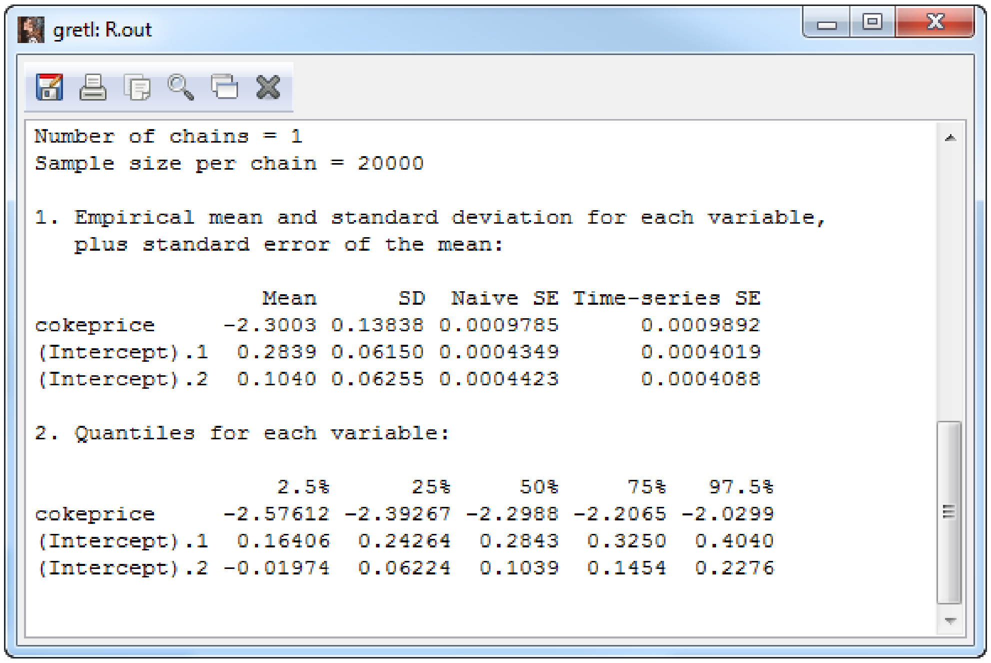
Once the data are arranged, load the **MCMCpack** library and use `MCMCmnl` to estimate the model. The conditional logit model uses choice specific variables. For `MCMCmnl` these choice-specific covariates have to be entered using a special syntax: `choicevar(cvar,"var","choice")` where `cvar` is the name of a variable in data, `var` is the name of the new variable to be created, and `choice` is the level of `bev.choice` that `cvar` corresponds to.

```
cola <- gretldata
cola[1:12,]

attach(cola)
pepsi.price <- price[seq(1,nrow(cola),by=3)]
sevenup.price <- price[seq(2,nrow(cola),by=3)]
coke.price <- price[seq(3,nrow(cola),by=3)]

pepsi <- choice[seq(1,nrow(cola),by=3)]
sevenup <- 2*choice[seq(2,nrow(cola),by=3)]
coke <- 3*choice[seq(3,nrow(cola),by=3)]

```

```
bev.choice <- pepsi + sevenup + coke

posterior <- MCMCmnl(bev.choice ~
               choicevar(coke.price, "cokeprice", "3") +
               choicevar(pepsi.price, "cokeprice", "1") +
               choicevar(sevenup.price, "cokeprice", "2"),
               mcmc=20000, baseline="3")
summary(posterior)

```

In this example, we specified that we want to normalize the conditional logit on the coke choice; this is done using the `baseline="3"` option in `MCMCmnl`.

The results appear in Figure 16.7.



```
Number of chains = 1
Sample size per chain = 20000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                 Mean      SD  Naive SE Time-series SE
cokeprice     -2.3003 0.13838 0.0009785      0.0009892
(Intercept).1  0.2839 0.06150 0.0004349      0.0004019
(Intercept).2  0.1040 0.06255 0.0004423      0.0004088

2. Quantiles for each variable:

                  2.5%      25%     50%     75%   97.5%
cokeprice     -2.57612 -2.39267 -2.2988 -2.2065 -2.0299
(Intercept).1  0.16406  0.24264  0.2843  0.3250  0.4040
(Intercept).2 -0.01974  0.06224  0.1039  0.1454  0.2276
```

Figure 16.7: Conditional logit results from the MCMCoprobit estimator in **R**

16.10.3 Ordered Probit

MCMCpack can also be used to estimate the ordered probit model. It is very easy and the results you get using the Markov chain Monte Carlo simulation method are **very** similar to those from maximizing the likelihood. In principle the maximum likelihood and the simulation estimator used by **MCMCpack** are asymptotically equivalent.[1] The difference between **MCMCpack** and

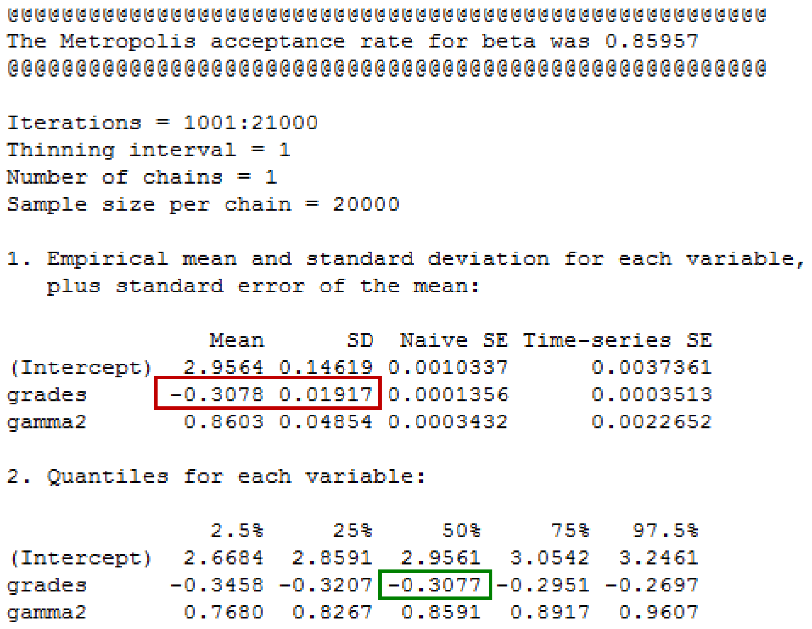
[1] Of course, if you decide to use more information in your prior then they can be substantially different.

Stata's MLE results occurs because the sample sizes for the datasets used is small.

```
nels <- gretldata
attach(nels)

library(MCMCpack)
  posterior <- MCMCoprobit(psechoice ~ grades, mcmc=20000)
  summary(posterior)
```

The first line converts the generic **gretldata** data frame that is loaded when you launch R from within **gretl**. The second line creates the data object called **nels**. The **attach(nels)** statement allows you to refer to the variables in **nels** data frame directly by their names.

The next line loads **MCMCpack** into **R**. Then the ordered probit estimator (MCMCoprobit) is called. The first argument of MCMCoprobit is the dependent variable **psechoice**, followed by a $\sim$, and then the independent variable **grades**. The last argument tells **R** how many simulated values to compute, in this case 20,000. The results of the simulation are stored in the object called **posterior**. The mean or median of this distribution is used as your point estimate (vis-a-vis the MLE). The last line of the program requests the summary statistics from the simulated values of the parameters. The results appear in Figure 16.10.3. One important difference between **MCMCpack**



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate for beta was 0.85957
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Iterations = 1001:21000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 20000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

               Mean      SD  Naive SE Time-series SE
(Intercept)  2.9564 0.14619 0.0010337      0.0037361
grades      -0.3078 0.01917 0.0001356      0.0003513
gamma2       0.8603 0.04854 0.0003432      0.0022652

2. Quantiles for each variable:

               2.5%     25%     50%     75%   97.5%
(Intercept)  2.6684  2.8591  2.9561  3.0542  3.2461
grades      -0.3458 -0.3207 -0.3077 -0.2951 -0.2697
gamma2       0.7680  0.8267  0.8591  0.8917  0.9607
```

Figure 16.8: Ordered probit results from the MCMCoprobit estimator in **R**

and the MLE is in how the results are reported. The model as specified in your textbook contains no intercept and 2 thresholds. To include a separate intercept would cause the model to be perfectly collinear. In **MCMCpack**, the default model includes an intercept and hence can contain only one threshold.

The 'slope' coefficient β, which is highlighted in Figure 16.10.3, is virtually the same as that we obtained using the MLE in **gretl**. The other results are also similar and are interpreted like the ones produced in **gretl**. The intercept in **MCMCpack** is equal to $-\mu_1$. The second cut-off in *POE4*'s no-intercept model is $\mu_2 = -(Intercept - \gamma_2)$, where γ_2 is the single threshold in the **MCMCpack** specification.

The standard errors are comparable and you can see that they are equivalent to 3 or 4 decimal places to those from the MLE.

16.11 Script

Frist, here are all of the functions used in this chapter. You'll need to run these before using the second part of the script.

Functions used in chapter 16

```
set echo off

function matrix me1(matrix *param, list x, scalar *q)
    matrix p = lincomb(x, param)
    return dnorm(p)*param[q]
end function

function matrix ame(matrix *param, list x)
    matrix p = lincomb(x, param)
    matrix me_matrix = dnorm(p)*param'
    matrix amfx = meanc(me_matrix)
    printf "\nThe average marginal effects are %8.4f \n", amfx
    return me_matrix
end function

function matrix me_at(matrix *param, matrix *xx, scalar *q)
    scalar p = xx*param
    return dnorm(p)*param[q]
end function

function matrix mep(matrix *param, matrix *xx, scalar *q)
    scalar p = xx*param
    return cnorm(p)
end function

function matrix dlogist(matrix *param, list x)
    matrix p = lincomb(x, param)
    matrix d = exp(-p)./(1.+exp(-p)).^2
    return d
end function

function matrix clogist(matrix *param, list x)
```

```
    matrix p = lincomb(x, param)
    matrix c = exp(p)./(1.+exp(p))
    return c
end function

function matrix ame_l(matrix *param, list x)
    matrix p = lincomb(x, param)
    matrix me_matrix = dlogist(&param,x)*param'
    matrix amfx = meanc(me_matrix)
    printf "\nThe average marginal effects are %8.4f \n", amfx
    return me_matrix
end function

function list mlogitprob(series y, list x, matrix theta)
    list probs = null
    matrix X = { x }
    scalar j = max(y)
    scalar k = cols(X)
    matrix b = mshape(theta,k,j-1)
    matrix tmp = X*b
    series den = (1 + sumr(exp(tmp)))

    loop for i=1..j --quiet
        if i = 1
            series p$i = 1/den
        else
            scalar q = i - 1
            series num = exp(X[q,]*b[,q])
            series p$i=num/den
        endif
        list probs += p$i
    endloop
    return probs
end function

function matrix mlogitprob_at(series y, matrix X, matrix theta)
    matrix probs = {}
    scalar j = max(y)
    scalar k = cols(X)
    matrix b = mshape(theta,k,j-1)
    matrix tmp = X*b
    scalar den = (1 + sumr(exp(tmp)))

    loop for i=1..j --quiet
        if i = 1
            scalar  p$i = 1/den
        else
            scalar q = i - 1
            scalar num = exp(X*b[,q])
            scalar p$i=num/den
        endif
        matrix probs = probs ~ p$i
    endloop
    return probs
end function

function mlogitlogprobs(series y, matrix X, matrix theta)
  scalar n = max(y)
  scalar k = cols(X)
  matrix b = mshape(theta,k,n)
  matrix tmp = X*b
  series ret = -ln(1 + sumr(exp(tmp)))
  loop for i=1..n --quiet
    series x = tmp[,i]
    ret += (y=$i) ? x : 0
  end loop
return series ret
end function
```

Once the functions have been run, the script below should produce all of the results in the chapter.

```
# run the function.inp first!
open "@gretldir\data\poe\transport.gdt"
set echo off
summary --simple
ols auto const time
probit auto const dtime --robust
matrix covmat = $vcv
scalar i_20 = $coeff(const)+$coeff(dtime)*2
scalar d_20 = dnorm(i_20)*$coeff(dtime)
printf "\nThe value of the index for dtime=20 is %8.4f\n \
the probability of choosing auto is %8.4f \n", i_20, d_20

# probit marginal effects and average mfx
series me = dnorm($coeff(const)+$coeff(dtime)*dtime)*$coeff(dtime)
scalar amf = mean(me)
summary me --simple

# probit average mfx using function
list x = const dtime
probit auto x
matrix coef = $coeff
matrix me_probit = ame(&coef, x)

# using Delta method to get standard errors for mfx
open "@gretldir\data\poe\transport.gdt"
summary --simple
list x = const dtime
probit auto x
```

```
matrix coef = $coeff
matrix covmat = $vcv
scalar q = 2
series mfx = me1(&coef, x, &q)
matrix amfx = mean(mfx)
matrix jac = fdjac(coef, me1(&coef, x, &q))
matrix mjac = meanc(jac)
matrix variance = qform(mjac,covmat)
matrix se = sqrt(variance)
printf "\nThe average marginal effect of dtime = %6.4f with\
 standard error %6.4f \n", amfx, se

# confidence interval for average mfx
scalar ub = amfx + critical(t,$df,0.025)*se
scalar lb = amfx - critical(t,$df,0.025)*se
printf "\nThe 95%% confidence interval for the AME of dtime\
 is (%6.4f, %6.4f) \n",lb,ub

# marginal effects and std errors at specific points
open "@gretldir\data\poe\transport.gdt"
summary --simple
list x = const dtime
probit auto x
matrix coef = $coeff
matrix covmat = $vcv
scalar q = 2
matrix xx = { 1, 2 }
matrix mfx = me_at(&coef, &xx, &q)
matrix jac = fdjac(coef, me_at(&coef, &xx, &q))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
printf "\nThe marginal effect of dtime when dtime=2 is %6.4f with\
standard error %6.4f \n", mfx, se

scalar ub = mfx + critical(z,0.025)*se
scalar lb = mfx - critical(z,0.025)*se
printf "\nThe 95%% confidence interval the ME with dtime=2 is\
(%6.4f, %6.4f) \n",lb,ub

# predicted probability and its confidence interval
probit auto x
scalar q = 2
matrix coef = $coeff
matrix xx = { 1, 3 }
matrix mp = mep(&coef, &xx, &q)
matrix jac_3 = fdjac(coef, mep(&coef, &xx, &q))
matrix variance = qform(jac_3,covmat)
matrix se = sqrt(variance)
printf "\nThe probability of driving when dtime=3 is %6.4f with\
standard error %6.4f \n", mp, se

```

```
scalar ub = mp + critical(z,0.025)*se
scalar lb = mp - critical(z,0.025)*se
printf "\nThe 95%% confidence interval the Prob with dtime=3\
is (%6.4f, %6.4f) \n",lb,ub

# comparing probit, logit, ols
open "@gretldir\data\poe\coke.gdt"
list x = pratio disp_pepsi disp_coke const
probit coke x --quiet
modeltab add
logit coke x --quiet
modeltab add
ols coke x --robust
modeltab add
modeltab show

# test hypotheses with probit
probit coke x
restrict
b[3]+b[4]=0
end restrict

probit coke x
restrict
b[3]=0
b[4]=0
end restrict

series c_p = disp_pepsi-disp_coke
probit coke x
scalar llu = $lnl
probit coke const pratio c_p
scalar llr = $lnl
scalar lr = 2*(llu-llr)
pvalue X 1 lr

# average mfx with logit
list x = const pratio disp_coke disp_pepsi
logit coke x
matrix coef = $coeff
ame_l(&coef,x)

# average mfx with probit
probit coke x
matrix coef = $coeff
ame(&coef,x)

# correlation among predictions
probit coke x --quiet
series pp = $yhat
logit coke x --quiet
```

```
series pl = $yhat
ols coke x --quiet
series po = $yhat
corr pp pl po

# Multinomial Logit
open "@gretldir\data\poe\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
matrix theta = $coeff
list n = mlogitprob(psechoice, x, theta)
smpl 1 12
print n --byobs
smpl full

# Average marginal effects
rename p1 p01
rename p2 p02
rename p3 p03

series grade1 = grades+1
list x1 = const grade1
list n1 = mlogitprob(psechoice, x1, theta)
series d1 = p1-p01
series d2 = p2-p02
series d3 = p3-p03
summary d* --simple

# mnl predictions at points
open "@gretldir\data\poe\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
matrix theta = $coeff
matrix Xm = {1 , quantile(grades,.50)}
matrix p50 = mlogitprob_at(psechoice, Xm, theta)
matrix Xm = {1 , quantile(grades,.05)}
matrix p05 = mlogitprob_at(psechoice, Xm, theta)
printf "\nThe predicted probabilities for student\
grades=%6.2f is %8.4f\n",quantile(grades,.05), p05
printf "\nThe predicted probabilities for student\
grades=%6.2f is %8.4f\n",quantile(grades,.50), p50

# mnl marginal effects at points
open "@gretldir\data\poe\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
matrix theta = $coeff
scalar q50 = quantile(grades,.50)
matrix Xm = {1 , q50-0.5}
matrix p0 = mlogitprob_at(psechoice, Xm, theta)
matrix Xm = {1 , q50+0.5}
matrix p1 = mlogitprob_at(psechoice, Xm, theta)
matrix me = p1-p0
printf "\nThe marginal effect of grades for student\
grades =%6.2f is %8.4f\n",median(grades), me

scalar q05 = quantile(grades,.05)
matrix Xm = {1 , q05-0.5}
matrix p0 = mlogitprob_at(psechoice, Xm, theta)
matrix Xm = {1 , q05+0.5}
matrix p1 = mlogitprob_at(psechoice, Xm, theta)
matrix me = p1-p0
printf "\nThe marginal effect of grades for student\
  grades=%6.2f is %8.4f\n", q05, me

# mnl logit with user written likelihood
open "@gretldir\data\poe\nels_small.gdt"
psechoice = psechoice-1  # dep. var. must be 0-based
list x = const grades
smpl full
matrix X = { x }
scalar k = cols(X)
matrix theta = zeros(2*k, 1)
mle loglik = mlogitlogprobs(psechoice,X,theta)
    params theta
end mle --hessian

# conditional logit
open "@gretldir\data\poe\cola2.gdt"
list y = pepsi sevenup coke
list x =  pr_pepsi pr_7up pr_coke
matrix theta = {-1.19, .283, .1038}

mle lln = clprobs(y, x, theta)
    params theta
end mle

#Ordered Probit
open "@gretldir\data\poe\nels_small.gdt"
probit psechoice const grades

# Marginal effects on probability of going to 4 year college
k = $ncoeff
matrix b = $coeff[1:k-2]
mu1 = $coeff[k-1]
mu2 = $coeff[k]

matrix X = {6.64}
matrix Xb = X*b
P3a = pdf(N,mu2-Xb)*b

matrix X = 2.635
matrix Xb = X*b
P3b = pdf(N,mu2-Xb)*b

printf "\nFor the median grade of 6.64, the marginal effect\
is %.4f\n", P3a
printf "\nFor the 5th percentile grade of 2.635, the marginal\
effect is %.4f\n", P3b

# Poisson Regression -- means and marginal effect
open "@gretldir\data\poe\olympics.gdt"
smpl year = 88 --restrict
logs pop gdp
poisson medaltot const l_pop l_gdp
scalar m1 = exp($coeff(const)+$coeff(l_pop)*ln(quantile(pop,0.5)) \
                   +$coeff(l_gdp)*ln(quantile(gdp,0.5)))
scalar m2 = exp($coeff(const)+$coeff(l_pop)*ln(quantile(pop,0.75)) \
                   +$coeff(l_gdp)*ln(quantile(gdp,0.5)))
scalar mft = exp($coeff(const)+$coeff(l_pop)*ln(quantile(pop,0.5)) \
                   +$coeff(l_gdp)*ln(quantile(gdp,0.5)))*$coeff(l_gdp)
printf "\nMarginal Effect at the medians is %.3f\n",mft
printf "\nAverage medal total for median gdp and pop is %.3f\n",m1
printf "\nAverage medal total for median gdp and 75th quantile\
pop is %.3f\n",m2

#Tobit
open "@gretldir\data\poe\mroz.gdt"
list xvars = const educ exper age kidsl6
tobit hours xvars
scalar H_hat = $coeff(const)+$coeff(educ)*mean(educ) \
                          +$coeff(exper)*mean(exper) \
                          +$coeff(age)*mean(age)+$coeff(kidsl6)*1
scalar z = cnorm(H_hat/$sigma)
scalar me_educ = z*$coeff(educ)
printf "\nThe computed scale factor = %6.5g\nand marginal effect of\
another year of schooling = %5.5g.\n", z, me_educ

matrix beta = $coeff
matrix X = { xvars }
matrix meanx = meanc(X)
matrix meanx[1,5]=1
scalar h_hat=meanx*beta
printf "\nTwo ways to compute a prediction get %8.4f and %8.4f\n", h_hat, H_hat

smpl hours > 0 --restrict
ols hours xvars
smpl --full
ols hours xvars

# tobit simulation
nulldata 200
series xs = 20*uniform()
```

```
list x = const xs
series ys = -9 + 1*xs
loop 1000 --progressive --quiet
   series y = ys + normal(0,4)
   series yc = (y > 0) ? y : 0
   ols y x
   ols yc x
   series w = (yc>0)
   wls w yc x
   tobit yc x
endloop

#Heckit
open "@gretldir\data\poe\mroz.gdt"

series kids = (kidsl6+kids618>0)
logs wage

list X = const educ exper
list W = const mtr age kids educ

probit lfp W
series ind = $coeff(const) + $coeff(age)*age + \
           $coeff(educ)*educ + $coeff(kids)*kids + $coeff(mtr)*mtr
series lambda = dnorm(ind)/cnorm(ind)
ols l_wage X lambda

heckit l_wage X ; lfp W --two-step
```

Appendix A

Gretl Commands

Estimation

`ar`	Autoregressive estimation	`ar1`	AR(1) estimation
`arbond`	Arellano-Bond	`arch`	ARCH model
`arima`	ARMA model	`biprobit`	Bivariate probit
`dpanel`	Dynamic panel models	`duration`	Duration models
`equation`	Define equation within a system	`estimate`	Estimate system of equations
`garch`	GARCH model	`gmm`	GMM estimation
`heckit`	Heckman selection model	`hsk`	Heteroskedasticity-corrected estimates
`intreg`	Interval regression model	`kalman`	Kalman filter
`lad`	Least Absolute Deviation estimation	`logistic`	Logistic regression
`logit`	Logit regression	`mle`	Maximum likelihood estimation
`mpols`	Multiple-precision OLS	`negbin`	Negative Binomial regression
`nls`	Nonlinear Least Squares	`ols`	Ordinary Least Squares
`panel`	Panel Models	`poisson`	Poisson estimation
`probit`	Probit model	`quantreg`	Quantile regression
`system`	Systems of equations	`tobit`	Tobit model
`tsls`	Instrumental variables regression	`var`	Vector Autoregression
`vecm`	Vector Error Correction Model	`wls`	Weighted Least Squares

Tests

`add`	Add variables to model	`adf`	Augmented Dickey-Fuller test
`chow`	Chow test	`coeffsum`	Sum of coefficients
`coint`	Engle-Granger cointegration test	`coint2`	Johansen cointegration test
`cusum`	CUSUM test	`difftest`	Nonparametric test for differences
`hausman`	Panel diagnostics	`kpss`	KPSS stationarity test
`leverage`	Influential observations	`levinlin`	Levin-Lin-Chu test
`meantest`	Difference of means	`modtest`	Model tests
`normtest`	Normality test	`omit`	Omit variables
`qlrtest`	Quandt likelihood ratio test	`reset`	Ramseys RESET
`restrict`	Testing restrictions	`runs`	Runs test
`vartest`	Difference of variances	`vif`	Variance Inflation Factors

Transformations

`diff`	First differences	`discrete`	Mark variables as discrete
`dummify`	Create sets of dummies	`lags`	Create lags
`ldiff`	Log-differences	`logs`	Create logs
`orthdev`	Orthogonal deviations	`sdiff`	Seasonal differencing
`square`	Create squares of variables		

Statistics

`anova`	ANOVA	`corr`	Correlation coefficients
`corrgm`	Correlogram	`fractint`	Fractional integration
`freq`	Frequency distribution	`hurst`	Hurst exponent
`mahal`	Mahalanobis distances	`pca`	Principal Components Analysis
`pergm`	Periodogram	`spearman`	Spearmanss rank correlation
`summary`	Descriptive statistics	`xcorrgm`	Cross-correlogram
`xtab`	Cross-tabulate variables		

Dataset

`append`	Append data	`data`	Import from database
`dataset`	Manipulate the dataset	`delete`	Delete variables
`scalar`	Generate a new variable	`info`	Information on data set
`labels`	Print labels for variables	`nulldata`	Creating a blank dataset
`open`	Open a data file	`rename`	Rename variables
`setinfo`	Edit attributes of variable	`setobs`	Set frequency and starting obser-
`setmiss`	Missing value code	`smpl`	Set the sample range
`store`	Save data	`varlist`	Listing of variables

Graphing

`boxplot`	Boxplots	`gnuplot`	Create a gnuplot graph
`graphpg`	Gretl graph page	`qqplot`	Q-Q plot
`rmplot`	Range-mean plot	`scatters`	Multiple pairwise graphs
`textplot`	ASCII plot		

Printing

`eqnprint`	Print model as equation	`modprint`	Print a user-defined model
`outfile`	Direct printing to file	`print`	Print data or strings
`printf`	Formatted printing	`sprintf`	Printing to a string
`tabprint`	Print model in tabular form		

Programming

`break`	Break from loop	`catch`	Catch errors
`clear`		`debug`	Debugging
`elif`	Flow control	`else`	
`end`	End block of commands	`endif`	Flow control
`endloop`	End a command loop	`foreign`	Non-native script
`function`	Define a function	`if`	Flow control
`include`	Include function definitions	`loop`	Start a command loop
`makepkg`	Make function package	`run`	Execute a script
`set`	Set program parameters	`sscanf`	Scanning a string

Utilities

`help`	Help on commands	`modeltab`	The model table
`pvalue`	Compute p-values	`quit`	Exit the program
`shell`	Execute shell commands		

Appendix B

Some Basic Probability Concepts

In this chapter, you learned some basic concepts about probability. Since the actual values that economic variables take on are not actually known before they are observed, we say that they are *random*. Probability is the theory that helps us to express uncertainty about the possible values of these variables. Each time we observe the outcome of a random variable we obtain an observation. Once observed, its value is known and hence it is no longer random. So, there is a distinction to be made between variables whose values are not yet observed (random variables) and those whose values have been observed (observations). Keep in mind, though, an observation is merely one of many possible values that the variables can take. Another draw will usually result in a different value being observed.

A probability distribution is just a mathematical statement about the possible values that our random variable can take on. The probability distribution tells us the relative frequency (or probability) with which each possible value is observed. In their mathematical form probability distributions can be rather complicated; either because there are too many possible values to describe succinctly, or because the formula that describes them is complex. In any event, it is common summarize this complexity by concentrating on some simple numerical characteristics that they possess. The numerical characteristics of these mathematical functions are often referred to as *parameters*. Examples are the mean and variance of a probability distribution. The mean of a probability distribution describes the average value of the random variable over *all* of its possible realizations. Conceptually, there are an infinite number of realizations therefore parameters are not known to us. As econometricians, our goal is to try to estimate these parameters using a finite amount of information available to us. We collect a number of realizations (called a sample) and then estimate the unknown parameters using a *statistic*. Just as a parameter is an unknown numerical characteristic of a probability distribution, a statistic is an observable numerical characteristic of a sample. Since the value of the statistic will be different for each sample drawn, it too is a random variable. The statistic is used to gain information about the parameter.

Expected values are used to summarize various numerical characteristics of a probability distributions. For instance, if X is a random variable that can take on the values 0,1,2,3 and these values occur with probability 1/6, 1/3, 1/3, and 1/6, respectively. The average value or mean of the probability distribution, designated μ, is obtained *analytically* using its expected value.

$$\mu = E[X] = \sum xf(x) = 0 \cdot \frac{1}{6} + 1 \cdot \frac{1}{3} + 2 \cdot \frac{1}{3} + 3 \cdot \frac{1}{6} = \frac{3}{2} \tag{B.1}$$

So, μ is a parameter. Its value can be obtained mathematically if we know the probability density function of the random variable, X. If this probability distribution is known, then there is no reason to take samples or to study statistics! We can ascertain the mean, or average value, of a random variable without every firing up our calculator. Of course, in the real world we only know that the value of X is not known before drawing it and we don't know what the actual probabilities are that make up the density function, $f(x)$. In order to Figure out what the value of μ is, we have to resort to different methods. In this case, we try to infer what it is by drawing a sample and estimating it using a statistic.

One of the ways we bridge the mathematical world of probability theory with the observable world of statistics is through the concept of a *population.* A statistical population is the collection of individuals that you are interested in studying. Since it is normally too expensive to collect information on everyone of interest, the econometrician collects information on a *subset* of this population–in other words, he takes a *sample.*

The population in statistics has an analogue in probability theory. In probability theory one must specify the set of all possible values that the random variable can be. In the example above, a random variable is said to take on 0,1,2, or 3. This set must be complete in the sense that the variable cannot take on any other value. In statistics, the population plays a similar role. It consists of the set that is relevant to the purpose of your inquiry and that is possible to observe. Thus it is common to refer to parameters as describing characteristics of populations. Statistics are the analogues to these and describe characteristics of the sample.

This roundabout discussion leads me to an important point. We often use the words mean, variance, covariance, correlation rather casually in econometrics, but their meanings are quire different depending on whether we are refereing to a probability distribution or a sample. When referring to the analytic concepts of mean, variance, covariance, and correlation we are specifically talking about characteristics of a probability distribution; these can only be ascertained through complete knowledge of the probability distribution functions. It is common to refer to them in this sense as population mean, population variance, and so on. These concepts do not have anything to do with samples or observations!

In statistics we attempt to estimate these (population) parameters using samples and explicit formulae. For instance, we might use the average value of a sample to estimate the average value of the population (or probability distribution).

	Probability Distribution	Sample
mean	$E[X] = \mu$	$\frac{1}{n}\sum x_i = \bar{x}$
variance	$E[X-\mu]^2 = \sigma^2$	$\frac{1}{n-1}\sum(x_i - \bar{x})^2 = s_x^2$

When you are asked to obtain the mean or variance of random variables, make sure you know whether the person asking wants the characteristics of the probability distribution or of the sample. The former requires knowledge of the probability distribution and the later requires a sample.

In **gretl** you are given the facility to obtain sample means, variances, covariances and correlations. You are also given the ability to compute tail probabilities using the normal, t-, F and χ^2 distributions. First we'll examine how to get summary statistics.

Summary statistics usually refers to some basic measures of the numerical characteristics of your sample. In **gretl** , summary statistics can be obtained in at least two different ways. Once your data are loaded into the program, you can select `Data>Summary statistics` from the pull-down menu. Which leads to the output in Figure B.2. The other way to get summary statistics is from

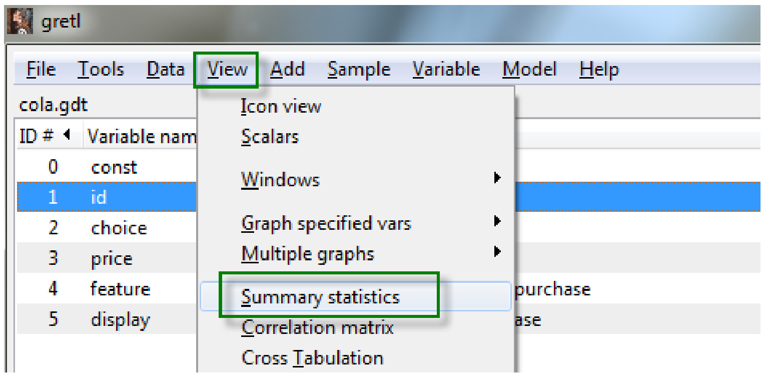
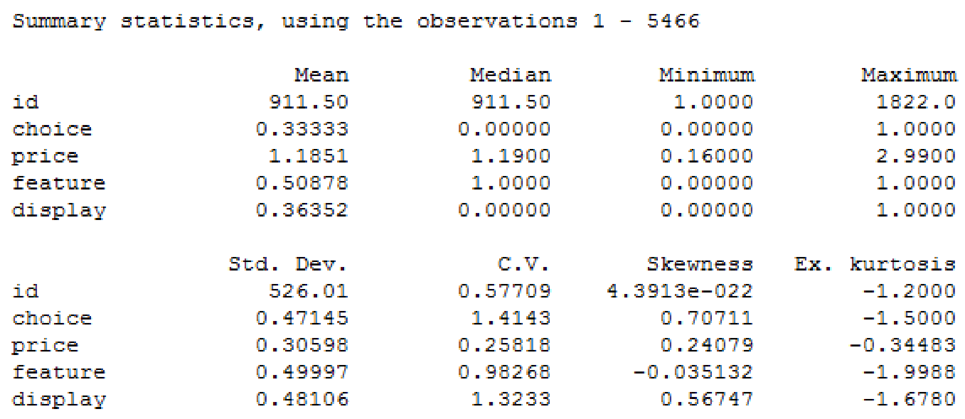


Figure B.1: Choosing summary statistics from the pull-down menu

the console or script. Recall, **gretl** is really just a language and the GUI is a way of accessing that language. So, to speed things up you can do this. Load the dataset and open up a console window. Then type `summary`. This produces summary statistics for all variables in memory. If you just want summary statistics for a subset, then simply add the variable names after `summary`, i.e., `summary x` gives you the summary statistics for the variable `x`.

Gretl computes the sample mean, median, minimum, maximum, standard deviation (S.D.), coefficient of variation (C.V.), skewness and excess kurtosis for each variable in the data set. You may recall from your introductory statistics courses that there are an equal number of observations in your sample that are larger and smaller in value than the median. The standard deviation is the square root of your sample variance. The coefficient of variation is simply the standard deviation divided by the sample mean. Large values of the C.V. indicate that your mean is not very precisely measured. Skewness is a measure of the degree of symmetry of a distribution. If the left tail (tail at small end of the the distribution) extends over a relatively larger range of the variable than the



```
Summary statistics, using the observations 1 - 5466

                     Mean         Median        Minimum        Maximum
id                 911.50         911.50         1.0000         1822.0
choice            0.33333        0.00000        0.00000         1.0000
price              1.1851         1.1900        0.16000         2.9900
feature           0.50878         1.0000        0.00000         1.0000
display           0.36352        0.00000        0.00000         1.0000

                Std. Dev.           C.V.       Skewness   Ex. kurtosis
id                 526.01        0.57709    4.3913e-022        -1.2000
choice            0.47145         1.4143        0.70711        -1.5000
price             0.30598        0.25818        0.24079       -0.34483
feature           0.49997        0.98268      -0.035132        -1.9988
display           0.48106         1.3233        0.56747        -1.6780
```

Figure B.2: Choosing summary statistics from the pull-down menu yields these results.

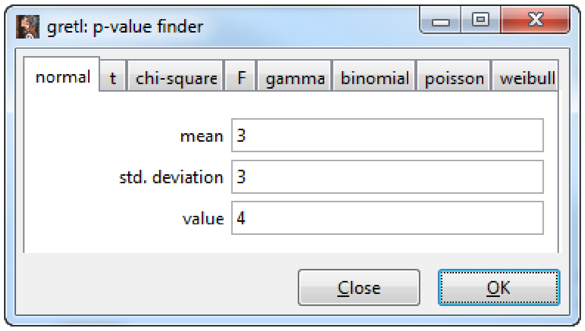
right tail, the distribution is negatively skewed. If the right tail covers a larger range of values then it is positively skewed. Normal and t-distributions are symmetric and have zero skewness. The χ^2_n is positively skewed. Excess kurtosis refers to the fourth sample moment about the mean of the distribution. 'Excess' refers to the kurtosis of the normal distribution, which is equal to three. Therefore if this number reported by **gretl** is positive, then the kurtosis is greater than that of the normal; this means that it is more peaked around the mean than the normal. If excess kurtosis is negative, then the distribution is flatter than the normal.

Sample Statistic	Formula
Mean	$\sum x_i/n = \bar{x}$
Variance	$\frac{1}{n-1}\sum(x_i-\bar{x})^2 = s_x^2$
Standard Deviation	$s = \sqrt{s^2}$
Coefficient of Variation	$s/\bar{x}$
Skewness	$\frac{1}{n-1}\sum(x_i-\bar{x})^3/s^3$
Excess Kurtosis	$\frac{1}{n-1}\sum(x_i-\bar{x})^4/s^4 - 3$

You can also use **gretl** to obtain tail probabilities for various distributions. For example if $X \sim N(3,9)$ then $P(X \geq 4)$ is

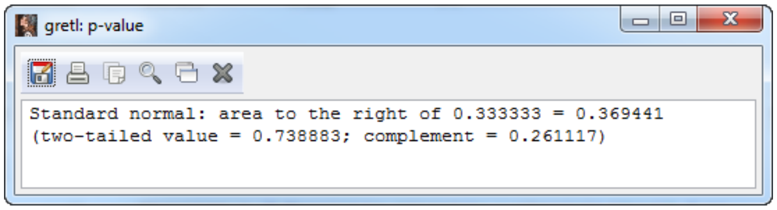
$$P[X \geq 4] = P[Z \geq (4-3)/\sqrt{9}] = P[Z \geq 0.334]\doteq 0.3694 \tag{B.2}$$

To obtain this probability, you can use the `Tools>P-value finder` from the pull-down menu. Then, give **gretl** the value of X, the mean of the distribution and its standard deviation using the dialog box shown in Figure B.3. The result appears in Figure B.4. **Gretl** is using the mean



mean 3
std. deviation 3
value 4

Figure B.3: Dialog box for finding right hand side tail areas of various probability distributions.



```
Standard normal: area to the right of 0.333333 = 0.369441
(two-tailed value = 0.738883; complement = 0.261117)
```

Figure B.4: Results from the p value finder of $P[X \geq 4]$ where $X \sim N(3, 9)$. Note, the area in the tail of this distribution to the right of 4 is .369441.

and standard deviation to covert the normal to a standard normal (i.e., z-score). As with nearly everything in **gretl** , you can use a script to do this as well. First, convert 4 from the $X \sim N(3, 9)$ to a standard normal, $X \sim N(0, 1)$. That means, subtract its mean, 3, and divide by its standard error, $\sqrt{9}$. The result is a scalar so, open a script window and type:

```
scalar z1 = (4-3)/sqrt(9)
```

Then use the `cdf` function to compute the tail probability of `z1`. For the normal cdf this is

```
scalar c1 = 1-cdf(z,z1)
```

The first argument of the `cdf` function, z, identifies the probability distribution and the second, $z1$, the number to which you want to integrate. So in this case you are integrating a standard normal cdf from minus infinity to z1=.334. You want the other tail (remember, you want the probability that Z is greater than 4) so subtract this value from 1.

In your book you are given another example $X \sim N(3, 9)$ then find $P(4 \leq X \leq 6)$ is

$$P[4 \leq X \leq 6] = P[0.334 \leq Z \leq 1] = P[Z \leq 1] - P[Z \leq .33] \tag{B.3}$$

Take advantage of the fact that $P[Z \leq z] = 1 - P[Z > z]$ to obtain use the p-value finder to obtain:

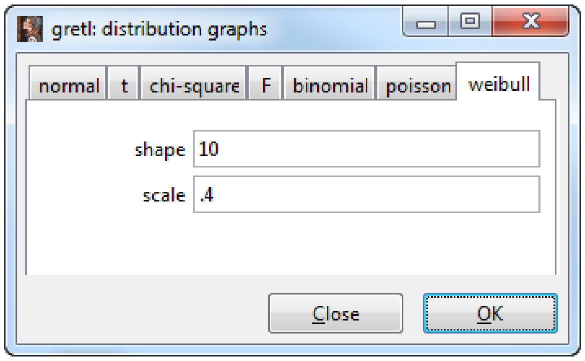
$$(1 - 0.1587) - (1 - 0.3694) = (0.3694 - 0.1587) = 0.2107 \tag{B.4}$$

Note, this value differs slightly from the one given in your book due to rounding error that occurs from using the normal probability table. When using the table, the $P[Z \leq .334]$ was truncated to $P[Z \leq .33]$; this is because your tables are only taken out to two decimal places and a practical decision was made by the authors of your book to forgo interpolation (contrary to what your Intro to Statistics professor may have told you, it is hardly ever worth the effort to interpolate when you have to do it manually). **Gretl**, on the other hand computes this probability out to machine precision as $P[Z \leq \frac{1}{3}]$. Hence, a discrepancy occurs. Rest assured though that these results are, aside from rounding error, the same.

Using the `cdf` function makes this simple and accurate. The script is

```
scalar z1 = (4-3)/sqrt(9)
scalar z2 = (6-3)/sqrt(9)
scalar c1 = cdf(z,z1)
scalar c2 = cdf(z,z2)
scalar area = c2-c1
```

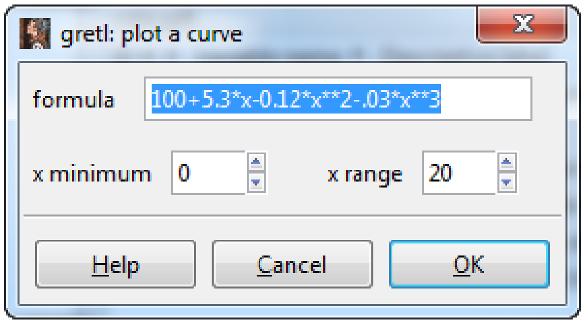
Gretl has a handy new feature that allows you to plot probability distributions. If you've ever wondered what a Weibull(10,0.4) looks like then this is the utility you have waited for. From the main menu choose `Tools>Distribution graphs` from the main menu. The following dialog will appear:



shape 10

scale .4

You can plot normal, t, χ^2, F, binomial, poisson, and weibull probability density functions. Fill in the desired parameters and click **OK**. For the normal, you can also tell **gretl** whether you want the pdf or the cdf. This utility is closely related to another that allows you to plot a curve. The curve plotting dialog is also found in the `Tools` menu.



formula 100+5.3*x-0.12*x**2-.03*x**3

x minimum 0 x range 20

The dialog box allows you to specify the range of the graph as well as the formula, which must be a function of x. Once the graph is plotted you can edit it in the usual way and add additional formulae and lines as you wish. Please note that **gnuplot** uses `**` for exponentiation (raising to a power).

Appendix C

Some Statistical Concepts

The hip data are used to illustrate computations for some simple statistics in your text.

C.1 Summary Statistics

Using a script or operating from the console, open the hip data, *hip.gdt*, and issue the `summary` command. This yields the results shown in Table C.1. This gives you the mean, median, mini-

```
                Summary Statistics, using the observations 1 - 50
                  for the variable 'y' (50 valid observations)

  Mean                    17.158
  Median                  17.085
  Minimum                 13.530
  Maximum                 20.400
  Standard deviation       1.8070
  C.V.                     0.10531
  Skewness                -0.013825
  Ex. kurtosis            -0.66847
```

Table C.1: Summary statistics from the hip data

mum, maximum, standard deviation, coefficient of variation, skewness and excess kurtosis of your variable(s). Once the data are loaded, you can use **gretl**'s language to generate these as well. For instance, `scalar y_bar = mean(y)` yields the mean of the variable `y`. To obtain the sample variance use `scalar y_var = sum((y-y_bar)^2)/($nobs-1)`. The script below can be used to compute other summary statistics as discussed in your text.

```
open "@gretldir\data\poe\hip.gdt"
summary
scalar y_bar = mean(y)
scalar y_var = sum((y-y_bar)^2)/($nobs-1)
scalar y_se = sqrt(y_var)
scalar se_ybar = sqrt(y_var/$nobs)

scalar mu2 = sum((y-y_bar)^2)/($nobs)
scalar mu3 = sum((y-mean(y))^3)/($nobs)
scalar mu4 = sum((y-mean(y))^4)/($nobs)
printf "\n mean = %5.4f\n sample variance = %5.4f\n sample\
std deviation = %5.4f\n",y_bar,y_var,y_se
printf "\n mu2 = %5.4f\n mu3 = %5.4f\n mu4 = %5.4f\n",mu2,mu3,mu4
```

Then, to estimate skewness, $S = \tilde{\mu}^3/\tilde{\sigma}^3$, and excess kurtosis, $K = \tilde{\mu}^4/\tilde{\sigma}^4 - 3$:

```
scalar sig_tild = sqrt(mu2)
scalar skew = mu3/sig_tild^3
scalar ex_kurt = mu4/sig_tild^4 -3
printf "\n std dev. of the mean = %5.4f\n skewness = %5.4f\n\
excess kurtosis = %5.4f\n",se_ybar,skew,ex_kurt
```

Note, in **gretl**'s built in `summary` command, the *excess* kurtosis is reported. The normal distribution has a theoretical kurtosis equal to 3 and the excess is measured relative to that. Hence, excess kurtosis $= \tilde{\mu}^4/\tilde{\sigma}^4 - 3$

If hip size in inches is normally distributed, $Y \sim N(\mu, \sigma^2)$. Based on our estimates, $Y \sim N(17.158, 3.265)$. The percentage of customers having hips greater than 18 inches can be estimated.

$$P(Y > 18) = P\left(\frac{Y-\mu}{\sigma} > \frac{18-\mu}{\sigma}\right) \tag{C.1}$$

Replacing μ and σ by their estimates yields

```
scalar zs = (18 - mean(y))/sd(y)
pvalue z zs
```

The last line actually computes the p-value associated with z-score. So, the `pvalue` command requests that a p-value be returned, the second argument (`z`) indicates the distribution to be used (in this case, `z` indicates the normal), and the final argument (`zs`) is the statistic itself, which is computed in the previous line. The result is 0.3207, indicating that about 32% of the population would not fit into a seat that is 18 inches wide.

How large would a seat have to be to be able to fit 95% of the population? Find y^* to satisfy

$$P(Y \leq y^*) = \frac{y^* - \bar{y}}{\hat{\sigma}} \leq \frac{y^* - 17.1582}{1.8070} = 0.95 \tag{C.2}$$

In **gretl** you need to find the value of $Z = (y^* - \bar{y})/\hat{\sigma}$ that satisfies the probability. The `invcdf` function does this. Since Z is standard normal

```
scalar zz = invcdf(n,.95)
scalar ystar = sd(y)*zz+mean(y)
print ystar
```

The seat width is estimated to be 20.13 inches.

C.2 Interval Estimation

Estimating a confidence interval using the hip data is also easy to do in **gretl**. Since the true variance, σ^2, is not known, the t-distribution is used to compute the interval. The interval is

$$\bar{y} \pm t_c \frac{\hat{\sigma}}{\sqrt{N}} \tag{C.3}$$

where t_c is the desired critical value from the student-t distribution. In our case, $N = 50$ and the desired degrees of freedom for the t-distribution is $N - 1 = 49$. The **gretl** command `critical(t,49,.025` can be used to return the 0.025 critical value from the t_{49} distribution shown in Figure C.1

The computation is

```
open "@gretldir\data\poe\hip.gdt"
scalar y_sd = sd(y)
scalar ybar_sd = y_sd/sqrt($nobs)
scalar lb = mean(y) - 2.01*ybar_sd
scalar ub = mean(y) + 2.01*ybar_sd
```

which indicates that the interval [16.64,17.67] works 95% of the time. Note these numbers differ slightly from those in your book because we used 2.01 as our critical value. Hill et al. carry their critical value out to more decimal places and hence the difference. You can use **gretl**'s internal functions to improve accuracy. Replace 2.01 with `critical(t,$nobs-1,0.025)` and see what happens!

```
scalar lb = mean(y) - critical(t,$nobs-1,0.025)*ybar_sd
scalar ub = mean(y) + critical(t,$nobs-1,0.025)*ybar_sd
printf "\nThe 95\% confidence interval is (%5.4f, %6.4f)\n",lb,ub
```

C.3 Hypothesis Tests

Hypothesis tests are based on the same principles and use the same information that is used in the computation of confidence intervals. The first test is on the null hypothesis that hip size does not exceed 16.5 inches against the alternative that it does. Formally, $H_0 : \mu = 16.5$ against the alternative $H_a : \mu > 16.5$. The test statistic is computed based on the sample average, $\bar{Y}$ and is

$$t = \frac{\bar{Y} - 16.5}{\hat{\sigma}/\sqrt{N}} \sim t_{N-1} \tag{C.4}$$

if the null hypothesis is true. Choosing the significance level, $\alpha = .05$, the right-hand side critical value for the t_{49} is 1.677. The average hip size is 17.1582 with standard deviation 1.807 so the test statistic is

$$t = \frac{17.1582 - 16.5}{1.807/\sqrt{50}} = 2.576 \tag{C.5}$$

The **gretl** code to produce this is:

```
open "@gretldir\data\poe\hip.gdt"
scalar df = $nobs-1
scalar y_bar = mean(y)
scalar y_sd = sd(y)
scalar ybar_sd = y_sd/sqrt($nobs)
scalar tstat = (y_bar-16.5)/(ybar_sd)
scalar c = critical(t,df,0.025)
pvalue t df tstat
```

The `scalar c = critical(t,49,0.025)` statement can be used to get the $\alpha = 0.025$ critical value for the t distribution with 49 degrees of freedom. The next line, `pvalue t 49 tstat`, returns the p-value from the t distribution with 49 degrees of freedom for the computed statistic, `tstat`.

The two-tailed test is of the hypothesis, $H_0 : \mu = 17$ against the alternative, $H_a : \mu \neq 17$.

$$t = \frac{\bar{Y} - 17}{\hat{\sigma}/\sqrt{N}} \sim t_{N-1} \tag{C.6}$$

if the null hypothesis is true. Choosing the significance level, $\alpha = .05$, the two sided critical value is ± 2.01. Hence, you will reject the null hypothesis if $t < -2.01$ or if $t > 2.01$. The statistic is computed

$$t = \frac{17.1582 - 17}{1.807/\sqrt{50}} = .6191 \tag{C.7}$$

and you cannot reject the null hypothesis. The **gretl** code is:

```
scalar tstat = (y_bar-17)/(ybar_sd)
scalar c = critical(t,df,0.025)
pvalue t df tstat
```

C.4 Testing for Normality

Your book discusses the Jarque-Bera test for normality which is computed using the skewness and kurtosis of the least squares residuals. To compute the Jarque-Bera statistic, you'll first need to obtain the summary statistics from your data series.

From **gretl** script

```
open "@gretldir\data\poe\hip.gdt"
summary
```

You could also use the point and click method to get the summary statistics. This is accomplished from the output window of your regression. Simply highlight the hip series and then choose `Data>Summary statistics>selected variables` from the pull-down menu. This yields the results in Table C.1.

One thing to note, **gretl** reports excess kurtosis rather than kurtosis. The excess kurtosis is measured relative to that of the normal distribution which has kurtosis of three. Hence, your computation is

$$JB = \frac{N}{6}\left(\text{Skewness}^2 + \frac{(\text{Excess Kurtosis})^2}{4}\right) \tag{C.8}$$

Which is

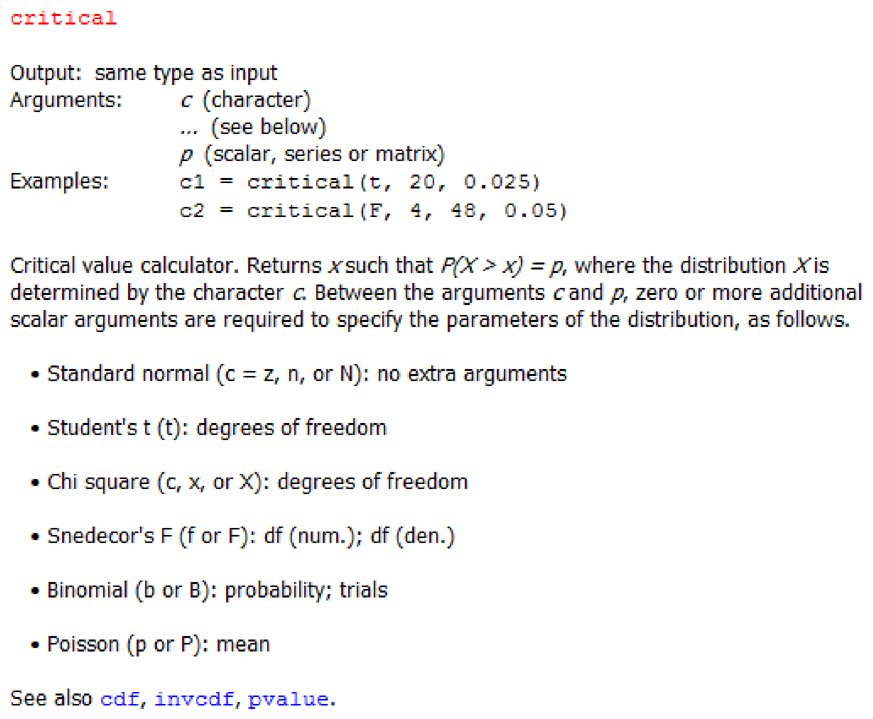
$$JB = \frac{50}{6}\left(-0.0138^2 + \frac{-0.66847^2}{4}\right) = .9325 \tag{C.9}$$

Using the results in section C.1 for the computation of skewness and kurtosis, the **gretl** code is:

```
scalar sig_tild = sqrt(sum((y-mean(y))^2)/($nobs))
scalar mu3 = sum((y-mean(y))^3)/($nobs)
scalar mu4 = sum((y-mean(y))^4)/($nobs)
scalar skew = mu3/sig_tild^3
scalar kurt = mu4/sig_tild^4
scalar JB = ($nobs/6)*(skew^2+(kurt-3)^2/4)
pvalue X 2 JB
```

C.5 Script

```
set echo off
open "@gretldir\data\poe\hip.gdt"
summary
scalar y_bar = mean(y)
scalar y_var = sum((y-y_bar)^2)/($nobs-1)
scalar y_se = sqrt(y_var)
scalar se_ybar = sqrt(y_var/$nobs)

scalar mu2 = sum((y-y_bar)^2)/($nobs)
scalar mu3 = sum((y-mean(y))^3)/($nobs)
scalar mu4 = sum((y-mean(y))^4)/($nobs)
printf "\n mean = %5.4f\n sample variance = %5.4f\n sample\
std deviation = %5.4f\n",y_bar,y_var,y_se
printf "\n mu2 = %5.4f\n mu3 = %5.4f\n mu4 = %5.4f\n",mu2,mu3,mu4

scalar sig_tild = sqrt(mu2)
scalar skew = mu3/sig_tild^3
scalar ex_kurt = mu4/sig_tild^4 -3
printf "\n std dev. of the mean = %5.4f\n skewness = %5.4f\n\
excess kurtosis = %5.4f\n",se_ybar,skew,ex_kurt

# Using the estimates
scalar zs = (18 - mean(y))/sd(y)
pvalue z zs
scalar zz = invcdf(n,.95)
scalar ystar = sd(y)*zz+mean(y)
print ystar

# Confidence interval
open "@gretldir\data\poe\hip.gdt"
scalar y_sd = sd(y)
scalar ybar_sd = y_sd/sqrt($nobs)
scalar lb = mean(y) - 2.01*ybar_sd
scalar ub = mean(y) + 2.01*ybar_sd
scalar lb = mean(y) - critical(t,$nobs-1,0.025)*ybar_sd
scalar ub = mean(y) + critical(t,$nobs-1,0.025)*ybar_sd
printf "\nThe 95\% confidence interval is (%5.4f, %6.4f)\n",lb,ub

# t-test
open "@gretldir\data\poe\hip.gdt"
scalar df = $nobs-1
scalar y_bar = mean(y)
scalar y_sd = sd(y)
scalar ybar_sd = y_sd/sqrt($nobs)
scalar tstat = (y_bar-16.5)/(ybar_sd)
scalar c = critical(t,df,0.025)
pvalue t df tstat
scalar tstat = (y_bar-17)/(ybar_sd)
scalar c = critical(t,df,0.025)
pvalue t df tstat

# Jarque-Bera
scalar sig_tild = sqrt(sum((y-mean(y))^2)/($nobs))
scalar mu3 = sum((y-mean(y))^3)/($nobs)
scalar mu4 = sum((y-mean(y))^4)/($nobs)
scalar skew = mu3/sig_tild^3
scalar kurt = mu4/sig_tild^4
scalar JB = ($nobs/6)*(skew^2+(kurt-3)^2/4)
pvalue X 2 JB
```



critical

Output: same type as input

Arguments: *c* (character)
... (see below)
p (scalar, series or matrix)

Examples:
```
c1 = critical(t, 20, 0.025)
c2 = critical(F, 4, 48, 0.05)
```

Critical value calculator. Returns *x* such that *P(X > x) = p*, where the distribution *X* is determined by the character *c*. Between the arguments *c* and *p*, zero or more additional scalar arguments are required to specify the parameters of the distribution, as follows.

- Standard normal (c = z, n, or N): no extra arguments
- Student's t (t): degrees of freedom
- Chi square (c, x, or X): degrees of freedom
- Snedecor's F (f or F): df (num.); df (den.)
- Binomial (b or B): probability; trials
- Poisson (p or P): mean

See also cdf, invcdf, pvalue.

Figure C.1: Obtaining critical values from the t distribution using **hansl**.

Appendix D

Using **R** with **gretl**

Another feature of **gretl** that makes it extremely powerful is its ability to work with another free program called **R**. **R** is actually a programming language for which many statistical procedures have been written. Although **gretl** is powerful, there are still many things that it won't do, at least without some additional programming. The ability to export **gretl** data into **R** makes it possible to do some sophisticated analysis with relative ease.

Quoting from the **R** web site

> **R** is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.
>
> The design of **R** has been heavily influenced by two existing languages: Becker, Chambers & Wilks' **S** and Sussman's **Scheme**. Whereas the resulting language is very similar in appearance to **S**, the underlying implementation and semantics are derived from **Scheme**.
>
> The core of **R** is an interpreted computer language which allows branching and looping as well as modular programming using functions. Most of the user-visible functions in **R** are written in **R**. It is possible for the user to interface to procedures written in the C, C++, or FORTRAN languages for efficiency. The **R** distribution contains functionality for a large number of statistical procedures. Among these are: linear and generalized linear models, nonlinear regression models, time series analysis, classical parametric and nonparametric tests, clustering and smoothing. There is also a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations. Additional modules (add-on packages) are available for a variety of specific purposes (see **R** Add-On Packages).
>
> **R** was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. In addition, a large

group of individuals has contributed to **R** by sending code and bug reports.

Since mid-1997 there has been a core group (the **R** Core Team) who can modify the **R** source code archive. The group currently consists of Doug Bates, John Chambers, Peter Dalgaard, Seth Falcon, Robert Gentleman, Kurt Hornik, Stefano Iacus, Ross Ihaka, Friedrich Leisch, Uwe Ligges, Thomas Lumley, Martin Maechler, Duncan Murdoch, Paul Murrell, Martyn Plummer, Brian Ripley, Deepayan Sarkar, Duncan Temple Lang, Luke Tierney, and Simon Urbanek.

R has a home page at http://www.R-project.org/. It is free software distributed under a GNU-style copyleft, and an official part of the GNU project (GNU S).

R can be downloaded from http://www.r-project.org/, which is referred to as CRAN or the comprehensive **R** archive network. To install **R**, you'll need to download it and follow the instructions given at the CRAN web site. Also, there is an appendix in the **gretl** manual about using **R** that you may find useful. The remainder of this brief appendix assumes that you have **R** installed and linked to **gretl** through the programs tab in the `File>Preferences>General` pull down menu. Make sure that the 'Command to launch GNR R' box points to the `RGui.exe` file associated with your installation of **R**.

D.1 Ways to Use R in gretl

The standard method of working with **R** is by writing scripts, or by typing commands at the **R** prompt, much in the same way as one would write **gretl** scripts or work with the **gretl** console. This section is a gentle introduction to using R in general with a few tips on using it with **gretl**. As you will see, there are several ways in which to use **R** in **gretl**.

D.1.1 Using the `foreign` command

In section 10.3.4 a `foreign` statement was used to actually execute **R** routines from within **gretl** and to pass results to **gretl** for further processing. A foreign block has the basic structure:

Basic foreign block for R
```
foreign language=R --send-data --quiet
    [ R code to create a matrix called 'Rmatrix' ]
    gretl.export(Rmatrix)
end foreign

matrix m = mread("@dotdir/Rmatrix.mat")
```

The `foreign` command uses the `language=R` to open **R** and to ready it for further computing outside of **gretl**. The `--send-data` option sends the current **gretl** data set to **R**. The `--quiet` option prevents the output from **R** from being echoed in the **gretl** output. The block is closed and **R** exited with the `end foreign` command. What appears in between are statements coded in **R**. The last statement, `gretl.export(Rmatrix)`, is used to export a matrix computation that I have called '`Rmatrix`' to **gretl**. **R** attaches a `.mat` suffix to `Rmatrix` automatically. The matrix is written to the **gretl** working directory on your harddrive. To read the matrix and ready it for further processing, use the `mread` command (`matrix read`). The `mread("@dotdir/Rmatrix.mat")` tells **gretl** to look in the working directory (`@dotdir`)for `Rmatrix.mat`.

This achieves the same effect as submitting the enclosed **R** commands via the GUI in the noninteractive mode (see section 30.3 of the Gretl Users Guide). In other words, it allows you to use **R** commands from within **gretl** . Of course, you have to have installed **R** separately, but this greatly expands what can be done using **gretl**.

D.1.2 Opening an R session

To illustrate, open the *cola.gdt* data in **gretl**.

```
open "C:\Program Files (x86)\gretl\data\poe\cola.gdt"
```

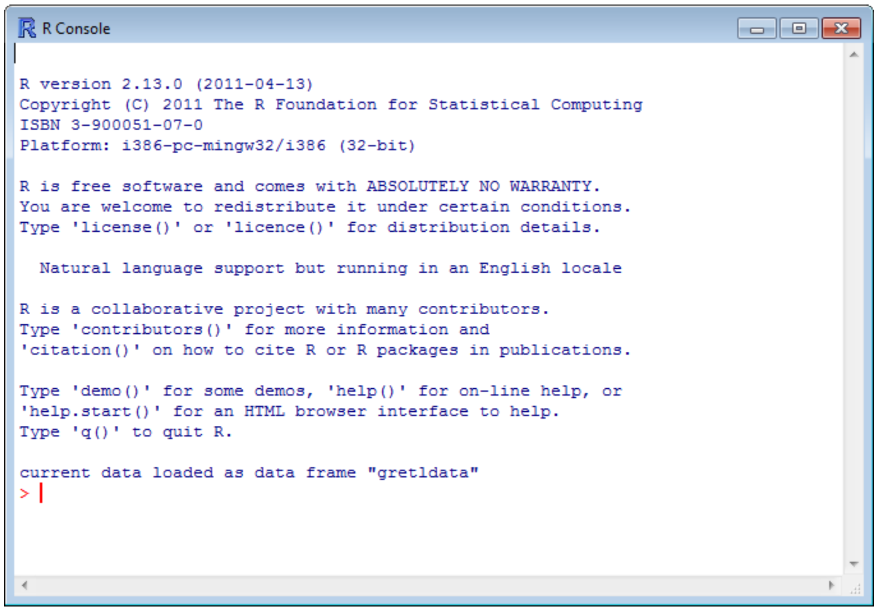
Now, select `Tools>start GNU R` from the pull-down menu. The current **gretl** data set, in this case *cola.gdt*, will be transported into **R**'s required format. You'll see the **R** console which is shown in Figure D.1. The message in **R** tells you that the data are loaded into an **R data frame** called `gretldata`. You can now use **R** with the data loaded from **gretl**. **Gretl**'s data import features are very good and it makes an excellent front-end for getting data into **R**.

D.1.3 R Script from gretl

[1]Opening an **R** window and keying in commands is a convenient method when the job is small. In some cases, however, it would be preferable to have **R** execute a script prepared in advance. One way to do this is via the `source()` command in **R**. Alternatively, **gretl** offers the facility to edit an **R** script and run it, having the current dataset pre-loaded automatically. This feature can be accessed via the File, Script Files menu entry. By selecting User file, one can load a pre-existing **R** script; if you want to create a new script instead, select the New script, **R** script menu entry. Figure D.2

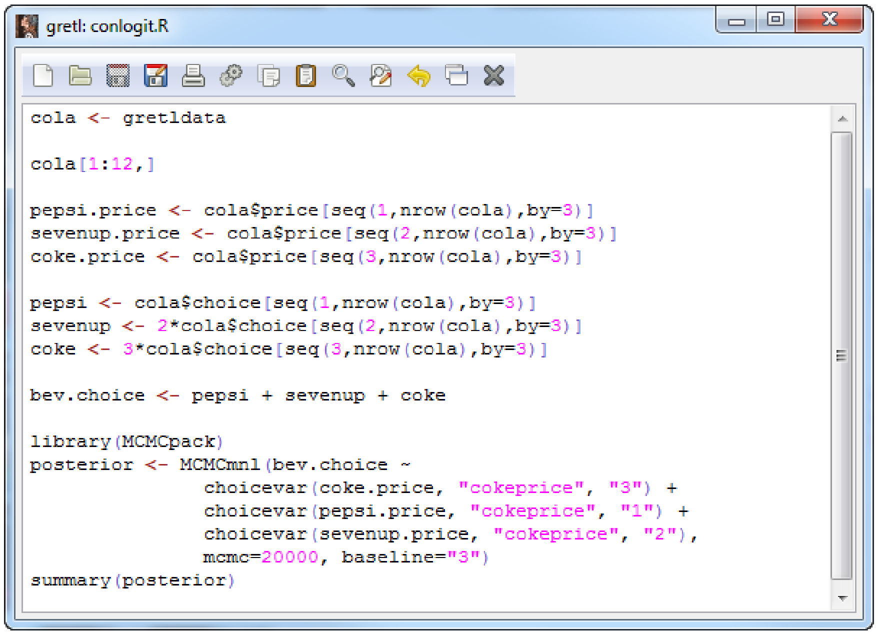
In either case, you are presented with a window very similar to the editor window used for ordinary **gretl** scripts, as in Figure D.2.

[1]This is taken almost directly from the **gretl** Users Guide, chapter 30



```
R version 2.13.0 (2011-04-13)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

current data loaded as data frame "gretldata"
> 
```

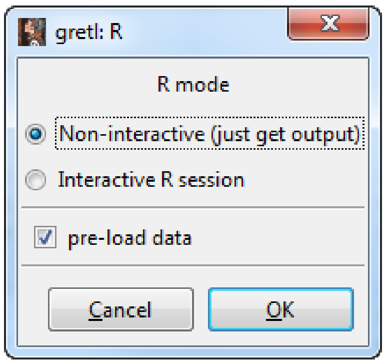
Figure D.1: The **R** console when called from **gretl**. Choose `Tools>Start GNU R` from the main **gretl** window.



```
cola <- gretldata

cola[1:12,]

pepsi.price <- cola$price[seq(1,nrow(cola),by=3)]
sevenup.price <- cola$price[seq(2,nrow(cola),by=3)]
coke.price <- cola$price[seq(3,nrow(cola),by=3)]

pepsi <- cola$choice[seq(1,nrow(cola),by=3)]
sevenup <- 2*cola$choice[seq(2,nrow(cola),by=3)]
coke <- 3*cola$choice[seq(3,nrow(cola),by=3)]

bev.choice <- pepsi + sevenup + coke

library(MCMCpack)
posterior <- MCMCmnl(bev.choice ~
               choicevar(coke.price, "cokeprice", "3") +
               choicevar(pepsi.price, "cokeprice", "1") +
               choicevar(sevenup.price, "cokeprice", "2"),
               mcmc=20000, baseline="3")
summary(posterior)
```

Figure D.2: Using **R** from the **R** script editor in **gretl**.

There are two main differences. First, you get syntax highlighting for **R**s syntax instead of **gretl**'s. Second, clicking on the **Execute** button (the gears icon), launches an instance of **R** in which your commands are executed. Before **R** is actually run, you are asked if you want to run **R** interactively or not in this dialog box:



Non-interactive (just get output) [✓]
Interactive R session
pre-load data ✓

An interactive run opens an **R** instance similar to the one seen in the previous section: your data will be pre-loaded (if the pre-load data box is checked) and your commands will be executed. Once this is done, you will find yourself in **R** and at the **R** prompt. From here you can enter more **R** commands.

A non-interactive run, on the other hand, will execute your script, collect the output from **R** and present it to you in an output window; **R** will be run in the background. This was the approach taken in the canonical correlation analysis from chapter 10, since we did not have further use for **R** and the results were being passed back to **gretl**.

D.2 A few basic commands and conventions

The first thing I usually do is to change the name to something less generic, e.g., cola, using

```
> cola <-gretldata
```

You can also load the current **gretl** data into **R** manually as shown below. To load the data in properly, you have to locate the `Rdata.tmp` file that **gretl** creates when you launch **R** from the GUI. Mine was cleverly hidden in `C:/Users/Lee/AppData/Roaming/gretl/Rdata.tmp`. Once found, use the read.table command in **R** as shown. The system you are using (Windows in my case) dictate whether the slashes are forward or backward. Also, I read the data in as `cola` rather than the generic `gretldata` to make things easier later. **R**.

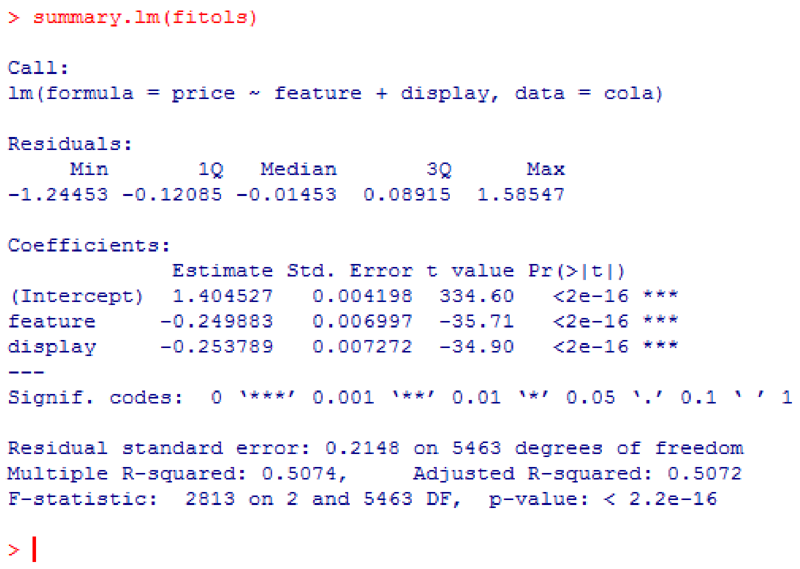
```
> cola <- read.table("C:/Users/Lee/AppData/Roaming/gretl/Rdata.tmp",
+                    header = TRUE )
```

The addition of `Header = TRUE` to the code that **gretl** writes for you ensures that the variable names, which are included on the first row of the `Rdata.tmp`, get read into **R** properly. Then, to run the regression in **R**.

R code to estimate a linear model and print results

```
fitols <- lm(price~feature+display,data=cola)
summary(fitols)
anova(fitols)
```

The `fitols <- lm(price feature+display,data=cola)` command estimates a linear regression model with price as the dependent variable. The results are stored into memory under the name `fitols`. The variables `feature` and `display` are included as regressors. **R** automatically includes an intercept. To print the results to the screen, you have to use the `summary(fitols)` command. Before going further, let me comment on this terse piece of computer code. First, in **R**



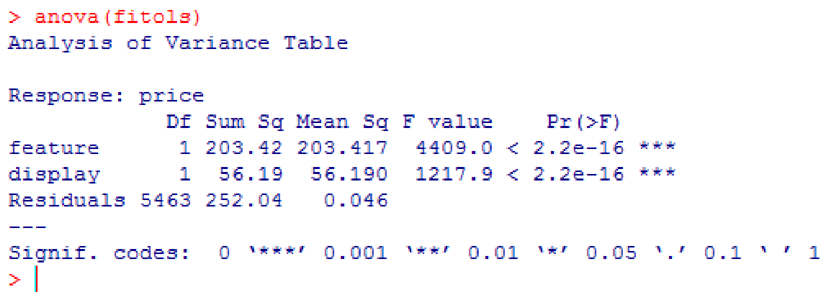
```
> summary.lm(fitols)

Call:
lm(formula = price ~ feature + display, data = cola)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.24453 -0.12085 -0.01453  0.08915  1.58547 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.404527   0.004198  334.60   <2e-16 ***
feature     -0.249883   0.006997  -35.71   <2e-16 ***
display     -0.253789   0.007272  -34.90   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 0.2148 on 5463 degrees of freedom
Multiple R-squared: 0.5074,     Adjusted R-squared: 0.5072 
F-statistic:  2813 on 2 and 5463 DF,  p-value: < 2.2e-16 

> 
```

Figure D.3: The `fitols <- lm(price feature+display,data=cola)` command estimates a linear regression model with price as the dependent variable. The variables `feature` and `display` are included as regressors.

the symbol `<-` is used as the assignment operator[2]; it assigns whatever is on the right hand side (`lm(y∼x,data=gretldata)`) to the name you specify on the left (`fitols`). It can be reversed `->` if you want to call the object to its right what is computed on its left.

[2]You can also use =, but it only assigns in one direction–right is assigned to left.

The `lm` command stands for 'linear model' and in this example it contains two arguments within the parentheses. The first is your simple regression model. The dependent variable is `price` and the independent variables `feature`, `display`, and a constant. The dependent variable and independent variables are separated by the symbol which substitutes in this case for an equals sign. The independent variables are separated by plus signs (+). In a linear model the meaning of this is unambiguous. The other argument points to the data set that contains these two variables. This data set, pulled into **R** from **gretl**, is by default called `gretldata`. We changed the name to `cola` above and that is what we refer to here. There are other options for the `lm` command, and you can consult the substantial pdf manual to learn about them. In any event, you'll notice that when you enter this line and press the return key (which executes this line) **R** responds by issuing a command prompt, and no results! **R** does not bother to print results unless you ask for them. This is handier than you might think, since most programs produce a lot more output than you actually want and must be coerced into printing less. The last line asks **R** to print the ANOVA table to the screen. This gives the result in Figure D.4. It's that simple!



```
> anova(fitols)
Analysis of Variance Table

Response: price
            Df Sum Sq Mean Sq F value    Pr(>F)
feature      1 203.42 203.417  4409.0 < 2.2e-16 ***
display      1  56.19  56.190  1217.9 < 2.2e-16 ***
Residuals 5463 252.04   0.046
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> 
```

Figure D.4: The `anova(olsfit)` command asks **R** to print the anova table for the regression results stored in olsfit.

To do multiple regression in **R**, you can also put each of your independent variables (other than the intercept) into a matrix and use the matrix as the independent variable. A matrix is a rectangular array (which means it contains numbers arranged in rows and columns). You can think of a matrix as the rows and columns of numbers that appear in a spreadsheet program like MS Excel. Each row contains an observation on each of your independent variables; each column contains all of the observations on a particular variable. For instance suppose you have two variables, $x1$ and $x2$, each having 5 observations. These can be combined horizontally into the matrix, X. Computer programmers sometimes refer to this operation as *horizontal concatenation*. Concatenation essentially means that you connect or link objects in a series or chain; to concatenate horizontally means that you are binding one or more columns of numbers together.

The function in **R** that binds columns of numbers together is `cbind`. So, to horizontally concatenate $x1$ and $x2$ use the command

```
X <- cbind(x1,x2)
```

which takes

$$x1 = \begin{pmatrix} 2 \\ 1 \\ 5 \\ 2 \\ 7 \end{pmatrix}, \quad x2 = \begin{pmatrix} 4 \\ 2 \\ 1 \\ 3 \\ 1 \end{pmatrix}, \quad \text{and yields } X = \begin{pmatrix} 2 & 4 \\ 1 & 2 \\ 5 & 1 \\ 2 & 3 \\ 7 & 1 \end{pmatrix}.$$

Then the regression is estimated using

```
fitols <- lm(y~X)
```

There is one more thing to mention about **R** that is very important and this example illustrates it vividly. **R** is case sensitive. That means that two objects x and X can mean two totally different things to **R**. Consequently, you have to be careful when defining and calling objects in **R** to get to distinguish lower from upper case letters.

D.3 Packages

The following is section is taken with very minor changes from Venables et al. (2006).

All **R** functions and datasets are stored in packages. Only when a package is loaded are its contents available. This is done both for efficiency (the full list would take more memory and would take longer to search than a subset), and to aid package developers, who are protected from name clashes with other code. The process of developing packages is described in section Creating **R** packages in *Writing R Extensions*. Here, we will describe them from a users point of view. To see which packages are installed at your site, issue the command `library()` with no arguments. To load a particular package (e.g., the MCMCpack package containing functions for estimating models in Chapter 16

```
> library(MCMCpack)
```

If you are connected to the Internet you can use the `install.packages()` and `update.packages()` functions (both available through the `Packages` menu in the Windows GUI). To see which packages are currently loaded, use

```
> search()
```

to display the search list.

To see a list of all available help topics in an installed package, use

```
> help.start()
```

to start the HTML help system, and then navigate to the package listing in the Reference section.

D.4 Stata Datasets

With **R** you can read in datasets in many different formats. Your textbook includes a dataset written in Stata's format and **R** can both read and write to this format. To read and write Stata's .dta files, you'll have to load the `foreign` package using the library command:

```
1 library(foreign)
2 nels <- read.dta("c:/temp/nels_small.dta")
3 pse <- nels$psechoice
4 attach(nels)
```

Line 2 reads the Stata dataset using the `read.dta` command directly into **R**. It is placed into an object called `nels`. There are two things to note, though. First, the slashes in the filename are backwards from the Windows convention. Second, you need to point to the file in your directory structure and enclose the path/filename in double quotes. **R** looks for the the file where you've directed it and, provided it finds it, reads it into memory. It places the variable names from Stata into the object. Then, to retrieve a variable from the object you create the statement in line 3. Now, you have created a new object called `pse` that contains the variable retrieved from the `nels` object called `psechoice`. This seems awkward at first, but believe it or not, it becomes pretty intuitive after a short time.

The command `attach(nels)` will take each of the columns of `nels` and allow you to refer to it by its variable name. So, instead of referring to `nels$psechoice` you can directly ask for `psechoice` without using the `nels$` prefix. For complex programs, using `attach()` may lead to unexpected results. If in doubt, it is probably a good idea to forgo this option. If you do decide to use it, you can later undo it using `detach(nels)`.

D.5 Final Thoughts

A very brief, but useful document can be found at `http://cran.r-project.org/doc/contrib/Farnsworth-EconometricsInR.pdf` (Farnsworth, 2008). This is a guide written by Grant Farnsworth about using **R** in econometrics. He gives some alternatives to using `MCMCpack` for the models discussed in Chapter 16.

Appendix E

Errata and Updates

Date	Action
2011-07-14	None yet
2011-08-16	Typo on p. 221 (section 9.8.1): `const` missing from regression.
2011-08-17	The **gretl** macro commands `@gretldir`, `@dotdir`, `@workdir` replace references to specific places on harddrive.
2011-08-17	Point out that `launch wgnuplot` is a Windows command.
2011-08-17	The `movavg` function is used to do exponential smoothing
2011-08-17	The multiplier script was modified to use the `elif` construct.
2011-08-17	Script to plot the multipliers from a matrix is added.
2011-08-17	"=" deleted from `ols` command in the script on page 247.
2011-08-17	*gig* was used to estimate TARCH using the GJR option.
2011-08-17	Version 2.1 of scripts posted.
2011-08-18	Error in script on page 248. Removed equal sign in ols command.
2011-08-18	Added the native canonical correlation function, cc, to chapter 10 text and script.
2011-08-18	Version 2.12 of scripts posted.
2011-08-18	Section 10.4 is added to study the performance of TSLS.
2011-08-18	Chapter 16. Variable ordering for conditional logit example was incorrect.

Appendix F

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. VERBATIM COPYING

3. COPYING IN QUANTITY

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if

Bibliography

Adkins, Lee C. (2011*a*), 'Using gretl for monte carlo simulations', *Journal of Applied Econometrics* **26**, n/a. doi: 10.1002/jae.1228.

Adkins, Lee C. (2011*b*), 'Using gretl for monte carlo simulations: A primer', *http://www.learneconometrics.com/pdf/MCgretl/index.htm* .

Anderson, T. W. and H. Rubin (1949), 'Estimation of the parameters of a single equation in a complete system of stochastic equations', *Annals of Mathematical Statistics* **20**, 46–63.

Arellano, M. (2003), *Panel Data Econometrics*, Oxford University Press, Oxford.

Barro, Robert J. and Jong Wha Lee (1996), 'International measures of schooling years and schooling quality', *American Economic Review* **82**(2), 218–223.

Beck, N. and J. N. Katz (1995), 'What to do (and not to do) with time-series cross-section data', *The American Political Science Review* **89**, 634647.

Cottrell, Allin and Riccardo Jack Lucchetti (2011), *Gretl User's Guide*, Department of Economics and Wake Forest University and Dipartimento di Economia Università Politecnica delle Marche, http://ricardo.ecn.wfu.edu/pub//gretl/manual/PDF/gretl-guide.pdf.

Cragg, J.G. and S.G. Donald (1993), 'Testing identifiability and specification in instrumental variables models', *Econometric Theory* **9**(2), 222–240.

Davidson, Russell and James G. MacKinnon (2004), *Econometric Theory and Methods*, Oxford University Press, New York.

Doornik, Jurgen A. and Henrick Hansen (2008), 'An omnibus test for univariate and multivariate normality', *Oxford Bulletin of Economics and Statistics* **70**, 927–939.

Elliott, G. T., J. Rothenberg and J. H. Stock (1996), 'Efficient tests for an autoregressive unit root. econometrica', **64**, 813836.

Epple, D. and Bennett T. McCallum (2006), 'Simultaneous equations econometrics: The missing example', *Economic Inquiry* **44**(2), 374–384.

Farnsworth, Grant V. (2008), Econometrics in **r**. http://cran.r-project.org/doc/contrib/Farnsworth-EconometricsInR.pdf.

Fuller, Wayne (1977), 'Some properties of a modification of the limited information estimator', *Econometrica* **45**, 939–953.

Greene, William H. (2003), *Econometric Analysis*, 5th edn, Prentice Hall, Upper Saddle River, N.J.

Grunfeld, Yehuda (1958), The Determinants of Corporate Investment, PhD thesis, University of Chicago.

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

Heckman, James J. (1979), 'Sample selection bias as a specification error', *Econometrica* **47**(1), 153–161.

Hill, R. Carter, William E. Griffiths and Guay Lim (2011), *Principles of Econometrics*, 4th edn, John Wiley and Sons.

Koop, Gary (2003), *Bayesian Econometrics*, John Wiley & Sons, Hoboken, NJ.

Kwiatkowski, D., P. C. B. Phillips, P. Schmidt and Y. Shin (1992), 'Testing the null of stationarity against the alternative of a unit root: How sure are we that economic time-series have a unit root?', *Journal of Econometrics* **54**, 159–178.

Lancaster, Tony (2004), *An Introduction to Modern Bayesian Econometrics*, Blackwell Publishing, Ltd.

Mixon Jr., J. Wilson and Ryan J. Smith (2006), 'Teaching undergraduate econometrics with gretl', *Journal of Applied Econometrics* **21**, 1103–1107.

Ramanathan, Ramu (2002), *Introductory Econometrics with Applications*, The Harcourt series in economics, 5th edn, Harcourt College Publishers, Fort Worth.

Schwert, G. W. (1989), 'Tests for unit roots: A monte carlo investigation', *Journal of Business and Economic Statistics* **2**, 147159.

Stock, James H. and Mark W. Watson (2006), *Introduction to Econometrics*, second edn, Addison Wesley, Boston, MA.

Stock, James H. and Motohiro Yogo (2005), Testing for weak instruments in linear IV regression, *in* Andrews, Donald W. K. and James H. Stock, eds, 'Identification and Inference for Econometric Models: Essays in Honor of Thomas Rothenberg', Cambridge University Press, pp. 80–108.

Venables, W. N., D. M. Smith and **R** Development Core Team (2006), 'An introduction to **r**'.

Index