# Using GRETL for Principles of Econometrics, 5th edition

Adkins, Lee

Oklahoma State University

12 November 2018

# Using **gretl** for Principles of Econometrics, 5th Edition
Version 1.0[1]

Lee C. Adkins
Professor of Economics
Oklahoma State University

November 12, 2018

[1] Visit http://www.LearnEconometrics.com/gretl.html for the latest version of this book.

# License

# Preface to 4th edition

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 book, 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 **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.

Finally, I want to say that my conversion to **gretl** was not immediate. In fact I still use other software as occasions require, though more infrequently. That said, I have become a true believer in the power of **gretl**. It is now my go to software. I trust it. It is simple to use and to program. In my opinion it combines the best of Gauss and Eviews. It is both a high level programming language and a useful front-end for doing standard econometrics. The ease with which one can move back and forth from both uses makes it truly unique. As a former Gauss user, I find **gretl** up to the tasks that I choose. I heartily recommend that you take some time to work with it and to learn it. You can't help but come to appreciate its power. Its worth is derived from what it does, not its price.

I want to thank the **gretl** team of Allin Cottrell and Riccardo Lucchetti for putting so much effort into **gretl**. I don't know how they find the time to make this such a worthwhile project. It is a terrific tool 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.

# Preface to 5th edition

Principles of Econometrics Hill et al. (2018) in now in its 5th edition and the book has undergone significant updating. Since the purpose of this manual is to show you how to reproduce all of the examples in POE5, a lot has changed here as well. Also, **gretl** itself has evolved in the years since the 2014 edition of this manual appeared.

There are several new commands (e.g., `midasreg` and `kalman`) and some of the options to existing commands have changed. Minor changes to syntax have also been made (for instance, logical equality is not `g==1` rather than `g=1`, `end loop` is not `endloop` and so on. There have been some additions to the available options and accessors. Some of the **gretl** menu tree has been rearranged as well.

In this edition, I have spent more time manipulating **gnuplot** through the new `plot` command. This command gives the user access to some of the important features of **gnuplot** in a relatively straightforward way.

The `printf` commands are used more extensively to produce output. This makes what the routines do more apparent with the passage of time. I've also used the assignment operator to add model results to the session. This is a wonderful facility that makes accumulating results and conducting subsequent tests very easy via the GUI.

I've also chosen to place the accompanying datasets into the working directory. Most operating systems have a "documents" directory where the user places new files. This is where I locate my working directory and it is where I choose to store the datasets (in a subdirectory called data). When working on remote systems, this location is usually available to the user. This is a bit clumsy in that **gretl** permits installation of the datasets into **gretl** itself. Once installed the datasets are available from tabs in the **gretl** data files window. Feel free to install the data elsewhere, but take care that the referenced file locations to the data files used in the supplied scripts will need to be modified.

You'll notice that the manual has grown by 50% since the last edition, despite trying to reduce redundancy by making better use of cross-referencing. A lot of this comes in Chapter 16 where in *POE5* the authors computed marginal effects AND their standard errors. Although this is fairly easy to compute in **gretl**, it requires new functions and some rather messing looking code. In this effort, I also used a very nice function package, *lp-mfx*, written by Allin Cottrell that computes various probabilities for qualitative choice models. Allin was also kind enough to let me use his Hausman-Taylor function in Chapter 10. Other packages from the **gretl** database are used, including *HIP* and *GIG* from Jack Lucchetti and *waldTest* by Oleh Komashko. Also, I want to thank Sven Schreiber for cleaning up and maintaining the growing library of function packages available from the server. Sven culled through every package to ensure that it was complete and in working order. I also want to thank Allin, Jack, and Sven for their support and feedback on this project.

Finally, I must remind users that the purpose of this manual is to supplement the textbook *POE5* Hill et al. (2018); it is not a stand alone work, though it is relatively self-contained. When confusion arises, please consult *POE5*. This has been a fun project and I hope users find it helpful as they explore the possibilities of **gretl**. It is fine software that is suitable for teaching and research. If it had been available when my career started – and of course a computer to run it – I'd have published a hundred papers by now (wishful thinking perhaps). I can confidently say, however, that had **gretl** been available in its current form my grasp of econometric principles and computing would be much stronger, especially earlier in my career. I hope others will find it as inspiring to use as I do.

# Contents

# List of Figures

**Chapter 1**

**Chapter 2**

**Chapter 3**

**Chapter 4**

**Chapter 5**

**Chapter 6**

**Chapter 7**

**Chapter 8**

**Chapter 9**

**Chapter 10**

**Chapter 12**

**Chapter 13**

**Chapter 14**

**Chapter 15**

**Chapter 16**

**Chapter B**

**Chapter C**

**Chapter E**

# Chapter 1

# Introduction

Some of the basic features of **gretl** are introduced in this chapter. 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, **gretl**'s powerful scripting language, **hansl**, will be introduced as well.

## 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 a 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, Stata to name a few) you may 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 access to several very useful databases served by Wake Forest University. From the **gretl** website, 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. (2018).

Gretl offers a full range of least-squares based estimators, either for a single equation and for a system, 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 expect an occasional bug, 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 macOS 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, 5th edition* (*POE5*) by Hill et al. (2018), 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/`.

There you will find compressed zip files that can be downloaded and installed on your computer. For the scripts in this book, I have mine installed in my `documents` folder

```
\Documents\gretl\poe5\data
```

directory of your computer's harddrive.[1] If you have installed **gretl** in any place other than `\Documents\gretl\poe5\data` then unzip the files into the new location. Another likely place on a Windows system is in your user directory (mine is `\leead`, which is in my `Users` directory on the C: drive.):

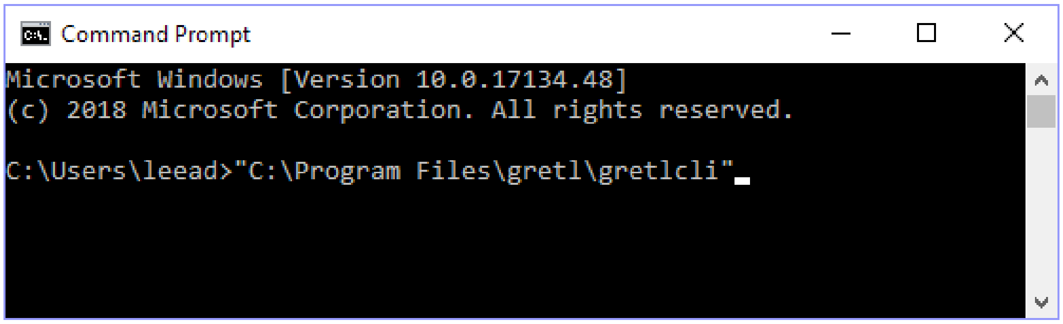
```
C:\Users\leead\AppData\Roaming\gretl\data\poe5
```

If you unzip the data file here, they you will need to change the included script files so that they point to the proper data location. If located here, you can simply issue an *open datasetname.gdt* to open your file.

### 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. The GUI allows 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 `"C:\Program Files\gretl\gretlcli.exe"` (Figure 1.1). Be sure to use the correct path to your **gretl** installation and to enclose everything in quotes if there are spaces in any of the file or directory names.

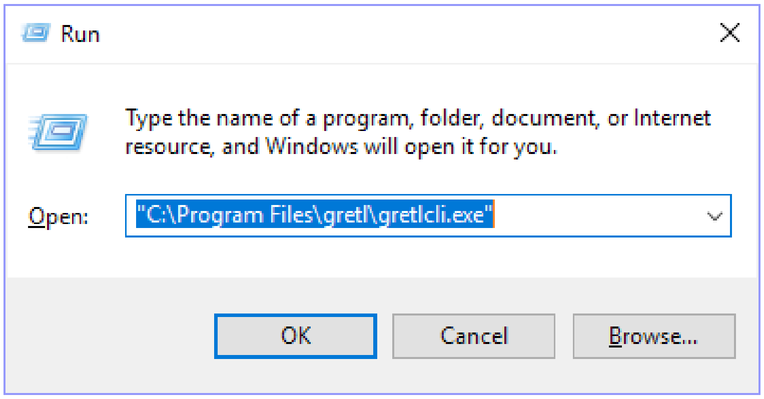


```
Microsoft Windows [Version 10.0.17134.48]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\leead>"C:\Program Files\gretl\gretlcli"
```

Figure 1.1: Opening the command line interface version of **gretl** from a command prompt.

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

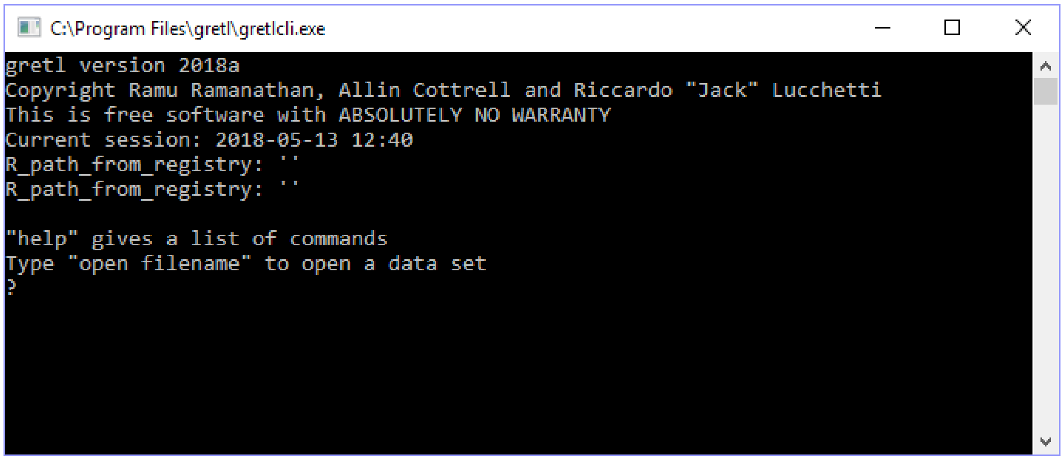
In Windows 10 the **Run** dialog box allows you to browse for the file. Choose **Start**>**Run** to open the dialog shown in Figure 1.2. In the box, use **Browse** button to locate the directory in which **gretl** is installed. Click **OK** and the command line version shown in figure 1.3 opens. There 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 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`.



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: "C:\Program Files\gretl\gretlcli.exe"

Figure 1.2: Opening the command line interface version of **gretl** using **Start**>**Run**



```
gretl version 2018a
Copyright Ramu Ramanathan, Allin Cottrell and Riccardo "Jack" Lucchetti
This is free software with ABSOLUTELY NO WARRANTY
Current session: 2018-05-13 12:40
R_path_from_registry: ''
R_path_from_registry: ''

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

Figure 1.3: The command line version of **gretl**

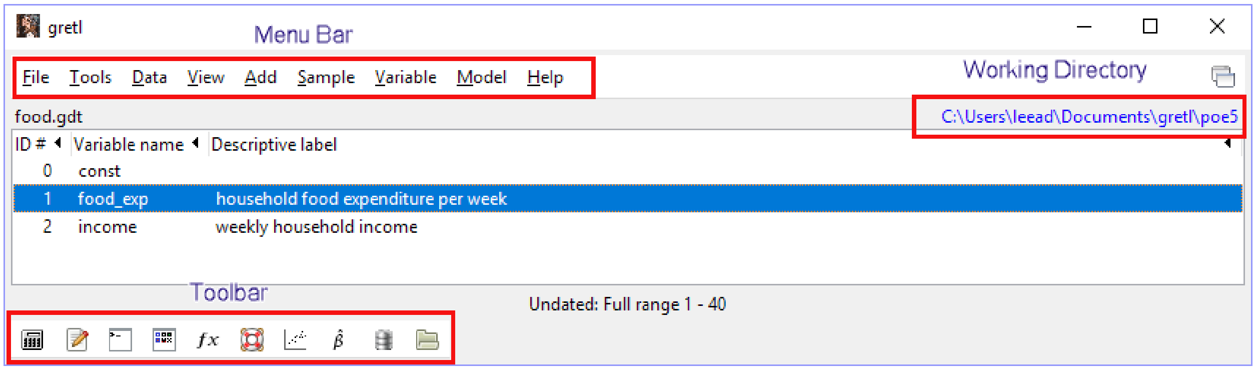
If you are in fact using the Microsoft Windows operating system, then you probably won't be using **gretl** from the command line very often anyway. This version of the program is probably the most useful for Linux users wishing to run **gretl** from a terminal window. If your machine is resource constrained,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

A better way to execute single **gretl** commands is through the **gretl console**. In normal practice, the console is 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.

To execute a series of commands, use **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, save the commands in a script file which can be reopened and run in one step to reproduce the results.

The script environment is often used to conduct Monte Carlo simulations in econometrics. Monte Carlo studies use computer simulation (sometimes referred to as experiments) to study the properties of statistics. This is especially useful when the mathematical properties of your statistic is particularly difficult to derive analytically. In the exercises below, there are rudimentary examples of how these experiments can be constructed and used in econometrics. Also, you can consult a separate paper of mine Adkins (2011*b*) that can be found at `http://www.learneconometrics.com/pdf/MCgretl/index.htm`.

The main window of the graphical user interface, which is opened using *gretl.exe*, is shown below in Figure 1.4.


gretl
Menu Bar
File Tools Data View Add Sample Variable Model Help
Working Directory
C:\Users\leead\Documents\gretl\poe5
Toolbar

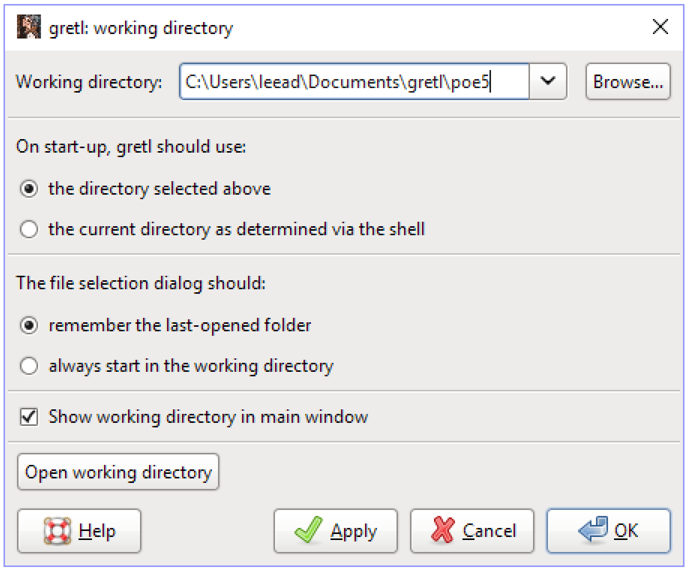
food.gdt

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

Undated: Full range 1 - 40

Figure 1.4: 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. Also, on the right-hand-side you'll see the current working directory. For this manual, I've created a **\gretl\poe5** directory in my documents folder to serve as my working directory. To set your working directory choose **File>Working directory** from the pull-down menu to open the dialog box shown in figure 1.5.



gretl: working directory

Working directory: C:\Users\leead\Documents\gretl\poe5

On start-up, gretl should use:

● the directory selected above

○ the current directory as determined via the shell

The file selection dialog should:

● remember the last-opened folder

○ always start in the working directory

☑ Show working directory in main window

Figure 1.5: Use this dialog to change the working directory. The working directory is where **gretl** reads and writes files.

### 1.1.3 Common Conventions

In the beginning, I will illustrate 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 appearance of these figures will diminish and I will direct you to the proper commands that can be executed from the console or as a script using commands only. More complex series of commands use **gretl** scripts, which as noted above, can be executed in a single batch.

Dialog selection via the GUI will refer to the menu path as **A>B>C** which indicates that you 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. The working directory is where **gretl** reads and writes to. To refer to this location generically, use the `@workdir` macro. The **gretl** installation directory 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 10 system, **gretl** is installed in the `c:\Program Files\gretl` directory. The data sets for *POE5* are in

"@workdir\data\". To refer to this location I can simply use "@workdir\data\".

## 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 many softwares use proprietary data formats that make transferring data between applications difficult. You'll notice that the authors of *POE5* 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 (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. Gretl provides 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, load the food expenditure data used in Chapter 2 of *POE5*. 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. From the main **gretl** window click on **File>Open data>Sample file** as shown in Figure 1.6.

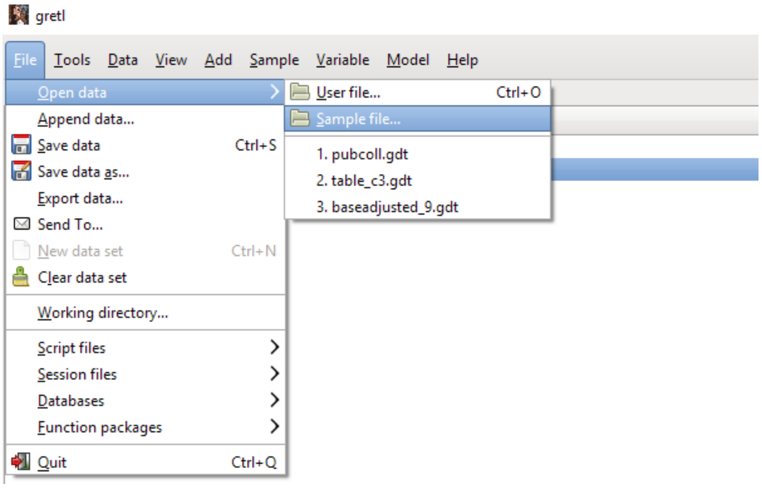
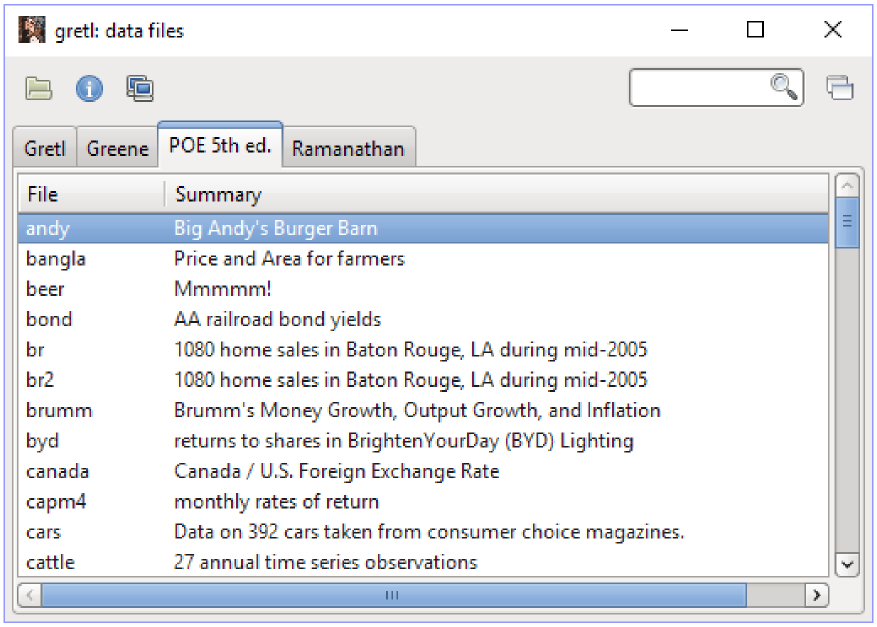


Figure 1.6: Opening sample data files from **gretl**'s main window

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 opens another window (Figure 1.7) that contains tabs for each of the data compilations that are installed in the `gretl/data` directory of your **gretl** program. If you installed the data sets that accompany this book into **gretl**'s installation directors (e.g., `c:\Program Files\gretl`) then a tab will appear like the one shown in Figure 1.7.



| 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. |
| cattle | 27 annual time series observations |

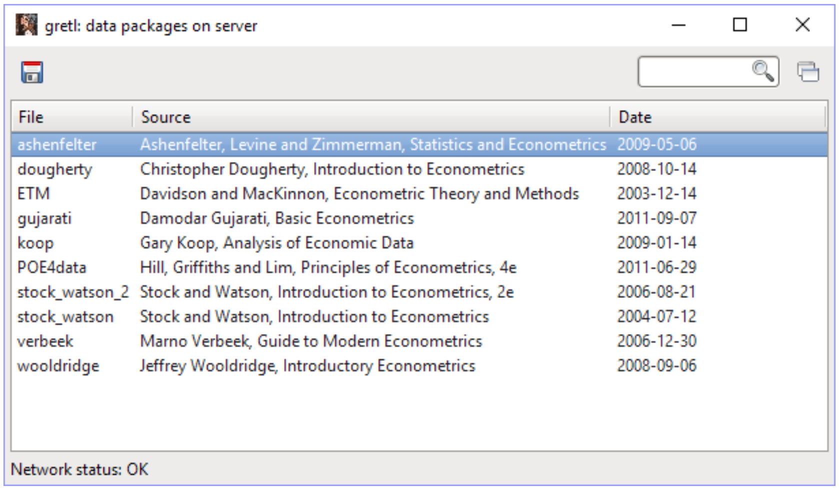
Figure 1.7: This is **gretl**'s data files window. Notice that in addition to *POE5*, data sets from Ramanathan (2002), Greene (2003), are installed on my system.

As of May 2018, there are data sets from several other prominent texts available on the **gretl** website. Click on the look on server icon in the data files dialog (third from the left). This reveals the following list (Figure 1.8) with links to the available downloads.

Click on the **POE 5th 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 1.9.

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 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.

Gretl gives you the opportunity to **import** data. Expanding this (**File**>**Open data**>**User file**) launches an open file dialog box shown in Figure 1.10. Expanding the scroll arrows reveals a list of supported import formats. These include CSV, 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**.



gretl: data packages on server

| File | Source | Date |
|---|---|---|
| ashenfelter | Ashenfelter, Levine and Zimmerman, Statistics and Econometrics | 2009-05-06 |
| dougherty | Christopher Dougherty, Introduction to Econometrics | 2008-10-14 |
| ETM | Davidson and MacKinnon, Econometric Theory and Methods | 2003-12-14 |
| gujarati | Damodar Gujarati, Basic Econometrics | 2011-09-07 |
| koop | Gary Koop, Analysis of Economic Data | 2009-01-14 |
| POE4data | Hill, Griffiths and Lim, Principles of Econometrics, 4e | 2011-06-29 |
| stock_watson_2 | Stock and Watson, Introduction to Econometrics, 2e | 2006-08-21 |
| stock_watson | Stock and Watson, Introduction to Econometrics | 2004-07-12 |
| verbeek | Marno Verbeek, Guide to Modern Econometrics | 2006-12-30 |
| wooldridge | Jeffrey Wooldridge, Introductory Econometrics | 2008-09-06 |

Network status: OK

Figure 1.8: These sets of data from various textbooks are available for installation into **gretl**. Highlight the one you want to install and click on the diskette icon.[3]

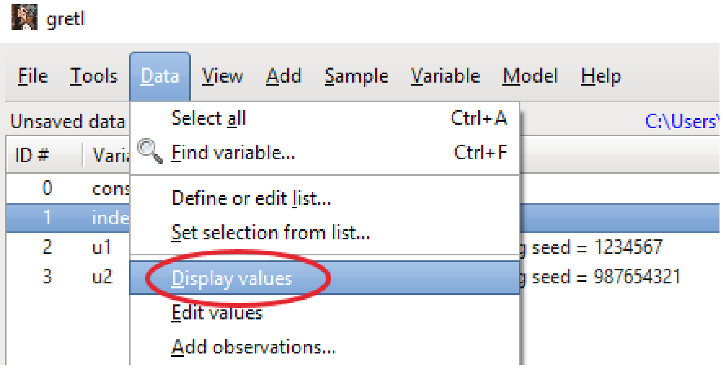
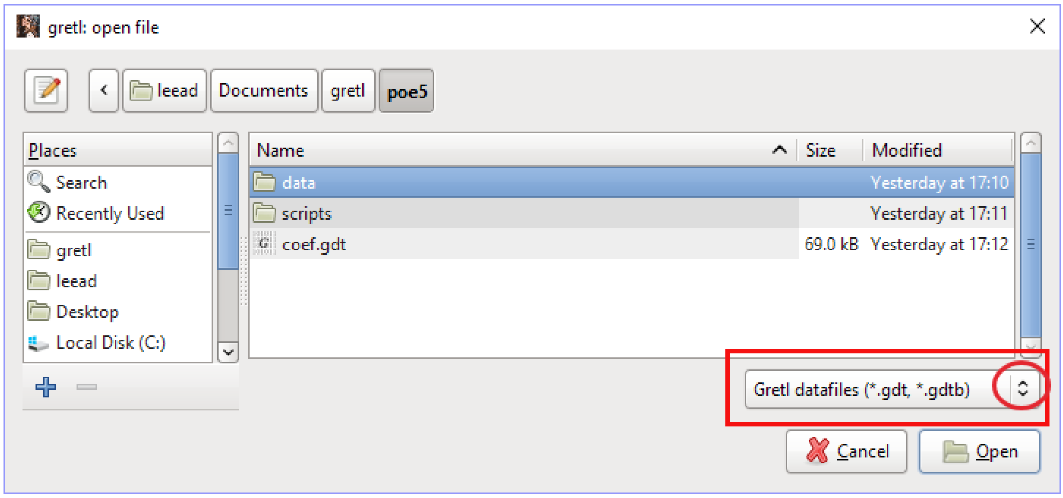


Figure 1.9: Use the cursor to highlight all of the variables. Then click **Data>Display values** to list the data set.

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 1.11) 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 *POE5* 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.



| Name | Size | Modified |
|---|---|---|
| data | | Yesterday at 17:10 |
| scripts | | Yesterday at 17:11 |
| coef.gdt | 69.0 kB | Yesterday at 17:12 |

Gretl datafiles (*.gdt, *.gdtb)

Figure 1.10: The open file dialog allows you to open **gretl** data sets and to import others in various formats.

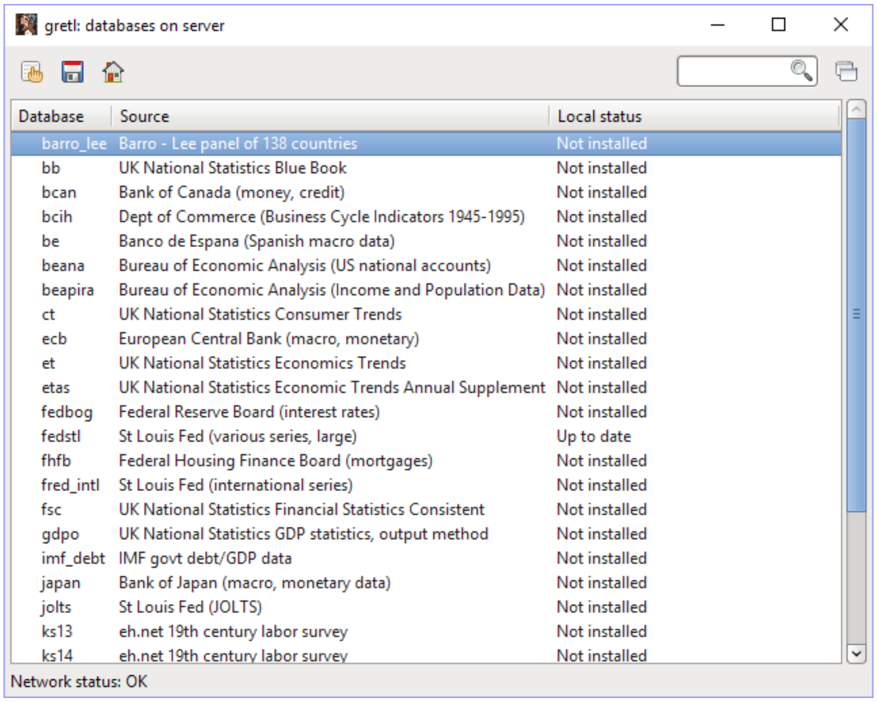
## 1.3 Using the gretl Language

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**. Hansel 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 1.4. From the console you execute commands, one by one by typing **gretl** code after the command



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 |
| ct | UK National Statistics Consumer Trends | 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 |
| fsc | UK National Statistics Financial Statistics Consistent | Not installed |
| gdpo | UK National Statistics GDP statistics, output method | Not installed |
| imf_debt | IMF govt debt/GDP data | 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 |

Network status: OK

Figure 1.11: There are a number of databases that contain useful data for use in your own projects. The left-most icon on the tool bar will list the series in the database. The diskette icon will install the series into your

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 (or down) as needed.

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 the correct language syntax, which can be found in the Gretl **Command Reference**. The command reference can be accessed using **CTRL+H** or from **Help** on the menu bar in the main **gretl** window.

Clicking on anything in blue will take you to the desired information for that command. Ob-

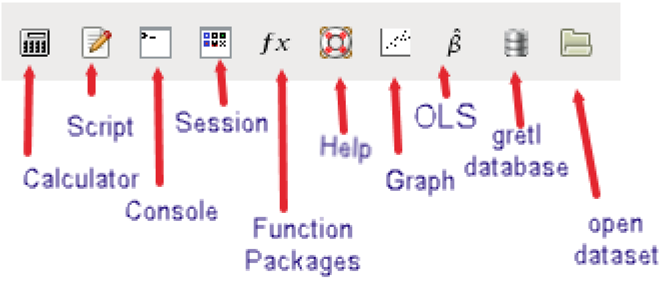
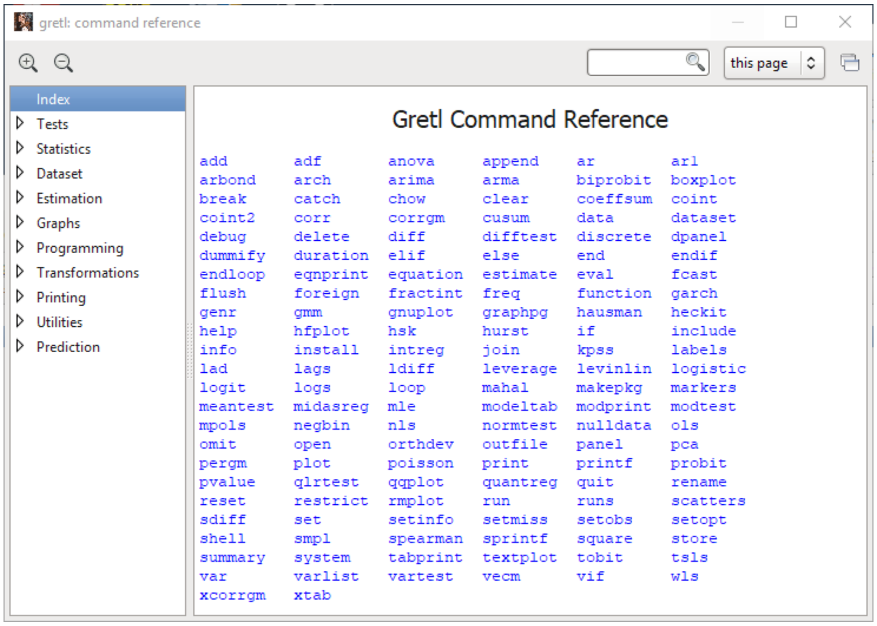


Figure 1.12: The toolbar appears at the bottom of the main menu.

viously, the keyboard shortcut F1 will also bring up the command reference (Figure 1.13). You'll also notice that .pdf versions of the command and function references can also be retrieved from the **Help** drop-down menu.



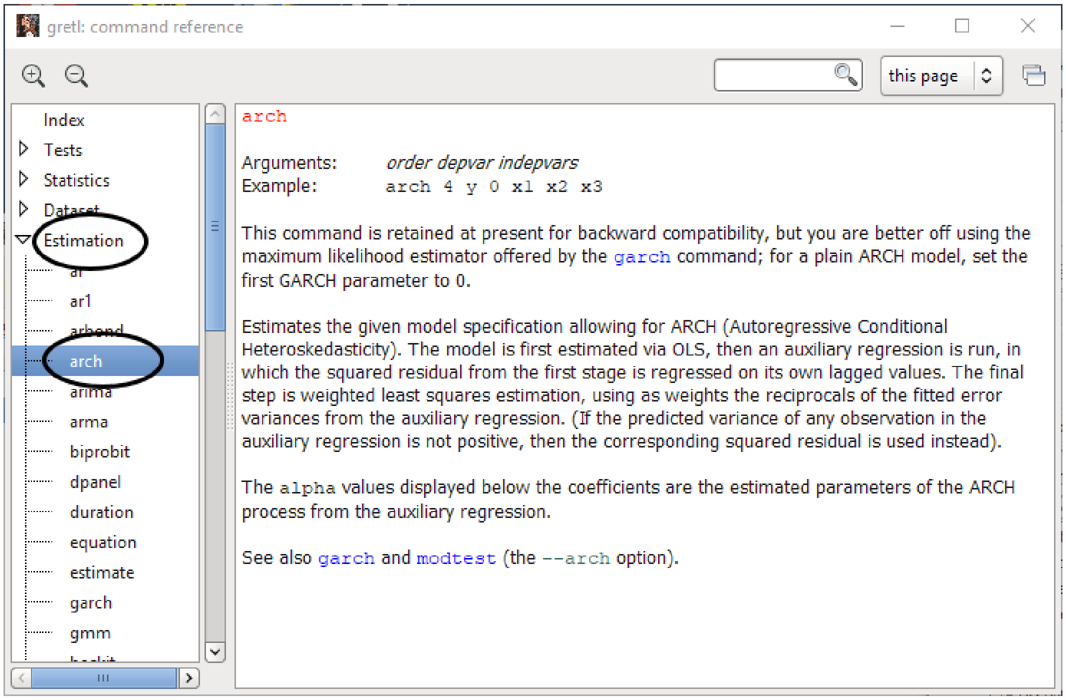
Gretl Command Reference

| | | | | | |
|---|---|---|---|---|---|
| add | adf | anova | append | ar | arl |
| arbond | arch | arima | arma | 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 | eval | fcast |
| flush | foreign | fractint | freq | function | garch |
| genr | gmm | gnuplot | graphpg | hausman | heckit |
| help | hfplot | hsk | hurst | if | include |
| info | install | intreg | join | kpss | labels |
| lad | lags | ldiff | leverage | levinlin | logistic |
| logit | logs | loop | mahal | makepkg | markers |
| meantest | midasreg | mle | modeltab | modprint | modtest |
| mpols | negbin | nls | normtest | nulldata | ols |
| omit | open | orthdev | outfile | panel | pca |
| pergm | plot | poisson | print | printf | probit |
| pvalue | qlrtest | qqplot | quantreg | quit | rename |
| reset | restrict | rmplot | run | runs | scatters |
| sdiff | set | setinfo | setmiss | setobs | setopt |
| shell | smpl | spearman | sprintf | square | store |
| summary | system | tabprint | textplot | tobit | tsls |
| var | varlist | vartest | vecm | vif | wls |
| xcorrgm | xtab | | | | |

Figure 1.13: 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.

Commands can be searched by topic from the command reference window. An index appears in the left side panel (see Figure 1.14). Choose the desired category from the list and select a command (e.g., **Estimation>arch**). The words indicated in blue text are links to related commands. For instance, clicking on arch takes you to the reference entry for ARCH 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



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 1.14: Finding help on the `arch` command using the Command Reference

the pull-down menu as shown in Figure 1.15.

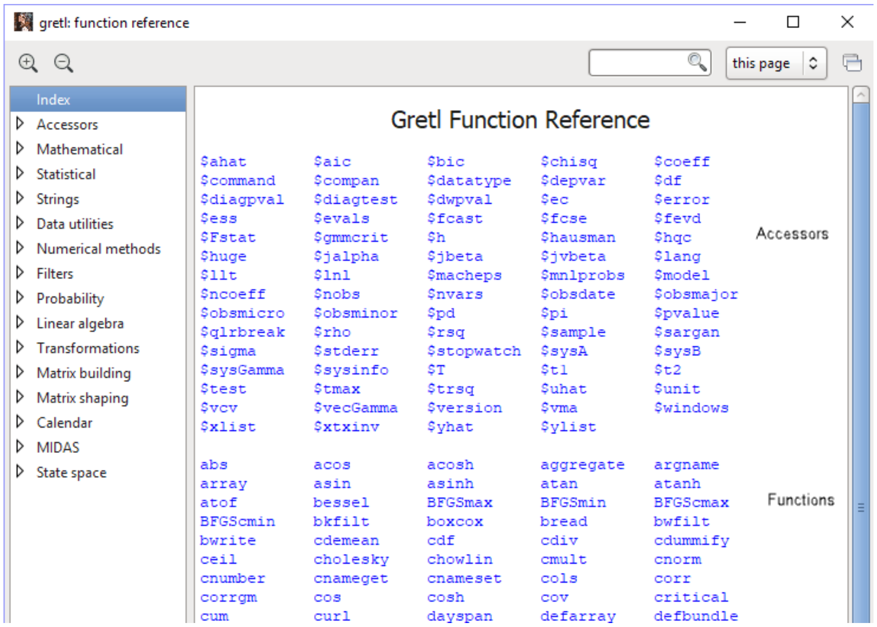
In addition, the current list of available accessors can be summoned using the `varlist` command:

```
varlist --type=accessor
```

By default `varlist` prints a listing of the series in the current dataset (if any); `ls` may be used as an alias. When the `--type` option is given, it should be followed (after an equals sign) by one of the following typenames: `series`, `scalar`, `matrix`, `list`, `string`, `bundle` or `accessor`. The effect is to print the names of all currently defined objects of the named type. The `varlist` command can quickly become your best friend in **gretl**.

### 1.3.2 Scripts

Gretl commands can be collected and saved into a file that can be executed at once and used again. This starts by opening a new **script** from the file menu. The command **File>Script files>New script>gretl script** from the pull-down menu opens the script editor shown in Figure 1.16. Type the commands you want to execute in the box using one line for each command.



Gretl Function Reference

Accessors

| | | | | |
|---|---|---|---|---|
| $ahat | $aic | $bic | $chisq | $coeff |
| $command | $compan | $datatype | $depvar | $df |
| $diagpval | $diagtest | $dwpval | $ec | $error |
| $ess | $evals | $fcast | $fcse | $fevd |
| $Fstat | $gmmcrit | $h | $hausman | $hqc |
| $huge | $jalpha | $jbeta | $jvbeta | $lang |
| $llt | $lnl | $macheps | $mnlprobs | $model |
| $ncoeff | $nobs | $nvars | $obsdate | $obsmajor |
| $obsmicro | $obsminor | $pd | $pi | $pvalue |
| $qlrbreak | $rho | $rsq | $sample | $sargan |
| $sigma | $stderr | $stopwatch | $sysA | $sysB |
| $sysGamma | $sysinfo | $T | $t1 | $t2 |
| $test | $tmax | $trsq | $uhat | $unit |
| $vcv | $vecGamma | $version | $vma | $windows |
| $xlist | $xtxinv | $yhat | $ylist | |

Functions

| | | | | |
|---|---|---|---|---|
| abs | acos | acosh | aggregate | argname |
| array | asin | asinh | atan | atanh |
| atof | bessel | BFGSmax | BFGSmin | BFGScmax |
| BFGScmin | bkfilt | boxcox | bread | bwfilt |
| bwrite | cdemean | cdf | cdiv | cdummify |
| ceil | cholesky | chowlin | cmult | cnorm |
| cnumber | cnameget | cnameset | cols | corr |
| corrgm | cos | cosh | cov | critical |
| cum | curl | dayspan | defarray | defbundle |

Figure 1.15: The function reference can be accessed by **Help>Function reference** from the pull-down menu.

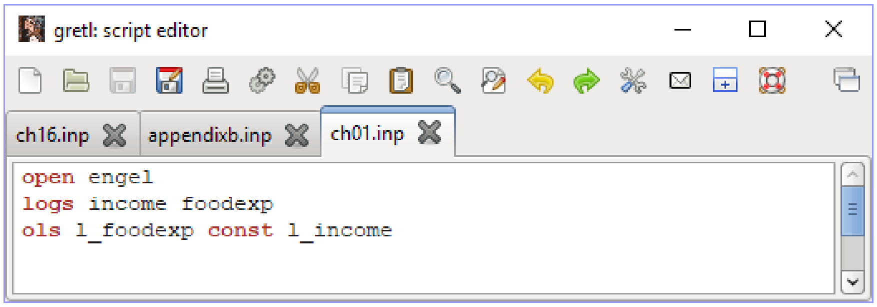
The **continuation command**, the backslash (\), is used when there is a very long command that exceeds one line. To save the file, use the "save" button at the top of the box (first one from the left). If this is a new file, you'll be prompted to provide a name for it; this one I called *engel_ch1*, which shows up at the top of the editor window.

To run the program, click the mouse on the "gear" button. In the figure shown, the *engel.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 $\ln(\mathit{foodexp})$ as its dependent variable and $\ln(\mathit{income})$ 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.[4]

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 lifesaver 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 (Sorry, help not found) or you are taken to the topic from the command reference. Generally, this works successfully on commands that are highlighted in color in the script editor.

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



```
open engel
logs income foodexp
ols l_foodexp const l_income
```

Figure 1.16: The 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.

### 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 1.17. 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 have made.

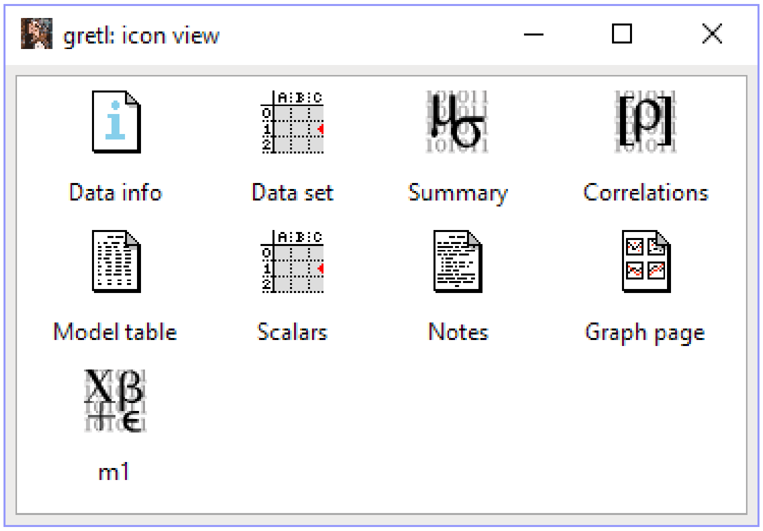


Figure 1.17: The session window

Objects are represented as icons and these objects can be saved with the session for later use. When you re-open a saved session, these objects are available again. To add a model to your session, use the **File>Save to session as icon** option from the model's pull-down menu. Or, most **gretl** estimation and graph commands can be assigned to an object using the assignment operator <-. For instance, to assign a least squares estimated model to a session icon called m1 in a script, use:

```
m1 <- ols l_foodexp const l_income
```

To add a graph, right click on the graph and choose the option **save to session as icon**. Most graphs can also be assigned to an icon from a script as well. Don't forget to save the session before exiting **gretl**if future access to these is desired; 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 1.18.

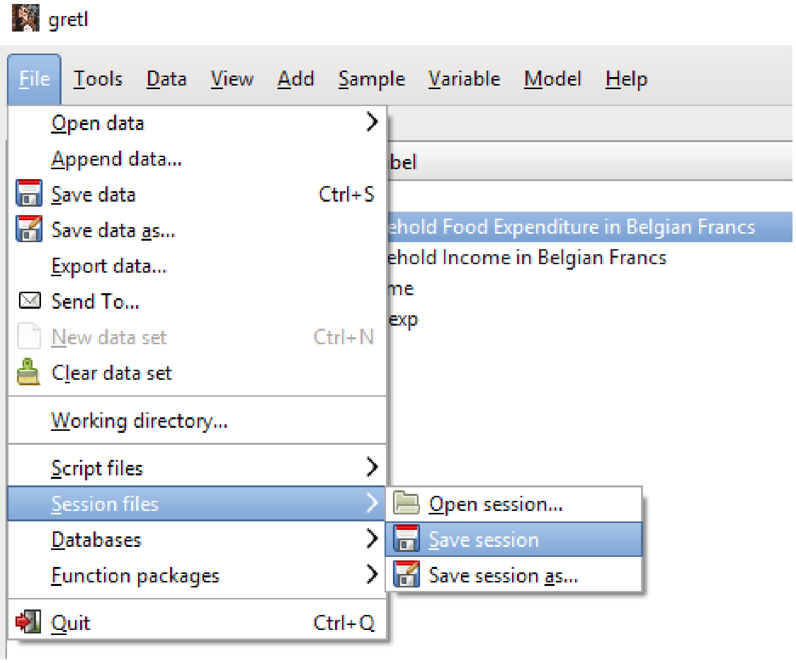


Figure 1.18: Saving a session

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, 2018, pp. 16):

> 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 6.3 you'll find an example that uses the model table and an example on page (192).

### 1.3.4 Generating New Variables

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

Gretl is 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.

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, one can use the `series` command or `genr`. Suppose there is a variable in the 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.

So, the 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.

You may want a scalar result to be expanded into a series or vector. This is done 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 the keyword indicates that the result should be a scalar: if it is not, an error occurs.

In many cases, genr, series, scalar, or matrix statements can be omitted 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.

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. I think there are good reasons to get started on the right foot by adopting good programming practices.[5] 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 writing and debugging functions. Gretl functions are a powerful way to extend **gretl**'s capabilities. They can be 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. Note, there are many user written functions in this manual, so be prepared.

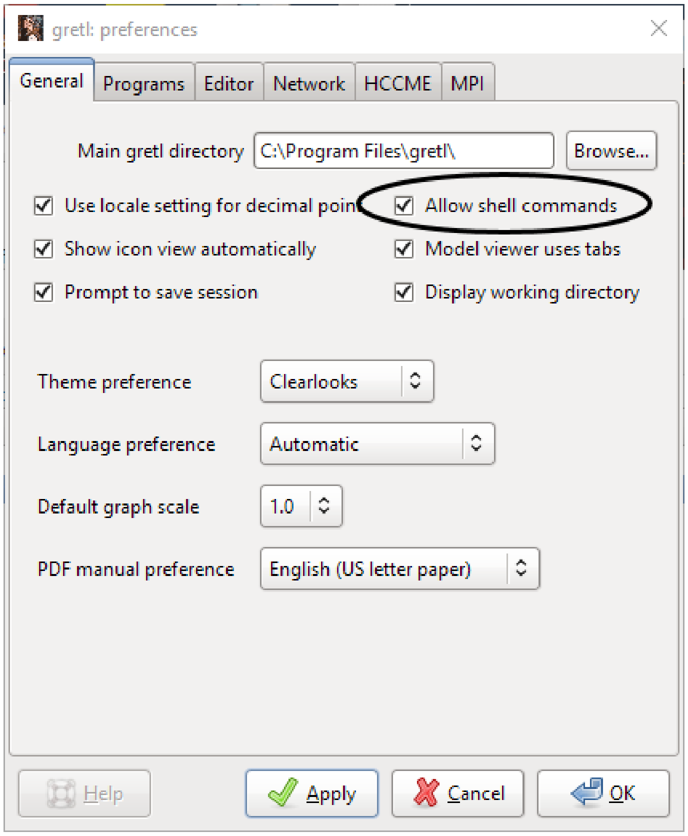
## 1.4 GNUPLOT

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** (which is actually pronounced "new plot") in a script or from the console, the output may be written to a file that is placed in the working directory of **gretl**. If you are not sure where that is, click

[5] 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.

**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 it should be fairly easy to locate.

To view the graph and to edit it requires opening the **gnuplot** program. Before launching **gnuplot** for the first time, open **gretl**'s preference and enable **Allow shell commands** in the **General** preferences tab (see Figure 1.19).



Main gretl directory C:\Program Files\gretl\

☑ Use locale setting for decimal poin
☑ Allow shell commands
☑ Show icon view automatically
☑ Model viewer uses tabs
☑ Prompt to save session
☑ Display working directory

Theme preference Clearlooks

Language preference Automatic

Default graph scale 1.0

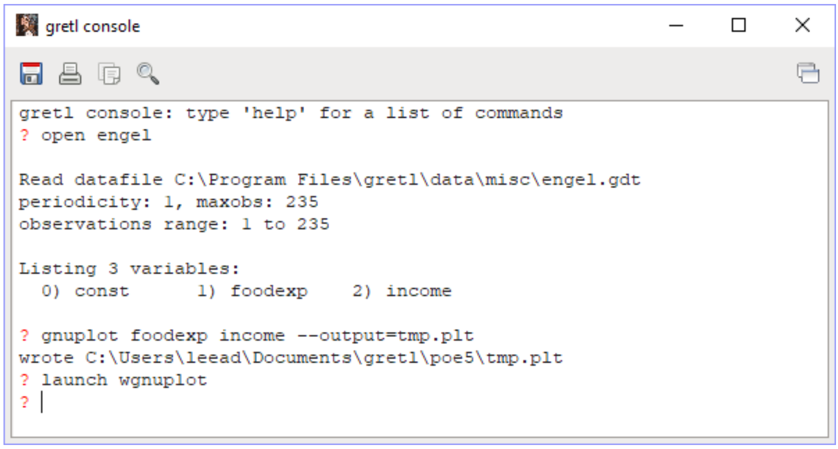
PDF manual preference English (US letter paper)

Figure 1.19: The **General** tab of the **preferences** dialog. To launch **gnuplot** from the console you need to enable **Allow shell commands**.

In MS Windows, open the **gretl** console and type:

```
open engel
gnuplot foodexp income --output=tmp.plt
launch wgnuplot
```

This will look like



```
gretl console: type 'help' for a list of commands
? open engel

Read datafile C:\Program Files\gretl\data\misc\engel.gdt
periodicity: 1, maxobs: 235
observations range: 1 to 235

Listing 3 variables:
  0) const      1) foodexp    2) income

? gnuplot foodexp income --output=tmp.plt
wrote C:\Users\leead\Documents\gretl\poe5\tmp.plt
? launch wgnuplot
? 
```

.

Now, navigate to the **gnuplot** window shown in Figure 1.20 and at the **gnuplot** command prompt type

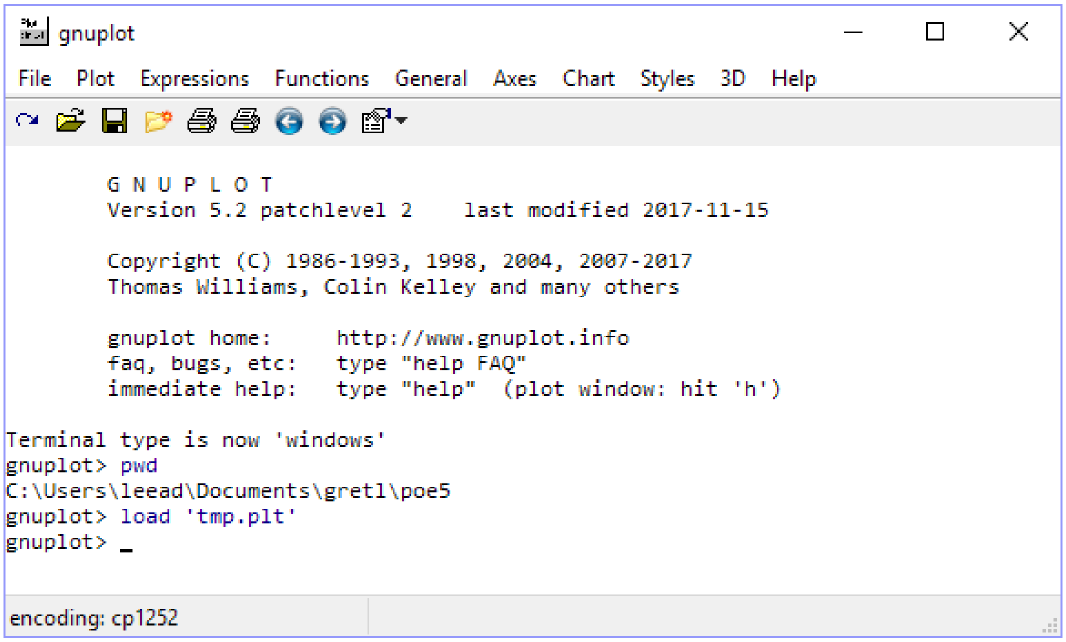
```
pwd
```

This will reveal the current directory and it should should be your working directory, which is the default place where graphs are stored using **gretl**. If not, then use **gnuplot**'s **file>change directory** dialog to get to the desired location. 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 1.20 shows what this looks like.

Another way to do this is to open a command window (Figure 1.2) and type `"C:\Program Files\gretl\wgnuplot"` at the command prompt. The double quotes are necessary since the folder name has a space in it. This will launch the **gnuplot** program shown in Figure 1.20, from which you can search for and open graphs that are written to the harddrive. This implementation has improved since the last version of this manual and is better documented in the **gretl** Users Guide. 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; graphs created in scripts are saved in the working directory by default, but may be directed to the screen using the appropriate option.

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.

There are a number of other types of plots you can make in **gretl**. These include boxplots, histograms, qqplots, mixed frequency time series, and range/mean plots. The underlying engine that generates these is **gnuplot**, but **gretl** gives you easy access to their generation. You can also


gnuplot
File Plot Expressions Functions General Axes Chart Styles 3D Help
encoding: cp1252

```
        G N U P L O T
        Version 5.2 patchlevel 2    last modified 2017-11-15

        Copyright (C) 1986-1993, 1998, 2004, 2007-2017
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'windows'
gnuplot> pwd
C:\Users\leead\Documents\gretl\poe5
gnuplot> load 'tmp.plt'
gnuplot> _
```

Figure 1.20: 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 `'tmp.plt'`.

access **gnuplot** by script through **File**>**Script files**>**New script**>**gnuplot script** from the main menu.

Finally, there is a new set of commands in **gretl** that provide an alternative to the `gnuplot` command. The `plot` block provides may be more convenient when you are producing an elaborate plot (with several options and/or gnuplot commands to be inserted into the plot file). The plot block accepts **gretl** options as well as **gnuplot** commands. The syntax to employ literal **gnuplot** commands in **gretl** is tricky, if only because **gnuplot** commands themselves have their own peculiar syntax. There are many examples in this manual that demonstrate some of these.

# 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. A simple food expenditure model is estimated by least squares. An elasticity is computed, predictions are made, data are graphed and some other statistics computed using least squares results are considered. At the end of the chapter, a simple Monte Carlo simulation is conducted to explore the properties of least squares in repeated sampling.

## 2.1 Simple Linear Regression Model

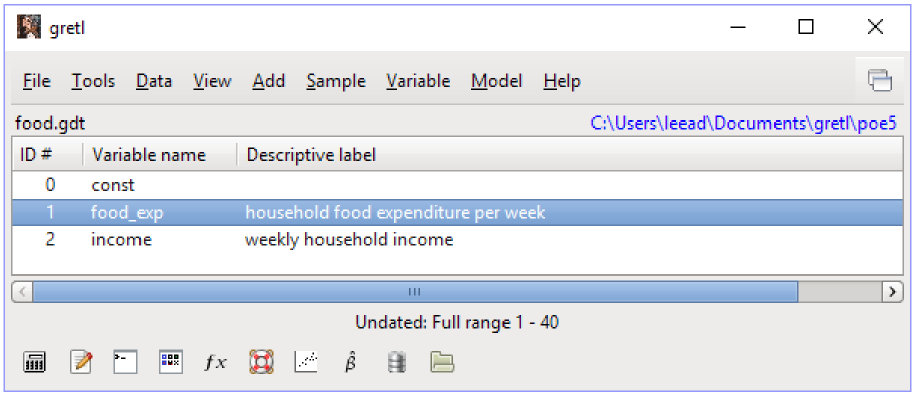
The simple linear regression model is

$$food\_exp_i = \beta_1 + \beta_2 income_i + e_i \quad t = 1, 2, \ldots, n \tag{2.1}$$

where $food\_exp_i$ is the dependent variable, $income_i$ is the independent variable, $e_i$ is random error, and $\beta_1$ and $\beta_2$ are the parameters to be estimated. The errors of the model, $e_i$, have an average value of zero for each value of $income_i$; each has the same variance, $\sigma^2$, and are uncorrelated with any of the other residuals. The independent variable, $income_i$, must take on at least two different values in your dataset. If not, a slope cannot be estimated! The error assumptions can be summarized as $e_i|income_i \; iid \; N(0, \sigma^2)$. 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 C:\Users\leead\Documents\gretl\poe5

| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 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 *POE5*.

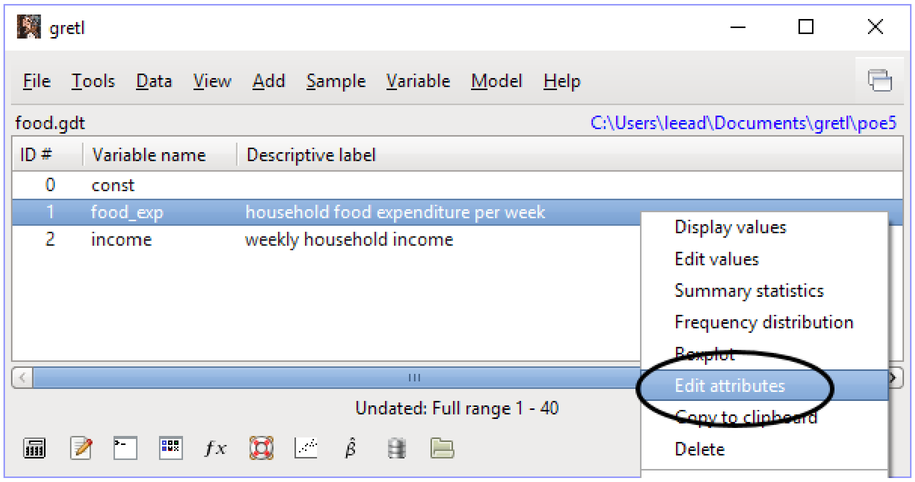
Load the data from the data file *food.gdt*. Recall, this is accomplished by the commands **File>Open data>Sample file** from the menu bar.[1] Choose *food* from the list. When the file containing the data are loaded into **gretl**, the main window will look like the one in Figure 2.1. Notice that the **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 graphing output or generating results for a report or paper, consider adding meaningful labels to your variables to make the output easier to understand. This can be accomplished by editing the attributes of the variables.

To do this, 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 the variable's name can be changed, a description assigned, and a display name given. 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 shortcut, **CTRL+E**. Finally, the `setinfo` command can be used to set the description and the label used in graphs.

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

```
open "@workdir\data\food.gdt"
setinfo food_exp -d "household food expenditure per week" \
    -n "Food Expenditure/Week"
```

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

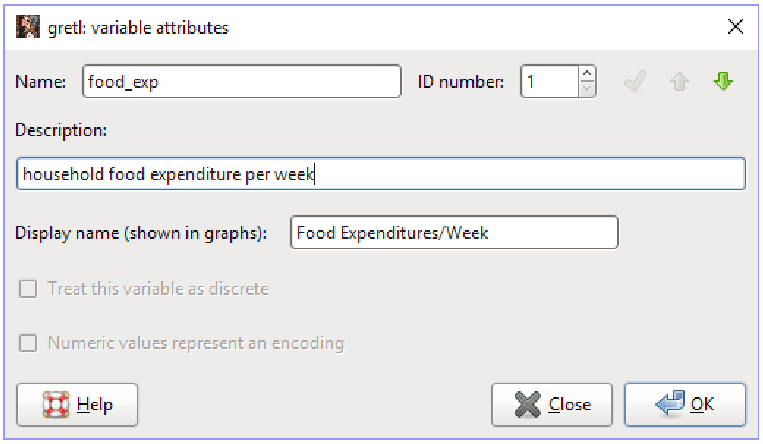


food.gdt C:\Users\leead\Documents\gretl\poe5

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

Undated: Full range 1 - 40

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.



Name: food_exp

ID number: 1

Description:

household food expenditure per week

Display name (shown in graphs): Food Expenditures/Week

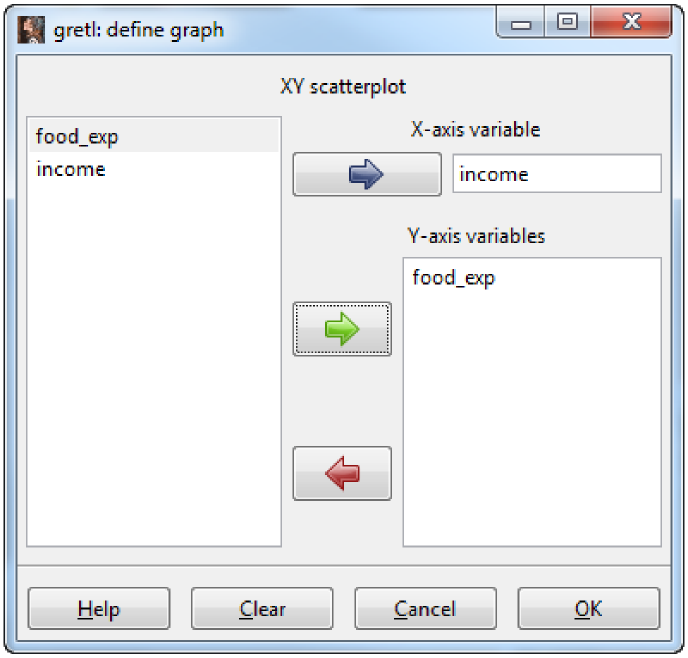
□ Treat this variable as discrete

□ Numeric values represent an encoding

Figure 2.3: Variable edit dialog box

```
4 setinfo income -d "weekly household income" -n "Weekly Income"
5 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 line 2 `setinfo` uses the continuation command (\) since this command is too long to fit on a single line. The `labels` command in line 5 will have **gretl** print the current descriptions to the screen.



gretl: define graph

XY scatterplot

X-axis variable: income

Y-axis variables: food_exp

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

## 2.3 Graph the Data

One way to generate a graph of the food expenditure data that resembles the one in Figure 2.6 of *POE5*, is to use the button on the **gretl** toolbar (fourth icon 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 plots weekly 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.

## 2.4 Estimate the Food Expenditure Relationship

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

$$food\_exp_i = \beta_1 + \beta_2 income_i + e_i \quad t = 1, 2, \ldots, n \tag{2.2}$$

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**

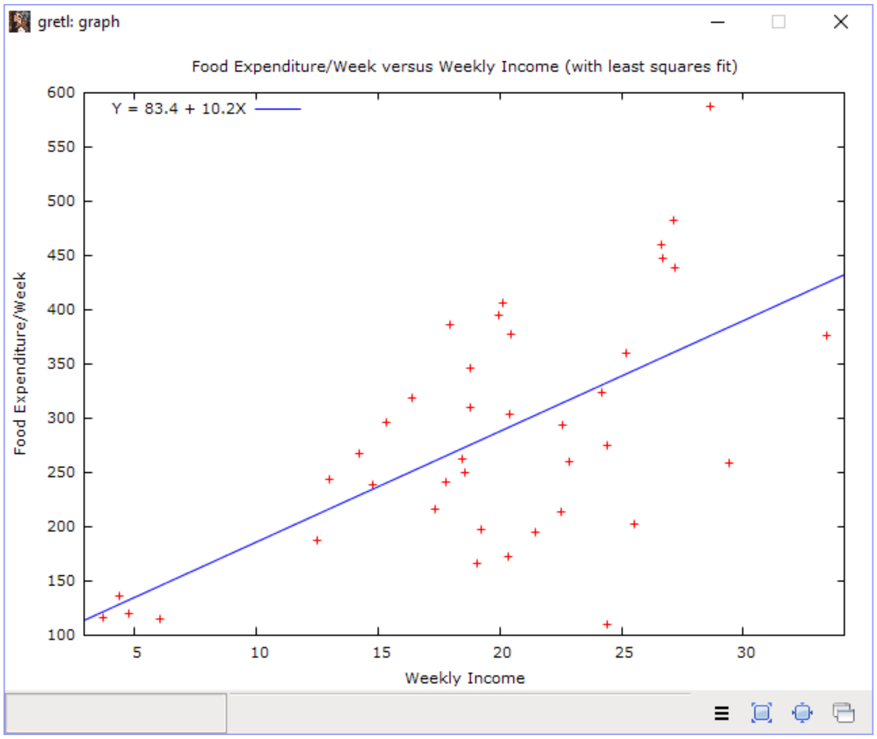


Figure 2.5: XY plot of the food expenditure data

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 ($\beta_1$) 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 green **Add** arrow 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.

At the question mark in the console simply type

```
ols foodexp const income
```

to estimate your regression function. The syntax is very simple, `ols` tells **gretl** that you want to estimate a linear regression 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, `foodexp` and `income`, 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.

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 |

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:

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

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

(standard errors in parentheses)

Finally, notice in the main **gretl** window (Figure 1.4) that the first column has a heading called **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 a lot of typing (and some mistakes). It can also make figuring out which variables are in the model, difficult so choose your poison.

### 2.4.1 Elasticity

Elasticity is an important concept in economics. It measures how responsive 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 $\beta_2$ 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 and equation (2.4) can be computed by hand. This yields the output shown in Figure 2.10.

Or from the console type:

```
 summary foodexp income
```

So, using the numbers from the regression and the summary statistics we get $10.2096*19.605/283.57 = 0.705855$.

This can be made easier by using the **gretl** language to do 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)
```

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 2.11.

### 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:

$$\widehat{food\_exp}_i = 83.42 + 10.21 income_i = 83.42 + 10.21(20) = 287.61 \tag{2.5}$$

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, 287.61.

### 2.4.3 Estimating Variance

In section 2.7 of *POE5*, 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 an estimate of the overall variance of the model's errors, $\sigma^2$. 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 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 $\hat{\sigma}^2$ can be calculated. 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 2.12. 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 *POE5* are presented with results from 10 different regressions (*POE5* 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 "@workdir\data\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 scripts less verbose and easier to modify. In **gretl** lists are series ID numbers and 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 "@workdir\data\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 x
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 2.13

A simple modification of the **hansl** script collects the results of the 10 samples and finds the average values of the estimated coefficients. Simply add the `progressive` option to line 3 as in:

```
3 loop foreach i ylist --progressive
```

This shows how easy it is to conduct a Monte Carlo simulation in **gretl**. This will be discussed at length below in section 2.8.

You can also generate your own random samples and conduct a Monte Carlo experiment using **gretl**. In this exercise 100 samples of data from the food expenditure data are generated, the slope and intercept estimated with each data set, and the sampling performance of the least squares estimator over those 100 different samples is summarized. What will become clear is that the outcome from any single sample is a poor indicator of the true value of the parameters.

Start with the food expenditure model:

$$food\_exp_i = \beta_1 + \beta_2 income_i + e_i \tag{2.6}$$

where $food\_exp_i$ is total food expenditure for the given time period and $income_i$ is income over the same period. 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 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 in a population 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 will 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. 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 $e_i \sim N(0, 88^2)$.

With this knowledge, the properties of the least squares estimator can be studied by generating samples of size 40 using the known data generation mechanism. One hundred food expenditure samples are created using the known parameter values, the model estimated for each using least squares, and then summary statistics are used 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 is unknown 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 $i^{th}$ household is $80 + 10 \times income_i$. 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, $e_i$, from that particular distribution. Repeat this for the remaining 39 individuals. This generates one Monte Carlo sample and which is 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, as many samples of size 40 as desired can be created. Furthermore, since the underlying parameters are for these samples are known, we can determine how close our estimators get to revealing their 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 "@workdir\data\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
```

The first line opens the food expenditure data set that resides in the `data` folder of the working directory. The next line, which is actually not necessary to do the experiments, sets the key, referred to as the **seed**, that initiates the pseudo-random numbers at a specific point. This is useful, since it will allow one to get the same results each time the script runs.

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 desired number of Monte Carlo samples and the option `--progressive` is a command that prevents the output at each iteration from being printed to the results window; the `--quiet` option will suppress some printing to the screen as well.

There are a couple of useful commands that can be added to the program. The `print` command collects (scalar) statistics that you have computed and finds their averages and standard deviations. The `store` command stores these in a **gretl** data file. These are discussed further below.

Within the loop itself, each new sample is generated and instructions are given about how it should be used and where to store desired results. The `series` command generates 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 its 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 2.14. 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 *POE5*. 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 need to be able to create models that allow the independent and dependent variable to be nonlinearly related. Consider the following simple regression

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

The parameter, $\beta_2$ 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 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

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

Obviously, the slope and elasticity depend on the size and price of the home. The user must select values at which these are to be evaluated. This is done in the script below where slopes for houses of size 2000, 4000, and 6000 square feet are computed. The elasticities are computed for prices of \$117,461.77, \$302,517.39, and \$610,943.42. The `scalar` and `series` variable types used here are not strictly necessary in **gretl**. I've used them to make things more clear and it is a good programming practice in general.

```
open "@workdir\data\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

$$\widehat{\text{price}} = \underset{(2890.4)}{55776.6} + \underset{(0.000313)}{0.01542}\ \text{sqft2}$$

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

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

and the graph of home price against size is shown on the righthand side of Figure 2.15.

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 2.16 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 histograms. The code is

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

Finally, the log-linear model is estimated and the predicted values from the regression are plotted against house size.

```
logs price
ols l_price const sqft
series l_yhat = $yhat
series yhat = exp(l_yhat)
gnuplot price yhat sqft  --output=display --suppress-fitted
```

In the first line, an alternative method of generating the natural logarithms is used. The `logs` command can be handy, especially when finding the logarithms of several series; just list all of the desired series after the `logs` command. The regression is estimated in line 2, the predicted values from the regression saved to a new series called `yhat` in line 3, and then converted back to price by taking the antilog in line 4. The price and predicted values are plotted against `sqft` in the last line, with the output sent to the computer display.

The estimated equation is:

$$\ln(\widehat{\text{price}}) = \underset{(0.0246)}{10.839} + \underset{(9.708\text{e--}006)}{0.0004113}\,\text{sqft}$$
$$n = 1080 \quad \bar{R}^2 = 0.6244 \quad F(1,1078) = 1794.8 \quad \hat{\sigma} = 0.32147$$
$$\text{(standard errors in parentheses)}$$

The graph appears on the left-hand side of Figure 2.15. Comparing the log-linear model to the quadratic shows that the nonlinearity estimated by the log-linear is similar, but a bit more pronounced.

Several useful statistics can be generated using these estimates. For instance, a quick prediction could be made about home prices for houses of given size.

$$\widehat{price} = \exp(10.839 + 0.0004113 sqft)$$

At 2000 and 4000 square feet, the simple prediction is:

```
scalar p_2000 = exp($coeff(const)+$coeff(sqft)*2000)
scalar p_4000 = exp($coeff(const)+$coeff(sqft)*4000)
```

which yields `p_2000=115975` and `p_4000=263991`.

Marginal effects

$$\frac{\partial \widehat{price}}{\partial sqft} = \hat{\gamma}_2 \widehat{price} = 0.0004113 \widehat{price}$$

For houses priced at $100,000 and $500,000 this is computed:

```
scalar me_100k = $coeff(sqft)*100000
scalar me_500k = $coeff(sqft)*500000
```

This produces `me_100 = 0.0411269` and `me_500 = 0.205634`.

Elasticities are the marginal effects multiplied by $x/y$. In this model it becomes $\hat{\gamma}_2 sqft$

```
scalar e_2000 = $coeff(sqft)*2000
scalar e_4000 = $coeff(sqft)*4000
```

which yields `e_2000 = 0.822538` and `e_4000 = 1.64508`.

## 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.

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

One can look at the empirical distributions of the two sets of home prices using histograms. In this case, the `smpl` command is used to limit the sample to each of the two cases.

```
1 open "@workdir\data\utown.gdt"
2 smpl utown == 0 --restrict
3 freq price --plot=display --nbins=13
4 smpl utown == 1 --replace --restrict
5 freq price --plot=display --nbins=13
```

In line 2 the `--restrict` option of the `smpl` command is used to restrict the sample to the observations for which the series `utown` is zero. The double equal sign is a logical operator (as opposed to an assignment operator). In this line it checks to see whether the value of `utown` is equal to 0. The `freq` command is used to generate the histogram for the price series. The `--plot=display` option will send the plot to the computer screen and the `--nbins=13` option sets the number of bins for the histogram to 13. The latter ensures that the plots look just like the ones in Figure 2.18 of *POE5*.

The regression model becomes

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

As pointed out in *POE5*, 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

$$E[price|utown] = \beta_1 + \beta_2 utown = \begin{cases} \beta_1 + \beta_2 & \text{if } utown = 1 \\ \beta_1 & \text{if } utown = 0 \end{cases} \tag{2.13}$$

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

$$\widehat{\text{price}} = \underset{(1.3181)}{215.732} + \underset{(1.8296)}{61.5091}\,\text{utown}$$

$$n = 1000 \quad \bar{R}^2 = 0.5306 \quad F(1, 998) = 1130.2 \quad \hat{\sigma} = 28.907$$

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

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 "@workdir\data\utown.gdt"
ols price const utown --quiet
scalar ut = $coeff(const)+$coeff(utown)
scalar other = $coeff(const)
printf "\nThe average home price: \n \
  University Town = $%.2f \n \
  Elsewhere = $%.2f\n", \
  ut*1000,other*1000
```

The output is

```
The average home price:
    University Town = $277241.60
    Elsewhere = $215732.49
```

The last command in this script uses a 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 the next section contains a brief explanation of how to use it.

### 2.7.1 Using printf

The `printf` command can be very useful in programming **gretl** to produce output that is comprehensible and neat. In the preceding example I have combined descriptive text and numerical results. The syntax of `printf` comes from the **C** programming language and it can be a bit tricky to use, so I will try to explain a little about it. I use it extensively in the rest of this book so that you get the used to it. Once used, 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 written to the screen, and the second contains the computations that you want placed within the text.

```
printf "\nThe average home price: \n \
  University Town = $%.2f \n \
  Elsewhere = $%.2f\n", \
  ut*1000,other*1000
```

The first part, called the **format string**, is enclosed in double quotes and occupies the first three lines. The `\n` command stands for 'new line' and it 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.

The \ that follows the line feed is the line continuation command. Putting the entire command on several lines makes it easier to code and harder to make an error.

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 and what kind of format to use for the numerical result you want printed. These formats are also adopted from the **C** programming language. The format `%f` is a fixed point format and the number that falls between the percent sign `%` and the desired format `f` indicates the overall width of what is to be printed and the number decimal places to print. So, `%.2f` tells **gretl** to print only two numbers to the right of the decimal without limiting the overall number of characters for the number. Note, the dollar sign (`$`) that precedes the format command `%.2f`) is actually part of the string that will be printed to the screen (e.g., `University Town = $`).

Recognized numeric formats for the format command are `%s`, `%e`, `%E`, `%f`, `%g`, `%G` and `%d`,[2] 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 each of the format commands. There must 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 `ut` will be printed at the first format command, `%.2f`, and the one in `other` will be printed at the second `%.2f`. Also, note that these can be operated on within the `printf` command. Each of these scalars is being multiplied by 1000.

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()`.

[2] `%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.

Finally, there is a trick to get `printf` to print a percent sign. Since `%` is used to mark the placement of numbers. To print a percent sign it must be preceded by another percent symbol, `%`; hence, `90%%` prints as 90%.

## 2.8 Monte Carlo Simulation

Appendix 2H in *POE5* discusses some of the rudimentary features of Monte Carlo simulations. Figure 2H.1 plots true pdfs for two normal random variables. One is $N(200, 50^2)$ and the other is $N(300, 50^2)$. The essential features of this graph can be generated in **gretl** from the GUI.

From the menu bar select, **Tools>Distribution graphs** from the pull-down menu. This opens the **add distribution graph** dialog shown in Figure 2.17. In this instance we choose the **normal** tab and set **mean** to 200 and **std. deviation** to 50. Click OK. Find the **menu** icon on the graph located in the lower right corner of the graph window. Click on it and select **Add another curve** from the fly-out menu. Then, return to the dialog and change the **mean** to 300 and click OK. This produces the graph shown in Figure 2.18

### 2.8.1 MC Basics

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:

$$E(y_i|\Omega_i) = \beta_1 + \beta_2 x_i \tag{2.14}$$

where $y_i$ is your dependent variable, $x_i$ the dependent variable, $\Omega_i$ the current information set, and $\beta_1$ and $\beta_2$ the parameters of interest. The information set $\Omega_i$ contains $x_i$ as well as other potential explanatory variables that determine the average of $y_i$. The conditional mean of $y_i$ 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 $y_i|\Omega_i$.

To complete 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.

### 2.8.2 A Simple Example

In this example the data generation process will be as follows. We will let $n = 40$ and based on the food expenditure model discussed above.

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

The errors of the model will *iid* $N(0, 88)$. The parameters $\beta_1 = 80$ and $\beta_2 = 10$.

```
# Monte Carlo simulation
open "@workdir\data\food.gdt"
set seed 3213789
loop 1000 --progressive --quiet
    series u = normal(0,88)
    series y1= 80+10*income+u
    ols y1 const income
endloop
```

The *food.gdt* data are loaded and a `seed` for the pseudo-random number generator is chosen. A `progressive` loop of 1000 iterations is initiated. The errors are generated from normals variates having a mean of zero and a standard deviation of 88. These are added to the systematic part of the model that depends on the `income` variable in the data as well as the chosen parameters for the simulation. Finally, the regression is run and the loop closed. The progressive option takes care of collecting results and printing them to the screen.

```
OLS estimates using the 40 observations 1-40
Statistics for 1000 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          79.7886        43.3898        42.6064        5.00518
  income          10.0183        2.09756        2.05451        0.241353
```

You can see that the average estimate of the mean over 1000 samples of size 40 is 79.8, which is very close to our parameter, 80. Likewise the slope is very close to 10.

### 2.8.3 MC using fixed regressors

In this example a set of regressors is generated and used repeatedly to generate new samples of the dependent variable using known parameters. This is what we did in the preceding section using the food expenditure data.

$$y_i = \beta_1 + \beta_2 x_i + e_i \quad i = 1, 2, \cdots, 40. \tag{2.16}$$

In this example we set the intercept $\beta_1 = 100$ and the slope $\beta_2 = 10$. The errors are $N(0, 50^2)$. 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 `nulldata` command opens an empty dataset containing 40 observations. The `series x` is generated using **gretl**'s conditional assignment operator.[3] Here is how it works. The series `x` is 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.

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 **hansl** script is

```
# 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 10000 --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
```

This loops from 1 to 10000 in increments of 1.

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

[3] 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`.

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 10000 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       100.275        25.0830        24.8378        2.86075
        x       9.97793        1.58222        1.57088       0.180930

Statistics for 10000 repetitions
```

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 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:

```
                  mean       std. dev
      b1       100.275        25.0830
      b2       9.97793        1.58222
    sig2       2500.41        574.421
```

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:

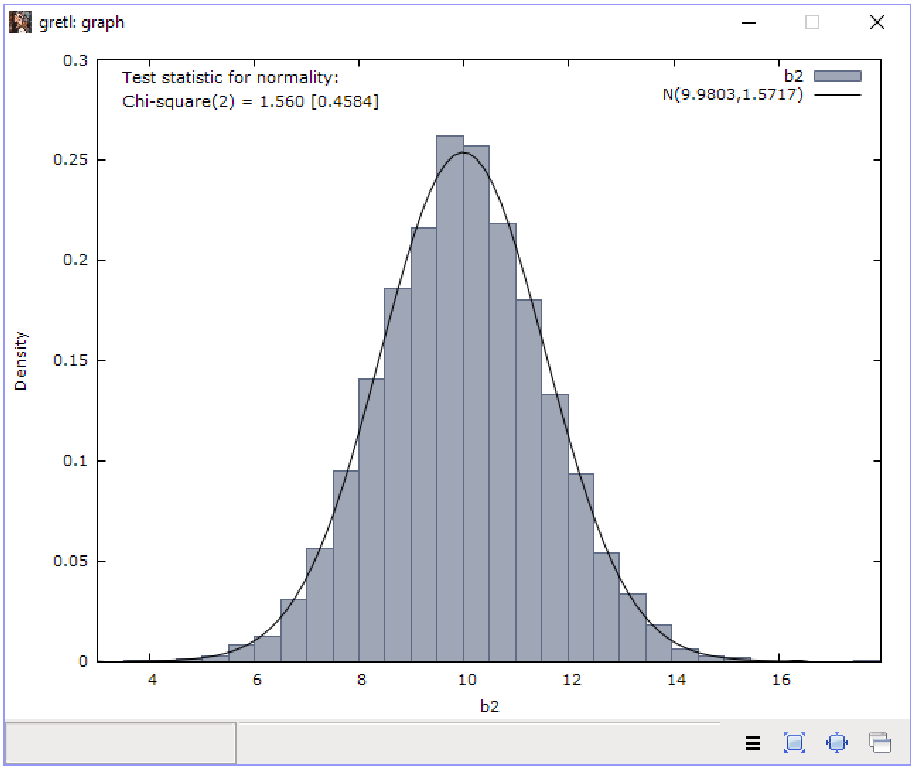
```
store: using filename C:\Users\leead\Documents\gretl\poe5\coef.gdt
wrote C:\Users\leead\Documents\gretl\poe5\coef.gdt
```

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 tells **gretl** to write the file to the currently defined working directory. 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 --plot=display
```

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.

### 2.8.4 MC using random regressors

In this simulation we replace the fixed regressors with random draws from a $N(15, 1.6^2)$ distribution.

```
# Generate systematic portion of model
nulldata 40
loop 10000 --progressive --quiet
    series x = normal(15,1.6)
    series y = 100+10*x + 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_random.gdt" b1 b2 sig2
endloop

open "@workdir\coef_random.gdt"
summary
freq b2 --normal --plot=display
```

The simulation results are:

```
OLS estimates using the 40 observations 1-40
Statistics for 10000 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       101.469        78.2445        76.4855       12.5717
         x       9.89660        5.18802        5.07225       0.835130
```

Although the means have not changed much, the coefficients are much more variable when the regressors are random. The standard deviation of the coefficients is roughly three times what is was in the fixed regressor case. The results are quite similar to those in Table 2H.2 in *POE5*.

## 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.

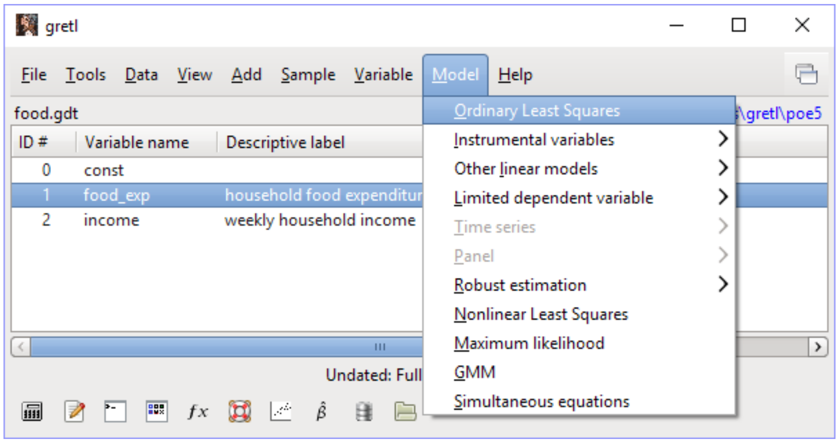
```
set echo off
open "@workdir\data\food.gdt"
```

```
setinfo food_exp -d "household food expenditure per week" \
    -n "Food Expenditure/Week"
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 --output=display

#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 "@workdir\data\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

#Find the averages using progressive loop
open "@workdir\data\table2_2.gdt"
list ylist =  y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
loop foreach i ylist --progressive
    ols ylist.$i const x
endloop

# slopes and elasticities at different points
open "@workdir\data\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

# estimate the quadratic model
open "@workdir\data\br.gdt"
square sqft
ols price const sqft
ols price const sq_sqft
series yhat = $yhat
gnuplot price yhat sqft  --output=display --suppress-fitted

# Example 2.7
# estimate the log-linear model
logs price
ols l_price const sqft
series l_yhat = $yhat
series yhat = exp(l_yhat)
# Figure 2.17
gnuplot price yhat sqft  --output=display --suppress-fitted

# marginal effects at $100,000 and $500,000
scalar me_100k = $coeff(sqft)*100000
scalar me_500k = $coeff(sqft)*500000
# predicted prices at 2000 and 4000 square feet
scalar p_2000 = exp($coeff(const)+$coeff(sqft)*2000)
scalar p_4000 = exp($coeff(const)+$coeff(sqft)*4000)
# elasticity at 2000 and 4000 square feet
scalar e_2000 = $coeff(sqft)*2000
scalar e_4000 = $coeff(sqft)*4000
# semi-elasticity
scalar se = $coeff(sqft)*100

# generate Figure 2.18 in POE5
open "@workdir\data\utown.gdt"
smpl utown = 0 --restrict
freq price --show-plot --nbins=13
smpl utown = 1 --replace --restrict
freq price --show-plot --nbins=13

# regression using indicator variables
open "@workdir\data\utown.gdt"
logs price
ols l_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

# Appendix 2H.4
open "@workdir\data\mc1_fixed_x.gdt"
```

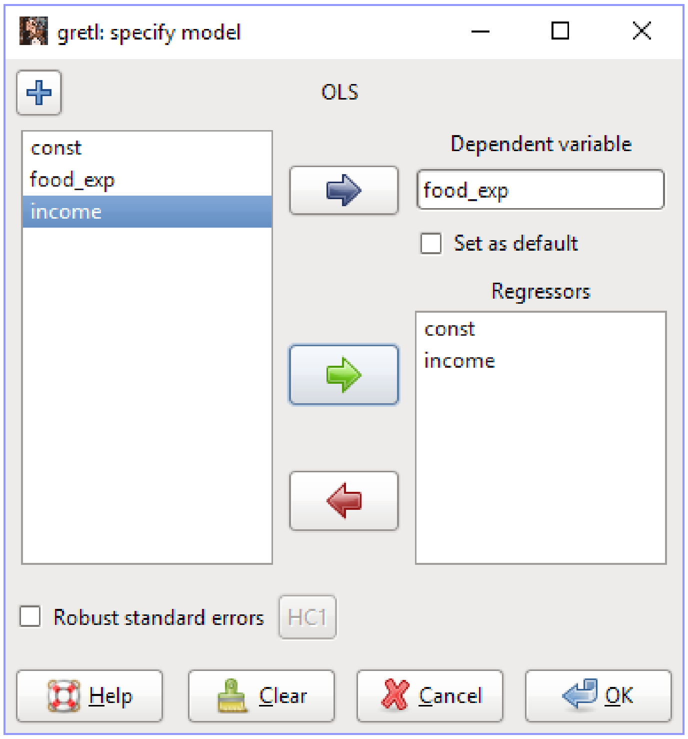
```
ols y const x


# Monte Carlo simulation
open "@workdir\data\food.gdt"
set seed 3213789
loop 1000 --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 10000 --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 --plot=display

# Monte Carlo simulation #3
# Generate systematic portion of model
nulldata 40
loop 10000 --progressive --quiet
    series x = normal(15,1.6)
    series y = 100+10*x + 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_random.gdt" b1 b2 sig2
endloop

open "@workdir\coef_random.gdt"
summary
```

```
freq b2 --normal --plot=display
```



| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | food_exp | household food expenditur |
| 2 | income | weekly household income |

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



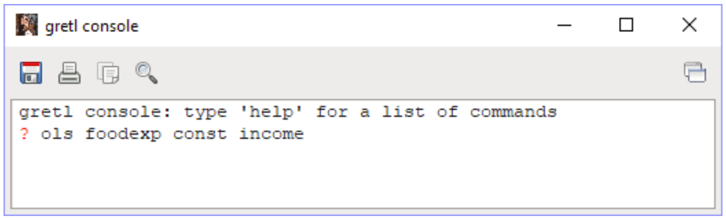
Dependent variable: food_exp

□ Set as default

Regressors: const, income

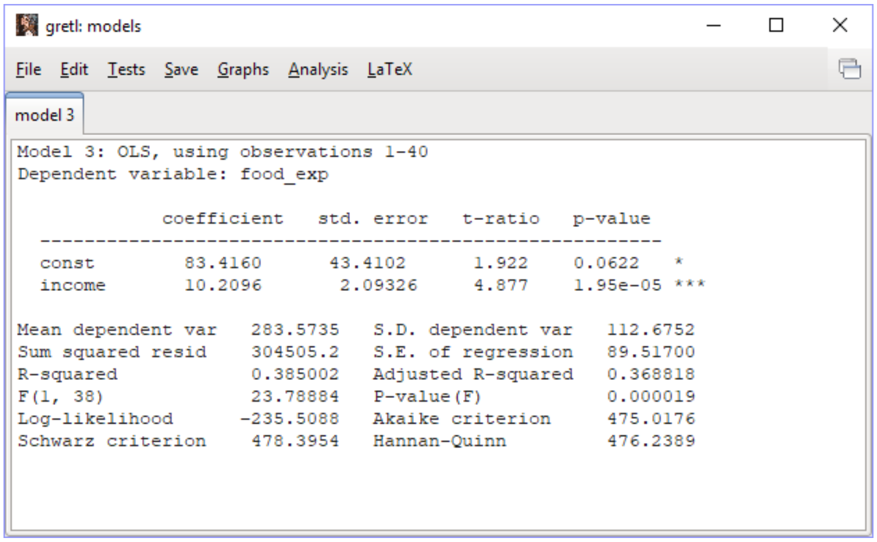
□ Robust standard errors HC1

Figure 2.7: The specify model dialog box opens when you select **Model>Ordinary least squares**



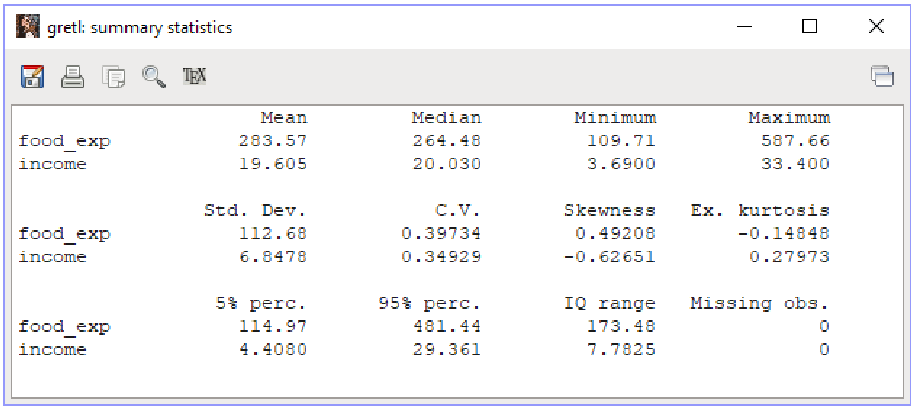
```
gretl console: type 'help' for a list of commands
? ols foodexp const income
```

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 commands.



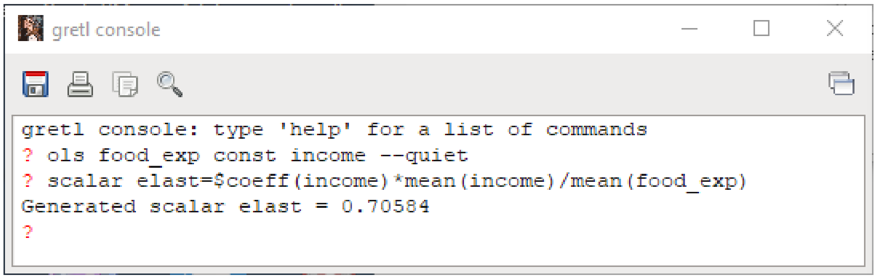
```
Model 3: OLS, using observations 1-40
Dependent variable: food_exp

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



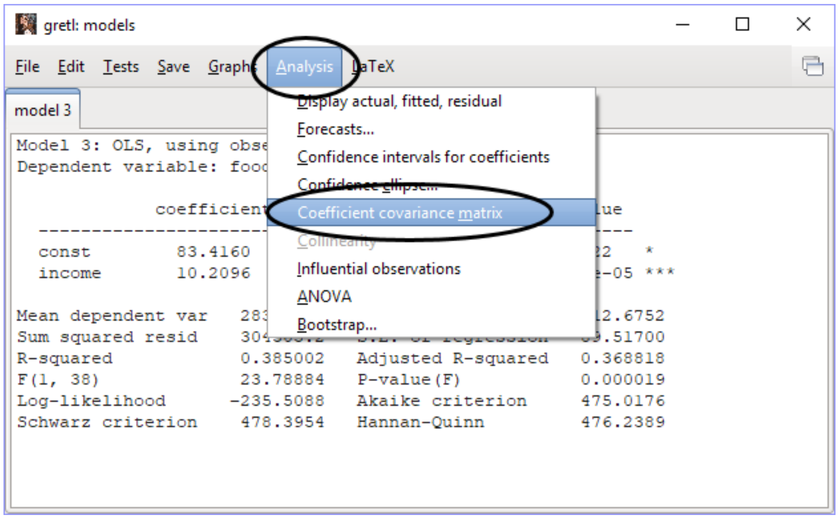
```
                        Mean         Median        Minimum        Maximum
food_exp              283.57         264.48         109.71         587.66
income                19.605         20.030         3.6900         33.400

                   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

                    5% perc.      95% perc.       IQ range   Missing obs.
food_exp              114.97         481.44         173.48              0
income                4.4080         29.361         7.7825              0
```

Figure 2.10: Summary statistics



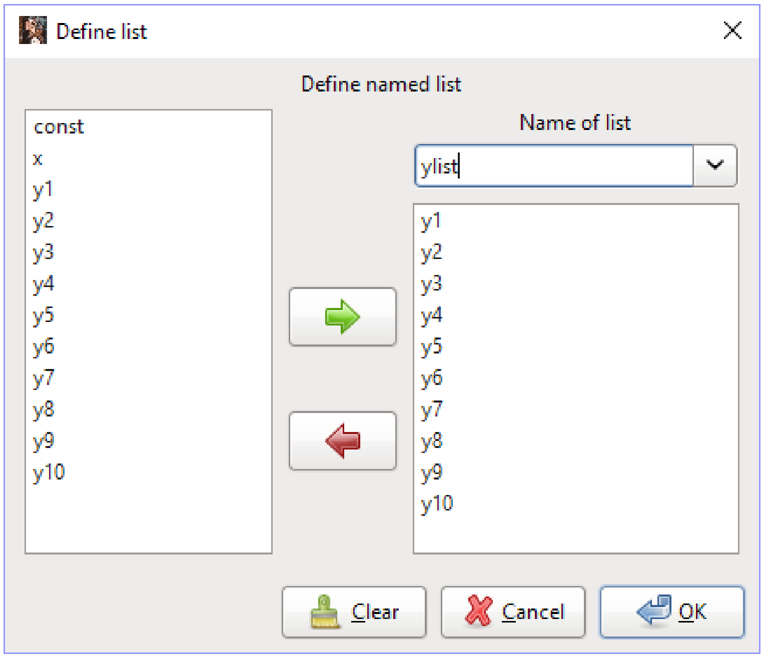
```
gretl console: type 'help' for a list of commands
? ols food_exp const income --quiet
? scalar elast=$coeff(income)*mean(income)/mean(food_exp)
Generated scalar elast = 0.70584
?
```

Figure 2.11: Results from commands written to the console that compute an elasticity based on a linear regression.



```
Model 3: OLS, using obse
Dependent variable: fooc

             coefficien                                  ue
  -------------------------                          -----
  const         83.4160                              22   *
  income        10.2096                              e-05 ***

Mean dependent var   283                             12.6752
Sum squared resid    304                             .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 2.12: Obtain the matrix that contains the least squares estimates of variance and covariance from the pull-down menu of your estimated model.



Name of list: ylist

y1
y2
y3
y4
y5
y6
y7
y8
y9
y10

Figure 2.13: 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 2.14: The summary results from 100 random samples of the Monte Carlo experiment.

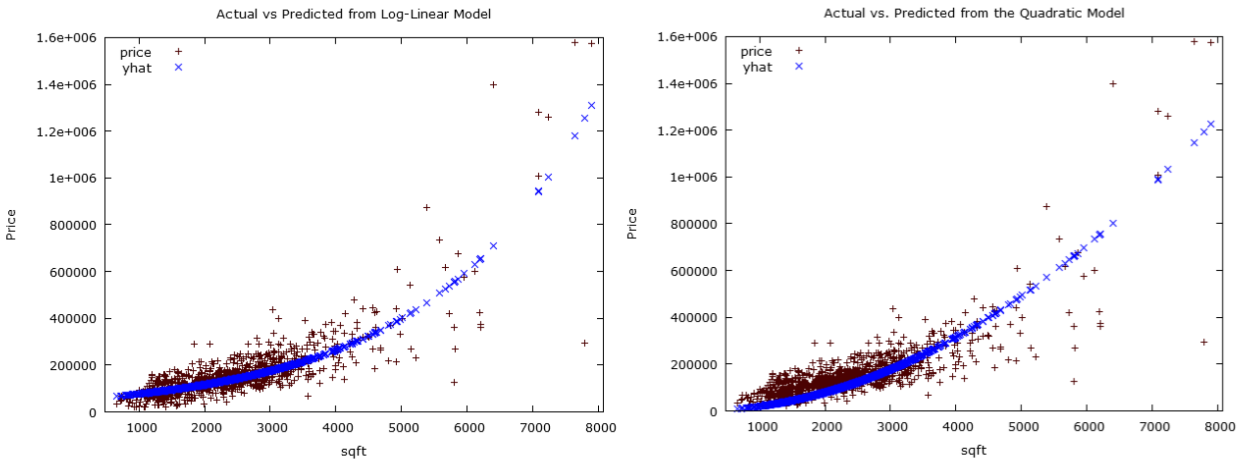


Figure 2.15: Price versus size from log-linear and quadratic models.

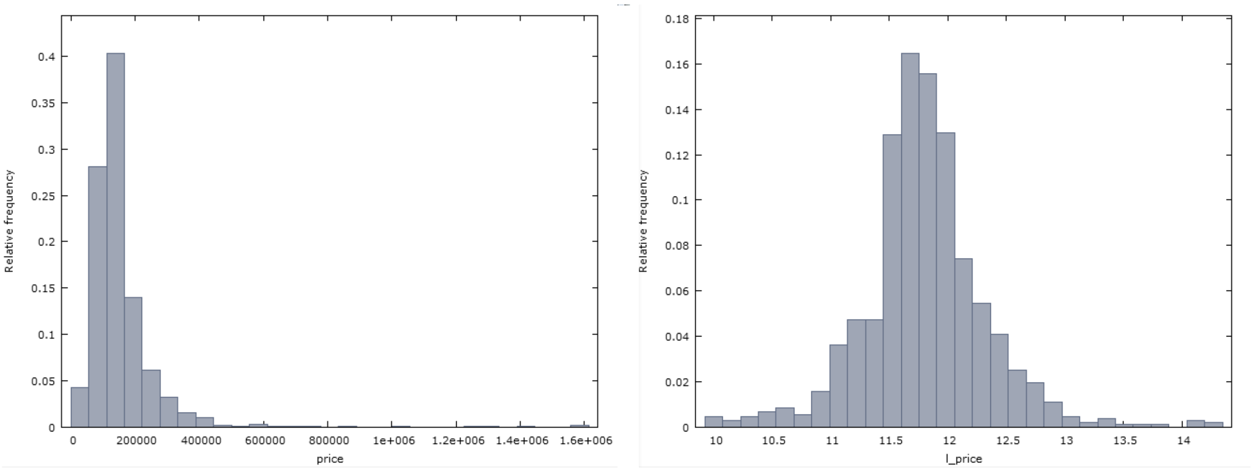
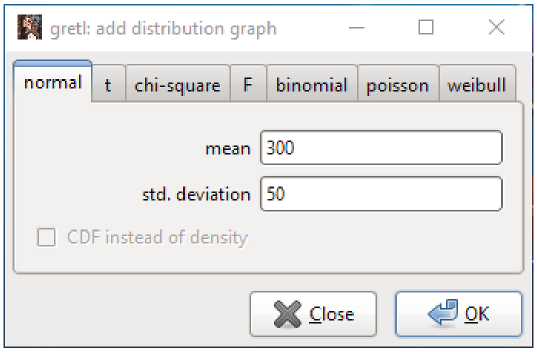


Figure 2.16: Price and its natural logarithm.



mean 300

std. deviation 50

□ CDF instead of density

Figure 2.17: This dialog allows you to graph various distributions. It can be used multiple times to overlay graphs.

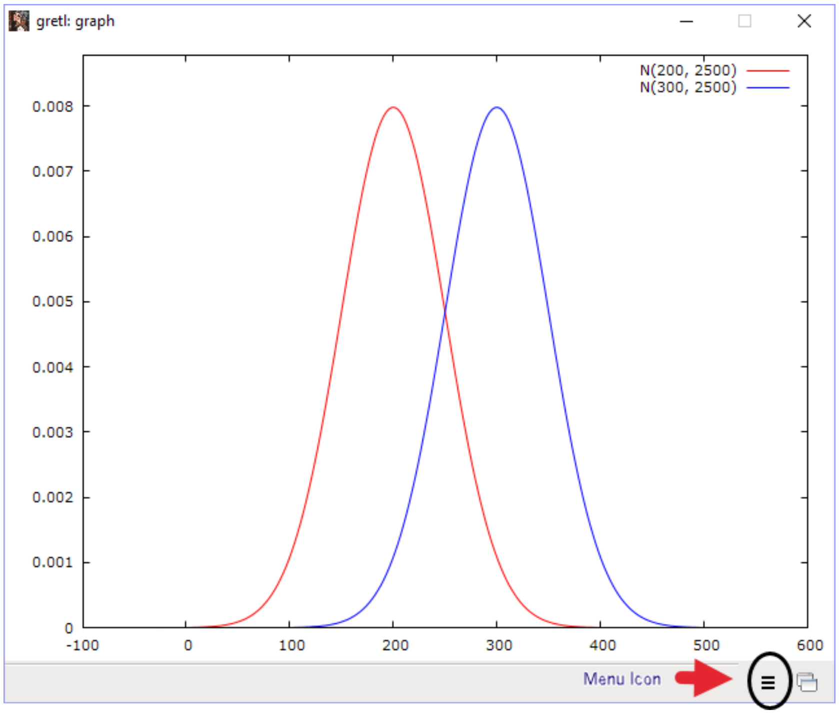


Figure 2.18: Figure 2H.1 in *POE5* plots two normal distributions having different means. Using the menu icon select **Add another curve** before returning the **Add distribution graph** dialog to insert the second graph. This similar graph is produced using one of the **Tools** in **gretl**.

# 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 frequency in repeated samples.[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 stored 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.

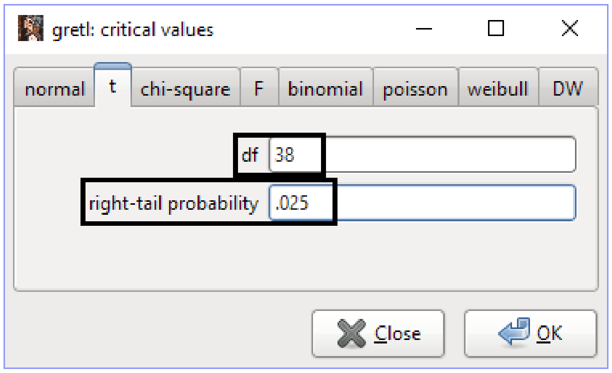
[1] This is probability in the frequency sense. Some authors fuss over the exact interpretation of a confidence interval (unnecessarily I think). You are often given stern warnings not to interpret a confidence interval as containing the unknown parameter with the given probability. However, the frequency 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 *POE5*

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

Recall that $b_k$ is the least squares estimator of $\beta_k$, 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 $\alpha$ is the total desired probability associated with the "rejection" area (the area outside of the confidence interval).

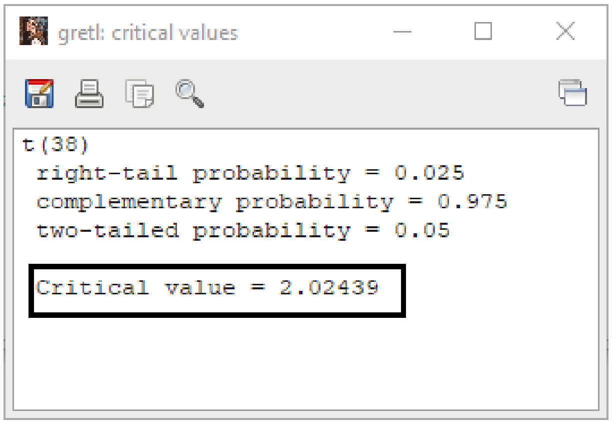
You'll need to know the critical value $t_c$, which can be obtained from a statistical table, the **Tools>Statistical tables** dialog contained in the program, or using the **gretl** command `critical`. First, try using the dialog box shown in Figure 3.1. Pick the tab for the $t$ distribution and tell **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]



df 38

right-tail probability .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.

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

### Example 3.1 in *POE5*

This example is based on the food expenditure model first considered in Chapter 2.

$$food\_exp_i = \beta_1 + \beta_2 income_i + e_i \quad i = 1, 2, \ldots, n$$

The goal is to estimate a 95% confidence interval for the slope, $\beta_2$. Using a combination of accessors and output from the critical value finder dialog we can generate the lower and upper bounds (using the **gretl** console) with the commands:

```
1 open "@workdir\data\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 $\beta_2$. The last line prints the results of the computation.

The **gretl** language syntax needs a little explanation. When **gretl** makes a computation, it will store certain results like coefficient estimates, their standard errors, sum of squared errors in volatile memory. These results can be accessed and used to compute other statistics, provided you know the accessor's name. These so-called `accessors` carry a `$` prefix and a list of what can be accessed after estimation can be found in the function reference or by using `varlist --type=accessor`. Lines 3 and 4 use accessors for the coefficients (`$coeff(income)`) and standard errors (`$stderr(income)`) of the variable in parentheses. The list of accessors is growing rapidly in response to user requests, so I recommend checking it whenever you are looking for a stored result to use in a computation.

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 $\beta_2$ is estimated using the variable `income`, its coefficient estimate is saved in `$coeff(income)`. The corresponding standard error is saved in `$stderr(income)`. Consult the function reference (Figure 1.15) to see 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:

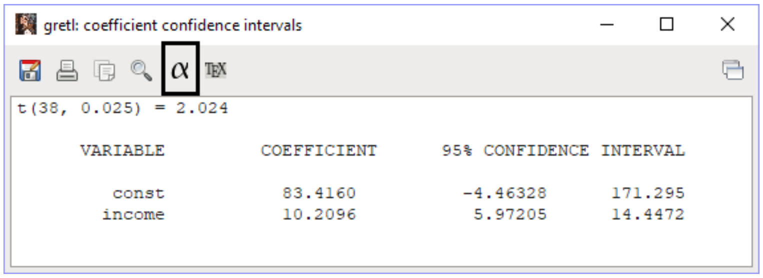
```
open "@workdir\data\food.gdt"
ols food_exp const income
scalar lb = $coeff(income) - critical(t,$df,0.025) * $stderr(income)
scalar ub = $coeff(income) + critical(t,$df,0.025) * $stderr(income)
print lb ub
```

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

The example found in section 3.1.3 of *POE5* 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 below.

```
Replaced scalar lb = 5.97205
Replaced scalar ub = 14.4472

            lb =  5.9720525

            ub =  14.447233
```

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 (Figure 2.7) and click **OK**. The results appear in the **models window** (Figure 2.9). Now choose **Analysis>Confidence intervals for coefficients** from the models window's pull-down menu to generate the result shown in Figure 3.3. The boxed $\alpha$ icon can be used to change the size of the confidence interval, which can be set to any (integer) percentage level you desire.



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

### A gretl Function to Compute Confidence Intervals

Since confidence intervals like this based on a $t$-distributed random variable are common, I wrote a simple program to produce them with minimal effort and to provide better looking output. **This function is used throughout the remainder of this manual** and can be found in the following section.

Since confidence intervals are computed for many models, it is worth writing a function in **gretl** that can be reused. The use of functions to perform repetitive computations makes programs shorter and reduces errors (unless your function is wrong, in which case every computation is incorrect!) In the next section, **gretl** functions are introduced and one that computes the model selection rules discussed above is presented.

## 3.2 Functions in gretl

Gretl provides a mechanism for defining **functions**, which may be called via the console, in the context of a script, or (if packaged appropriately) via the programs graphical interface. The syntax for defining a function is:

```
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 the following types: `void` (if the function does not return anything), `scalar`, `series`, `matrix`, `list`, `string` or `bundle`.
3. **function-name**, the unique identifier for the function. Names must start with a letter. They have a maximum length of 31 characters; anything longer 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 variables (scalars, matrices, etc.) the same name as one of your functions.
4. The functions 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 confidence interval function is designed to compute a $1 - \alpha\%$ confidence interval centered at a $t$-distributed random variable and print the results to the screen. Its basic structure is:

```
function void t_interval (scalar b, scalar se, scalar df, scalar p)
    [some computations]
    [print results]
    [return results]
end function
```

As required, it starts with the keyword `function`. The next word, `void`, indicates that the function will returned nothing when used. The next word is `t_interval`, which is the name given to the function. The `t_interval` function has four arguments that will be used as inputs. The first, `b`, is a $t$-distributed scalar statistic that is the interval's center, next is a scalar, `se`, that contains the estimated standard error of `b`, `df` is a scalar for the degrees of freedom, and `p` is the desired coverage probability of the interval. The inputs are separated by a comma and there are spaces between the list of inputs.

```
 1 function void t_interval(scalar b "statistic for interval’s center",
 2     scalar se "standard error of b",
 3     scalar df "degrees-of-freedom for the t-distribution",
 4     scalar p  "coverage probability for the interval")
 5     scalar alpha = (1-p)
 6     scalar lb = b - critical(t,df,alpha/2)*se
 7     scalar ub = b + critical(t,df,alpha/2)*se
 8     printf "\nThe %.2f confidence interval centered at %.2f is\
 9 (%.2f, %.2f)\n", p, b, lb, ub
10 end function
```

In line 5 the `p` is converted to $\alpha$ to be used in the critical value function inputs. Lines 6 and 7 compute the bounds of the interval and the final statement, `printf` produces output to the screen.[3]

At this point, the function can be highlighted and run. Then, run the regression and call the function using the appropriate arguments.

```
ols food_exp const income
t_interval($coeff(income),$stderr(income),$df,.95)
```

which produces the output:

```
The 95% confidence interval centered at 10.21 is (5.972, 14.447)
```

The function performs as expected.

[3] See section 2.7.1 for information on how to use `printf`.

## 3.3 Repeated Sampling

### Tables 3.1 and 3.2 in *POE5*

In this section, ten samples found 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:

```
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 --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 the user designated working directory `@workdir` on your harddrive.

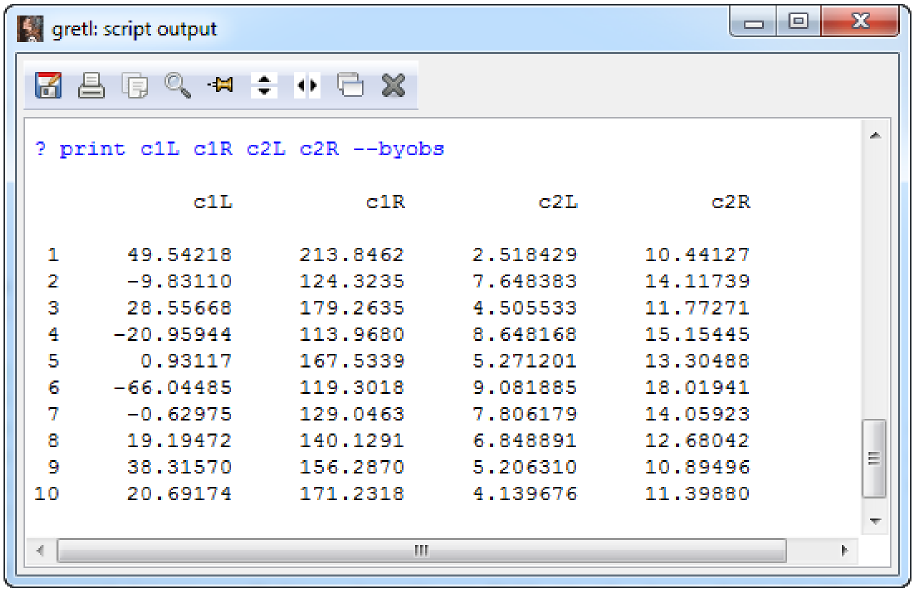
As if that is not easy enough, there is an even simpler syntax that will accomplish the same thing. It uses the fact that the dependent variables all begin with the letter 'y' and have number suffixes. In this case the `foreach` loop can be simplified by replacing lines 2-4 with:

```
2  list ylist = y*        # use the wildcard
3  loop foreach i ylist --progressive
4     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 Table 3.2 of *POE5*.

```
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 3.4. 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 $\beta_1$ falls also falls within the estimated boundaries in each of the samples. In a large number of samples, we expect about 5% of the intervals will not contian the true value of the parameters. This is explored in the next simulation.



```
? 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 3.4: Confidence intervals for 10 samples.

## 3.4 Monte Carlo Experiment

Once again, the consequences of repeated sampling can be explored using a simple Monte Carlo study. In this case, 100 samples are generated and we 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. A more thorough set of experiments can be found in sections 3.7.1 and 3.7.2 .

The new script looks like this:

```
 1 open "@workdir\data\food.gdt"
 2 set seed 3213798
 3
 4 loop 100 --progressive --quiet
 5     series u = normal(0,88)
 6     series y = 80 + 10*income + u
 7     ols y const income
 8
 9     scalar c1L = $coeff(const) - critical(t,$df,.025)*$stderr(const)
10     scalar c1R = $coeff(const) + critical(t,$df,.025)*$stderr(const)
11     scalar c2L = $coeff(income) - critical(t,$df,.025)*$stderr(income)
12     scalar c2R = $coeff(income) + critical(t,$df,.025)*$stderr(income)
13
14     # Compute the coverage probabilities of the Confidence Intervals
15     scalar p1 = (80>c1L && 80<c1R)
16     scalar p2 = (10>c2L && 10<c2R)
17
18     print p1 p2
19     store @workdir\cicoeff.gdt c1L c1R c2L c2R
20 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 *POE5*. In line 5 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 6. A regression is estimated. Then, the upper and lower bounds are computed. In lines 15 and 16 **gretl**'s "and" logical operator, `&&`, is used to determine whether the coefficient (80 or 10) falls within the computed bounds. The operator `&&` yields the intersection of two sets so 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 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

```
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         84.9634        45.3240        41.7692       4.92305
  income         9.76211        2.15209        2.01414       0.237393
```

```
Statistics for 100 repetitions

                 mean        std. dev
      p1      0.950000       0.217945
      p2      0.940000       0.237487
```

You can see that the intercept falls within the estimated interval 95 out of 100 times and the slope within its interval 94% of the time.

## 3.5 Hypothesis Tests

Hypothesis tests allow us to compare what we assume to be true with what we observe through data. Suppose that I believe that autonomous weekly food expenditure is no less than $40, I draw a sample, compute a statistic that measures food expenditure, and then compare my estimate to my conjecture using a hypothesis test.

### 3.5.1 One-sided Tests

In section 3.4 of *POE5* the authors test several hypotheses about $\beta_2$ in the food expenditure model. One null hypothesis is that $\beta_2 = 0$ against the alternative that it is positive (i.e., $\beta_2 > 0$). The test statistic is:

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

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 the $t$-statistic falls within the rejection region of the test; that is if it is larger than 1.686.

The required information 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 |

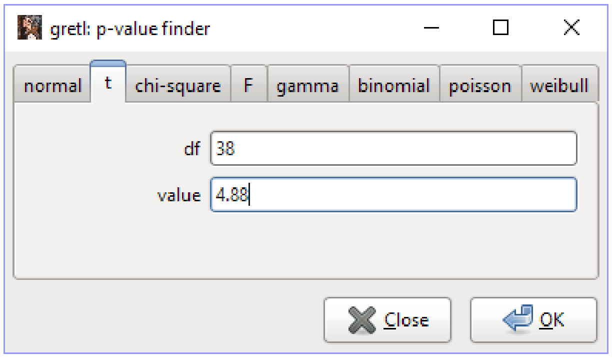
| | | | |
|---|---|---|---|
| Mean dependent var | 283.5735 | S.D. dependent var | 112.6752 |
| Sum squared resid | 304505.2 | S.E. of regression | 89.51700 |
| $R^2$ | 0.385002 | Adjusted $R^2$ | 0.368818 |
| $F(1, 38)$ | 23.78884 | P-value($F$) | 0.000019 |
| Log-likelihood | −235.5088 | Akaike criterion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

The computations

$$t = (b_2 - 0)/se(b_2) = (10.21 - 0)/2.09 = 4.88$$

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.

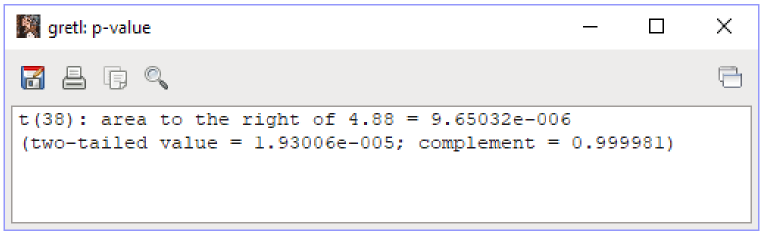
Gretl is used to get the $p$-value for this test using the **Tools** pull-down menu (Figure 3.5). In this dialog, you enter the desired degrees of freedom for your $t$-distribution (`38`), the value of $b_2$ (`10.21`), its value under the null hypothesis–something **gretl** refers to as 'mean' (`0`), and the estimated standard error from the printout (`2.09`). This yields the information in Figure 3.6: The area of a $t_{38}$ random variable to the right of 4.88, i.e., the $p$-value of the test, is almost zero. Since the $p$-value is well below $\alpha = .05$, the hypothesis is rejected.



df 38

value 4.88

Figure 3.5: The dialog box for obtaining $p$-values using the built in statistical tables in **gretl**.



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

Figure 3.6: The result produced by the $p$-value finder dialog box.

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`. It returns a scalar equal to the size of the computed area. Distribution choices include $z$ for Gaussian, $t$ for Student's t, $X$ for chi-square, F for $F$, $G$ for gamma, $B$ for binomial, $L$ for Laplace, $P$ for Poisson, $W$ for Weibull, or $E$ for generalized error. There are values for some non-central distributions as well ($\chi$, F, t). The argument `parameters` refers to the distribution's **known** parameters, as in its degrees of freedom. So, for this example try

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

# Example 3.2
scalar t2 = ($coeff(income)-0)/$stderr(income)
scalar p2 = pvalue(t,$df,t2)
printf "\nHa: b2>0  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio1, c2, p2
```

The result is

```
Ha: b2>0
  t = 4.88, critical value = 1.69
  alpha = 0.05, p-value = 0.000
```

The $p$-value (9.72931e-006) is rounded to zero and 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.

### Example 3.3 in *POE5*

In example 3.3, the authors of *POE5* test the hypothesis that $\beta_2 = 5.5$ against the alternative that $\beta_2 > 5.5$. The computations

$$t = (b_2 - 5.5)/se(b_2) = (10.21 - 5.5)/2.09 = 2.25$$

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 from the critical values window is shown below:

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

 Critical value = 2.42857
```

The 0.01 one-sided critical value is 2.42857. Since 2.25 is less than this, we cannot reject the null hypothesis at the 1% level of significance.

Example 3.3 is verified using the following **hansl** script

```
# Example 3.3
#One sided test (Ha:  b2>5.5)
scalar tratio2 = ($coeff(income) - 5.5)/ $stderr(income)
scalar c2 = critical(t,$df,.01)
scalar p2 = pvalue(t,$df,tratio2)
printf "\nHa: b2>5.5  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.01, p-value = %.3f\n", tratio2, c2, p2
```

The output printed to the screen is:

```
Ha: b2>5.5
  t = 2.25, critical value = 2.43
  alpha = 0.01, p-value = 0.015
```

### Example 3.4 in *POE5*

In example 3.4 of *POE5*, the authors conduct a one-sided test where the rejection region lies in 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 under the null hypothesis is

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

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

$$t = (b_2 - 15)/se(b_2) = (10.21 - 15)/2.09 = -2.29$$

Based on the desired level of significance, $\alpha = 0.05$, we reject the null in favor of the one-sided alternative since $t < -1.686$.

The **hansl** script to test this hypothesis is shown below:

```
# Example 3.4
#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 "\nHa: b2<15  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio3, c3, p3
```

This yields:

```
Ha: b2<15
  t = -2.29, critical value = -1.69
  alpha = 0.05, p-value = 0.014
```

The $p$-value of 0.014 is less than 5% and we conclude that the coefficient is less than 15 at this level of significance.

### 3.5.2 Two-sided Tests

#### Example 3.5 in *POE5*

Two-sided tests are explored in examples 3.5 and 3.6 of *POE5*. 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 $\beta_2$ is \$7.50. There simply isn't enough information in the sample to convince us otherwise.

```
# Example 3.5
#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 "\nHa: b2 not equal 7.5  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio4, c4, p4
```

```

#Confidence interval
t_interval($coeff(income),$stderr(income),$df,.95)
```

You can draw the same conclusions from using a confidence interval that you can from this two-sided $t$-test.

$$b_2 - t_c se(b_2) \le \beta_2 \le b_2 + t_c se(b_2)$$

The test results and confidence interval produced by the **hansl** script are:

```
Ha: b2 not equal 7.5
  t = 1.29, critical value = 2.02
  alpha = 0.05, p-value = 0.203

The 0.95 confidence interval centered at 10.21 is (5.97, 14.45)
```

From a hypothesis testing standpoint, 7.5 falls within this interval and you would not be able to reject the hypothesis that $\beta_2$ is different from 7.5 at the 5% level of significance.

### Example 3.6 in *POE5*

In example 3.6 a test of the overall significance of $\beta_2$ 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**.

```
# Example 3.6
#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 "\nHa: b2 not equal 0  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio5, c5, p5
```

This produces:

```
Ha: b2 not equal 0
```

```
t = 4.88, critical value = 2.02
alpha = 0.05, p-value = 0.000
```

which tells us that $\beta_2$ is significantly different from zero at 5%.

## 3.6 Linear Combination of Parameters

### Examples, 3.7, 3.8 and 3.9 in *POE5*

These examples use an estimate of expected food expenditures for a family with $2,000 per week of income. In Example 3.7 the expected expenditure is obtained, in 3.8 a 95% confidence interval for weekly food expenditure is obtained and in 3.9 a test is used to determine whether food expenditure exceeds $250 per week.

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. The average weekly food expenditure for a family earning $2000 per week based on the model is:

$$E(\mathit{food\_exp}|\mathit{income}) = \beta_1 + \beta_2 \mathit{income} \tag{3.2}$$

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

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

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

$$H_0 : \beta_1 + \beta_2 \leq 0 \quad H_1 : \beta_1 + 20\beta_2 > 250$$

The statistic

$$t = \frac{b_1 + 20b_2 - 250}{se(b_1 + 20b_2 - 250)} \sim t_{n-2} \text{ under } H_0 \tag{3.3}$$

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.

$$cov(b_1, b_2) = \begin{bmatrix} var(b_1) & cov(b_1, b_2) \\ cov(b_1, b_2) & var(b_2) \end{bmatrix} \tag{3.4}$$

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 the model estimation command 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.

```
1  ols food_exp const income
2  scalar avg_food_20 = $coeff(const)+20*$coeff(income)
3  scalar vc = $vcv[1,1]+20^2*$vcv[2,2]+2*20*$vcv[2,1]
4  scalar se = sqrt(vc)
5  scalar tval = ($coeff(const)+20*$coeff(income)-250)/se
6  scalar p = pvalue(t,$df,tval)
7  scalar crit = critical(t,$df,.025)
8
9  t_interval(avg_food_20,se,$df,.95)
10 printf "\nHa: Average weekly Foodexp|Income=2000 > 250 \n \
11 Average Expenditure = %3.2f, Standard Error = %.3f \n \
12 t = %.2f, critical value = %.2f \n \
13 alpha = 0.05, p-value = %.3f\n", avg_food_20, se, tval, crit, p
```

In the first line, the model is estimated. In line 2 average food expenditure when income is equal to $2000 is computed (income is measured in $100). In line 3 the accessor `$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.

The lower and upper 95% confidence intervals are computed in line 9 using the `t_interval` function that we defined earlier. Lines 10-13 generate the output for printing to the screen using the `printf` function.

```
The 0.95 confidence interval centered at 287.61 is (258.91, 316.31)

Ha: Average weekly Foodexp|Income=2000 > 250
  Average Expenditure = 287.61, Standard Error = 14.178
  t = 2.65, critical value = 2.02
  alpha = 0.05, p-value = 0.006
```

The 95% confidence interval for the average is ($258.91, $316.31). You can see that the manual calculations and those from the **hansl** script 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 exceeds $250.

## 3.7 Monte Carlo Simulations

### 3.7.1 Fixed Regressors

#### Appendix C3.3 of *POE5*

This simulation uses the experimental design from section 2.8.3. Hence, the same values of the regressors are used in each of the 10000 samples drawn. Several scalars are computed to measure the properties of the confidence intervals and tests. The scalars `p1` and `p2` take the value 1 when the compound statement in parentheses is true. That means, for instance, `p1`=1 if 100 falls within the computed interval. The `print p1` statement at the end of the loop (with a `--progressive` option) averages the values of `p1` over the 10000 replications. So, it produces the proportion of times that 100 lies within the computed interval. The scalar `p2` does the same for the slope.

The other scalars, `p3`, `p4`, `p5`, and `p6` compute statistics associated with tests of hypotheses. The Ha: for the intercept and slope are $\beta_1 > 100$ and $\beta_2 > 10$, respectively. If $\alpha = 0.05$ then these proportions should be close to 0.05 when Ho is true.

The scalars `p5` and `p6` measure the rejection of false hypotheses. Thus, `p5` measures the number of rejections of the hypothesis when $\beta_2 = 9$ and p6 measures the number of rejections of the hypothesis when $\beta_2 = 8$. This is related to the statistical power of the one-sided test and larger rejection proportions are better than smaller ones.

```
# Appendix 3.C
# Monte Carlo to measure coverage probabilities of confidence intervals
# and test size and power
nulldata 40
# Generate X
series x = (index>20) ? 20 : 10

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

loop 10000 --progressive --quiet
    series y = ys + randgen(z,0,50)
    ols y const x
    # 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 = $coeff(x) - critical(t,$df,.025)*$stderr(x)
    scalar c2R = $coeff(x) + critical(t,$df,.025)*$stderr(x)

    # Compute the coverage probabilities of the Confidence Intervals
    scalar p1 = (100>c1L && 100<c1R)
    scalar p2 = (10>c2L && 10<c2R)
    scalar p3 = (($coeff(const)-100)/$stderr(const))>critical(t,$df,.05)
    scalar p4 = (($coeff(x)-10)/$stderr(x))>critical(t,$df,.05)
    scalar p5 = (($coeff(x)-9)/$stderr(x))>critical(t,$df,.05)
    scalar p6 = (($coeff(x)-8)/$stderr(x))>critical(t,$df,.05)
    print p1 p2 p3 p4 p5 p6
endloop
```

The result from this script is shown below:

```
OLS estimates using the 40 observations 1-40
Statistics for 10000 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        99.6975        25.4357        24.7987        2.85096
        x        10.0176        1.61134        1.56841       0.180311

Statistics for 10000 repetitions
```

```
               mean        std. dev
p1         0.948800        0.220405
p2         0.947300        0.223434
p3        0.0502000        0.218357
p4        0.0552000        0.228370
p5         0.157500        0.364272
p6         0.345700        0.475596
```

The averages of p1 and p2 are expected to be close to 0.95, and they are. For instance for p2, 9473 of 10000 confidence intervals contained $\beta_2 = 10$. For p4 the true hypothesis Ho: $\beta_2 = 10$ was rejected in favor of $\beta_2 > 10$ in 5.52% of the samples (552 out of 10000).

When the null is false, as in Ho: $\beta_2 = 9$ vs $\beta_2 > 9$, then when $\beta_2 = 10$ as in the experiment, rejection is warranted. The mean of p5 measures the proportion of times the correct decision (rejection of Ho:) is made. In this case, for a 5% test, we rejected Ho: 1575/10000 times. The rejection rate increased to 3457/10000 when Ho: $\beta_2 = 8$.

### 3.7.2 Random Regressors

#### Appendix C3.4 of *POE5*

This simulation uses the same experimental design as used in section 2.8.4. Hence, new values of the regressors are generated at each of the 10000 samples. As in the preceding section scalars are computed to measure the properties of the confidence intervals and tests. The scalars p1 and p2 take the value 1 when the compound statement in parentheses is true. That means, for instance, p1=1 if 100 falls within the computed interval. The `print p1` statement at the end of the loop (with a `--progressive` option) averages the values of p1 over the 10000 replications. So, it produces the proportion of samples that 100 lies within the computed interval. The scalar p2 does the same for the slope.

The other scalars, p3, p4, p5, and p6 compute statistics associated with tests of hypotheses. The $H_a$ for the intercept and slope are $\beta_1 > 100$ and $\beta_2 > 10$, respectively. If $\alpha = 0.05$ then these proportions should be close to 0.05 when $H_o$ is true.

The scalars p5 and p6 measure the rejection of false hypotheses. Thus, p5 measures the number of rejections of the hypothesis when $\beta_2 = 9$ and p6 measures the number of rejections of the hypothesis when $\beta_2 = 8$. This is related to the statistical power of the one-sided test and larger rejection proportions are better than smaller ones.

```
# Appendix 3.C
# Monte Carlo to measure coverage probabilities of confidence intervals
# and test size and power
nulldata 40
loop 10000 --progressive --quiet
    series x = randgen(z,15,1.6)
    series y = 100+10*x + randgen(z,0,50)
    ols y const x
    scalar c1L = $coeff(const) - critical(t,$df,.025)*$stderr(const)
    scalar c1R = $coeff(const) + critical(t,$df,.025)*$stderr(const)
    scalar c2L = $coeff(x) - critical(t,$df,.025)*$stderr(x)
    scalar c2R = $coeff(x) + critical(t,$df,.025)*$stderr(x)

    # Compute the coverage probabilities of the Confidence Intervals
    scalar p1 = (100>c1L && 100<c1R)
    scalar p2 = (10>c2L && 10<c2R)
    scalar p3 = (($coeff(const)-100)/$stderr(const))>critical(t,$df,.05)
    scalar p4 = (($coeff(x)-10)/$stderr(x))>critical(t,$df,.05)
    scalar p5 = (($coeff(x)-9)/$stderr(x))>critical(t,$df,.05)
    scalar p6 = (($coeff(x)-8)/$stderr(x))>critical(t,$df,.05)
    print p1 p2 p3 p4 p5 p6
endloop
```

The result from this script is shown below:

```
OLS estimates using the 40 observations 1-40
Statistics for 10000 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       100.708        78.1931        76.6330        12.6580
        x       9.95236        5.17435        5.08156       0.839727

Statistics for 10000 repetitions

                  mean       std. dev
      p1      0.947300       0.223434
      p2      0.948000       0.222027
      p3     0.0533000       0.224631
      p4     0.0479000       0.213555
      p5     0.0693000       0.253964
      p6      0.102200       0.302911
```

Again, the averages of p1 and p2 are close to 0.95. For instance for p2, 9480 of 10000 confidence intervals contained $\beta_2 = 10$. For p4 the true hypothesis Ho: $\beta_2 = 10$ was rejected in favor of $\beta_2 > 10$ in 4.79% of the samples (479 out of 10000).

When the null is false, as in Ho: $\beta_2 = 9$ vs $\beta_2 > 9$, then when $\beta_2 = 10$ as in the experiment, rejection is warranted. The mean of p5 measures the proportion of times the correct decision (rejection of Ho:) is made. In this case, for a 5% test, we rejected Ho: 693/10000 times. The rejection rate increased to 1022/10000 when Ho: $\beta_2 = 8$. The size of the test was not affected by the presence of random regressors, but the power was diminished considerably.

## 3.8 Script

```
set echo off
set messages off
# Example 3.1
open "@workdir\data\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

function void t\_interval(scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %.2f confidence interval centered at %.2f is"\
           "(%.2f,\%.2f)\n", g, b, lb, ub
    end function

ols food_exp const income
t_interval($coeff(income),$stderr(income),$df,.95)

open "@workdir\data\food.gdt"
ols food_exp const income
scalar tratio1 = ($coeff(income) - 0)/ $stderr(income)

# Example 3.2
#One sided test (Ha:  b2 > zero)
scalar c2 = critical(t,$df,.05)
scalar p2 = pvalue(t,$df,tratio1)
printf "\nHa: b2>0  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio1, c2, p2

# Example 3.3
#One sided test (Ha:  b2>5.5)
scalar tratio2 = ($coeff(income) - 5.5)/ $stderr(income)
scalar c2 = critical(t,$df,.01)
scalar p2 = pvalue(t,$df,tratio2)
printf "\nHa: b2>5.5  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.01, p-value = %.3f\n", tratio2, c2, p2

# Example 3.4
#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 "\nHa: b2<15  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio3, c3, p3

# Example 3.5
#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 "\nHa: b2 not equal 7.5  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio4, c4, p4

#Confidence interval
t_interval($coeff(income),$stderr(income),$df,.95)

# Example 3.6
#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 "\nHa: b2 not equal 0  \n  t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", tratio5, c5, p5

# Example 3.7
#Linear Combinations of coefficients
open "@workdir\data\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 crit = critical(t,$df,.025)
scalar avg_food_20 = $coeff(const)+20*$coeff(income)

t_interval(avg_food_20,se,$df,.95)
printf "\nHa: Average weekly Foodexp|Income=2000 > 250 \n \
Average Expenditure = %3.2f, Standard Error = %.3f \n \
t = %.2f, critical value = %.2f \n \
alpha = 0.05, p-value = %.3f\n", avg_food_20, se, tval, crit, p
```

And for the repeated sampling exercise, the script is:

```
# Table 3.1
# repeated sampling exercise, the script is:
open "@workdir\data\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 b1_lb = b1 - critical(t,$df,.025)*s1
  scalar b1_ub = b1 + critical(t,$df,.025)*s1
  scalar b2_lb = b2 - critical(t,$df,.025)*s2
  scalar b2_ub = b2 + critical(t,$df,.025)*s2

  scalar sigma2 = $sigma^2
  store coeff.gdt b1 b2 s1 s2 b1_lb b1_ub b2_lb b2_ub sigma2
endloop

open @workdir\coeff.gdt
print  b1_lb b1_ub b2_lb b2_ub  --byobs
```

Monte Carlo to measure coverage probabilities of confidence intervals in section 3.4.

```
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 considered in this chapter. First, conditional predictions are generated using computations stored in memory after **gretl** estimates a model. Then, a commonly used measure of the quality of the linear fit provided by the regression is discussed. We then take a brief look at facilities within **gretl** for producing professional looking output to be used in reports and research.

Choosing a suitable functional form for a linear regression is important. Several choices are explored in this chapter. These include polynomial, linear-log, log-linear, and log-log specifications. We test residuals for normality. Normality of the model's errors is a useful property in that, when it exists, it improves the performance of least squares, tests and confidence intervals when sample sizes are small (finite).

Measures of the influence each observation has on your results are developed as well. The chapter ends with a couple of sections about conducting simulations and bootstrapping.

## 4.1 Prediction in the Food Expenditure Model

### Example 4.1 in *POE5*

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 store predicted values for all the observations or use `scalar` to save a computed 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 "@workdir\data\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. However, specifying the result as a `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% prediction 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 for the prediction 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) + $coeff(income)*20
scalar f=8013.2941+(8013.2941/40)+4.3818*(20-19.6047)^2
t_interval(yhat0,sqrt(f),$df,0.95))
```

This produces:

```
The 0.95 confidence interval centered at 287.609 is (104.132, 471.085)
```

[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`

### Example 4.1 revisited, using accessors

You may be wondering if accessors can be used to populate the inputs required for the computation of the forecast variance. 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.

```
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)
t_interval(yhat0,sqrt(f),$df,0.95))
```

This produces the same result as when some of the inputs used in the computation were hard coded:

```
The 0.95 confidence interval centered at 287.609 is (104.132, 471.085)
```

## 4.2 Coefficient of Determination

### Example 4.2 in *POE5*

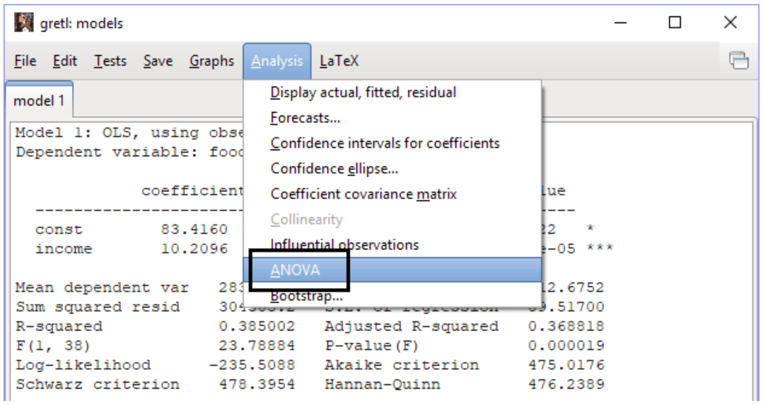
Some use regression analysis to “explain” variation in a dependent variable as a function of the independent variable(s). A summary statistic used for this purpose is the coefficient of determination, also known as $R^2$.

There are a number of ways to obtain $R^2$ in **gretl**. The simplest is to read it directly from **gretl**’s regression output. This is shown in Figure 4.3. After a regression, Gretl stores its $R^2$ computation in memory, which can be recalled using the accessor `$rsq`.

The most difficult way, 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.



```
Model 1: OLS, using obse
Dependent variable: fooc

                coefficient                              lue
  ------------------------                            ----
  const          83.4160                              22    *
  income         10.2096                              e-05 ***

Mean dependent var   283                              12.6752
Sum squared resid    304                              .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

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

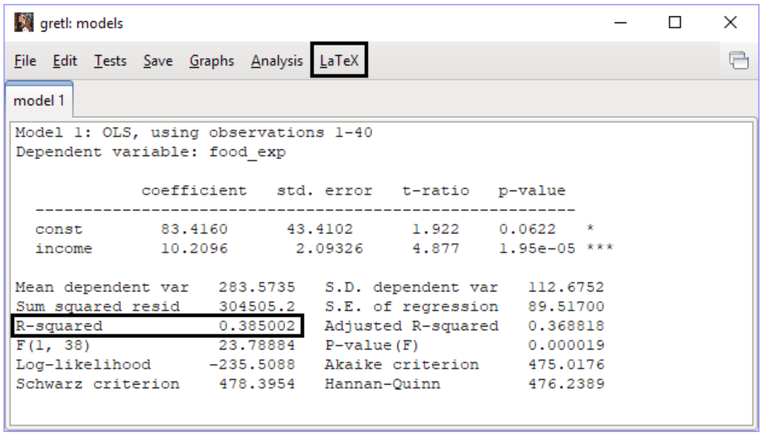
$$SST = SSR + SSE = 190627 + 304505 = 495132 \tag{4.4}$$

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. *POE5* 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 found below in section 4.7.3.



```
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
```

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

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 Figure 4.5 below. Then, **View>Correlation Matrix** will produce all the pairwise correlations between each variable 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, $0.6205^2 = .385$.

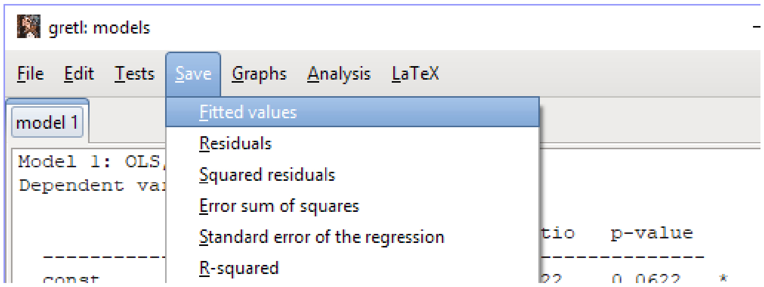


Figure 4.4: Using the pull-down menu in the Model window to add fitted values to your data set.

You can generate pairwise correlations from the console using

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

Again, it is not strictly necessary to use `scalar` or `genr` before `c1`. Gretl correectly identifies the variable type as a scalar and one can safely omit the `scalar` declaration command. In longer scripts, however, it's good practice to declare variable types in **gretl** so that error messages are thrown when the result doesn't match what you expect. This won't be discussed any further in the remainder of this manual where we will always identify new computations by their expected variable types.

## 4.3 Reporting Results

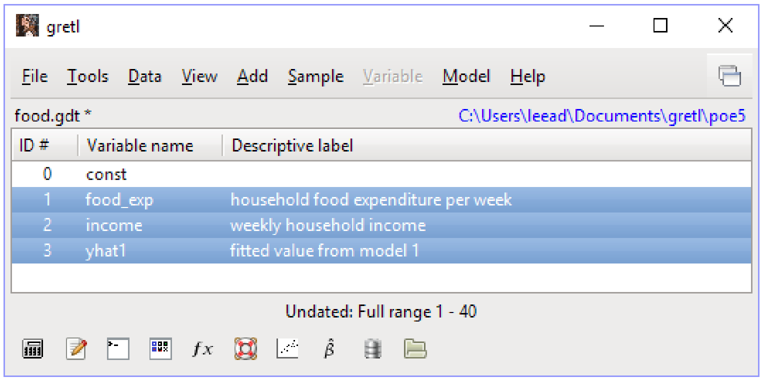
### Example 4.3 in *POE5*

Gretl includes facilities to aid in the production of good looking output. For instance, results from the models window can be copied and saved in several formats, including RTF(MS Word) and LaTeX.

LaTeX, 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, reports, presentation slides and research papers. 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 which takes some time to learn its intricacies and to appreciate its nuances. 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, many of the output windows **gretl** provide the ability to copy, save, or print properly formatted LaTeX tables and equations. For users of LaTeX, this makes generating regression

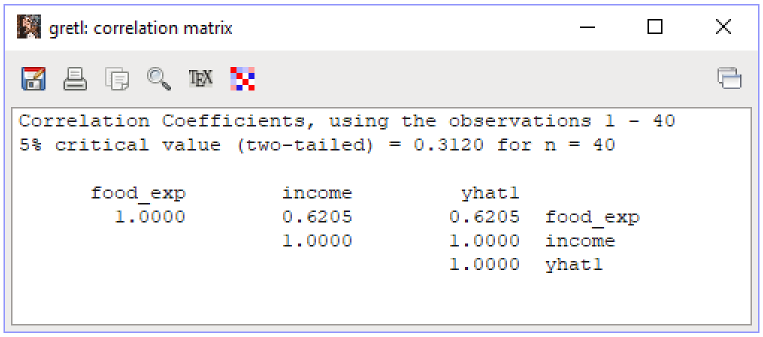


food.gdt * C:\Users\leead\Documents\gretl\poe5

| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | food_exp | household food expenditure per week |
| 2 | income | weekly household income |
| 3 | yhat1 | fitted value from model 1 |

Undated: Full range 1 - 40

Figure 4.5: Hold the control key and click on *food_exp*, *income*, and $\widehat{food\_exp} = yhat1$ from the food expenditure regression to select them.

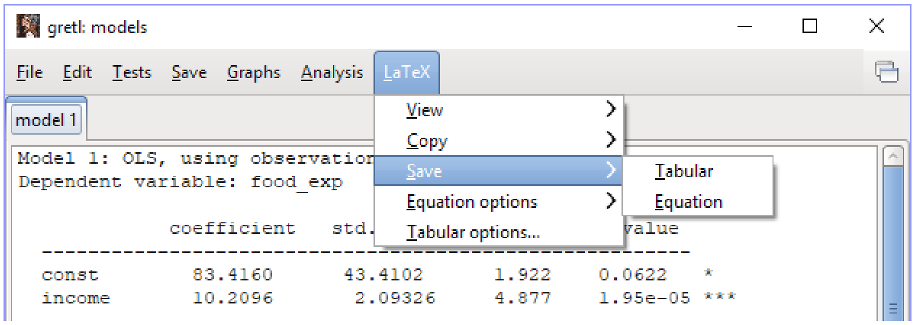


```
Correlation Coefficients, using the observations 1 - 40
5% critical value (two-tailed) = 0.3120 for n = 40

      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.

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, or are looking for a reason to learn 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 4.7. You can view, copy, or save the regression output in either tabular form or in equation form. You can choose to display 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.



```
Model 1: OLS, using observation
Dependent variable: food_exp

             coefficient    std.                 value
  ------------------------------------------------------
  const        83.4160      43.4102      1.922    0.0622   *
  income       10.2096       2.09326     4.877    1.95e-05 ***
```

Figure 4.7: Several options for defining the output of LaTeX are available. Highlighted here, you can either save an estimated model in equation or tabular form.

Another useful trick allows one to change the number of digits shown in **gretl** model windows. In a models window, right-click and a menu of options is revealed (Figure 4.8). At the end of the list is **Digits**. Click on this and select the number of digits to display in the output. You can also save, copy, or print in various formats.

**Gnuplot** (pronounced "new plot") is widely used to produce professional publication quality graphics. Gretl comes packaged with **gnuplot** and provides an interface that makes getting decent

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

looking graphs easy, certainly easier that doing them strictly in **gnuplot**. Some of this functionality will be explored below.

## 4.4 Choosing a Functional Form

### Example 4.4 in *POE5*

There is no reason to think that the relationship between *food_exp* and *income* is linear. 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. A linear model implies that rich and poor spend the same amount of an additional dollar of income. As seen in Chapter 3, nonlinearities can be modeled by transforming either the dependent or independent variable or both. 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 more appropriate. Fortunately, a simple transformation of the variables ($x$, $y$, or both)

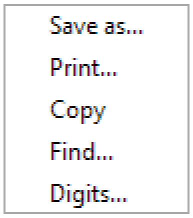


Figure 4.8: Right-click in the models window reveals this set of options.

yields a model that is linear in the parameters (but not necessarily in the variables).

The functional form you choose should be consistent with how the data are actually being generated. If you choose a functional form that when properly parameterized cannot generate your data, then your model is **misspecified**. The estimated model may, at best, not be 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 4.9 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 if

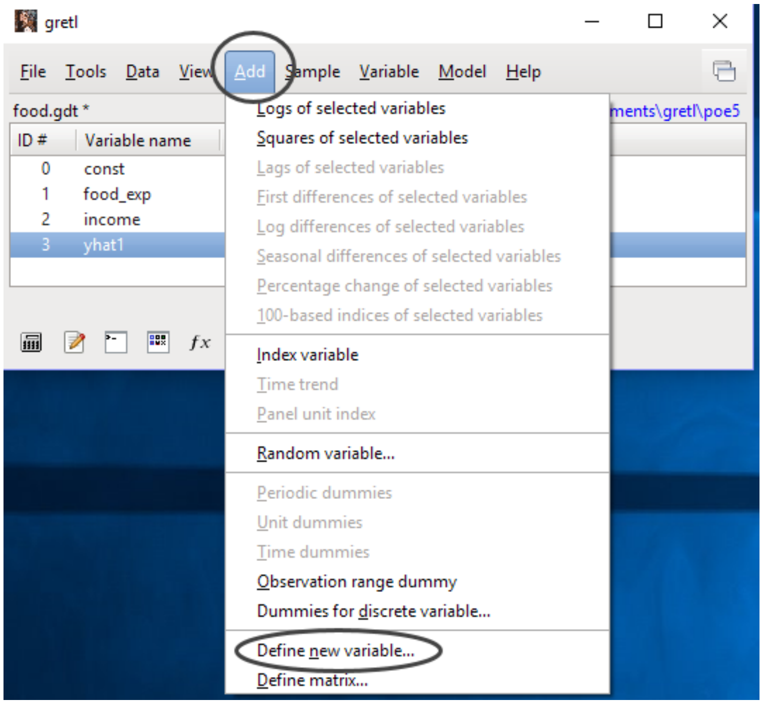

gretl
File
Tools
Data
View
Add
Sample
Variable
Model
Help
food.gdt *
ments\gretl\poe5
ID #
Variable name
0
const
1
food_exp
2
income
3
yhat1
Logs of selected variables
Squares of selected variables
Lags of selected variables
First differences of selected variables
Log differences of selected variables
Seasonal differences of selected variables
Percentage change of selected variables
100-based indices of selected variables
Index variable
Time trend
Panel unit index
Random variable...
Periodic dummies
Unit dummies
Time dummies
Observation range dummy
Dummies for discrete variable...
Define new variable...
Define matrix...

Figure 4.9: The Add pull-down menu is used to add new variables to **gretl** .

the dataset structure has been defined as time series or panel observations. The available options include being able to add the natural logarithm or the squared values of any highlighted variable to the dataset.

The next to last option **Define new variable** can be selected to perform more complicated transformations. This dialog uses the `series` command and which can use the large number of built-in functions to transform variables. A few of the possibilities include taking a square root (`sqrt`), sine (`sin`), cosine (`cos`), absolute value (`abs`), exponential (`exp`), minimum (`min`), maximum (`max`), and so on. Later in the book, I'll discuss changing the dataset's structure to enable time-series or panel transformations.

One can also create a matrix or scalar using the last option, **Define matrix**. This choice brings up a useful dialog that allows you to build a matrix from a series, a formula, or numerically.

### 4.4.1 Linear-Log Specification

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

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

Taking the logarithm of income and estimating the model

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

The `logs` command can be used to add the natural logs of several variables of selected variables to the dataset. The `logs` function to create ln(*income*) is:

```
logs income
```

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

Estimation of the model yields

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

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

(standard errors in parentheses)

In Figure 4.6 of *POE5* the authors plot *food_exp* and $\widehat{food\_exp}$ against *income*. A positive (nonlinear) relationship between the two is expected since the model was estimated using the natural logarithm of income. To produce this plot, estimate the regression to open the **models window**. Add the predicted values of from the regression to the dataset using **Save>Fitted values** from the models window's pull-down menu. Name the fitted value, `yhat2` and click **OK**.

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 4.10

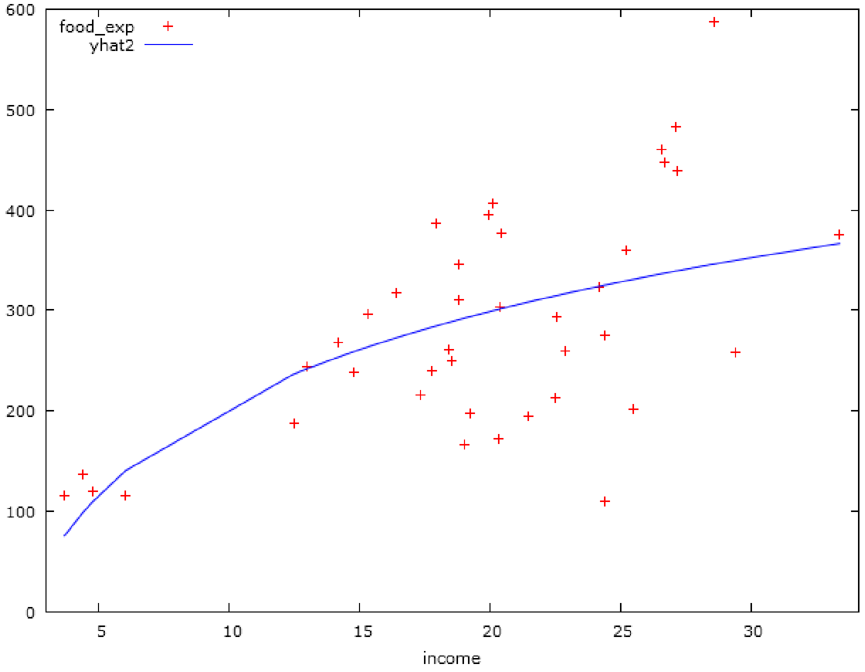


Figure 4.10: Graphing the linear-log model

A simpler approach is to open a console or a new script window and use the following commands:

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

which adds a **gnuplot** command to plot the two series against income. The first line estimates

[3] Click on each variable while holding down the CTRL key

[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.

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`, as the second.

When executed from the console, the graph summoned in line 3 is opened in a window on your screen. However, when executing these commands using a script, the graph is written to a file on your computer. To control where output is sent, use the `--output=` option in the gnuplot command. The graph can be sent to the screen or saved in a file of a given name in the desired format. You can also send the graph to a **session** by prefixing the command with a name and the assignment operator `<-`. Examples of these choices are:

```
gnuplot yhat2 food_exp income --output=display   # output to screen
gnuplot yhat2 food_exp income --output=graph.pdf # output to a pdf
g1 <- gnuplot yhat2 food_exp income              # send graph to session
```

### 4.4.2 Residual Plots

Misspecifying the model's functional form can lead to serious problems when making decisions based on the results. There are a number of statistical tests one can use to diagnose specification problems, but researchers often start by examining residual plots for evidence of any obvious misspecification.

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 shown in Figure 4.11 below. 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.

#### Linear-Log Model

The plot in Figure 4.12 is of the least squares residuals from the **linear-log** food expenditure model. These do not appear to be strictly random. Rather, they appear to be heteroskedastic, which means that for some levels of income, food expenditure varies more than for others (more variance for high incomes). The script to produce this is:

```
open "@workdir\data\food.gdt"
logs food_exp
ols food_exp const l_income
series uhat = $uhat
gnuplot uhat l_income --output=display
```

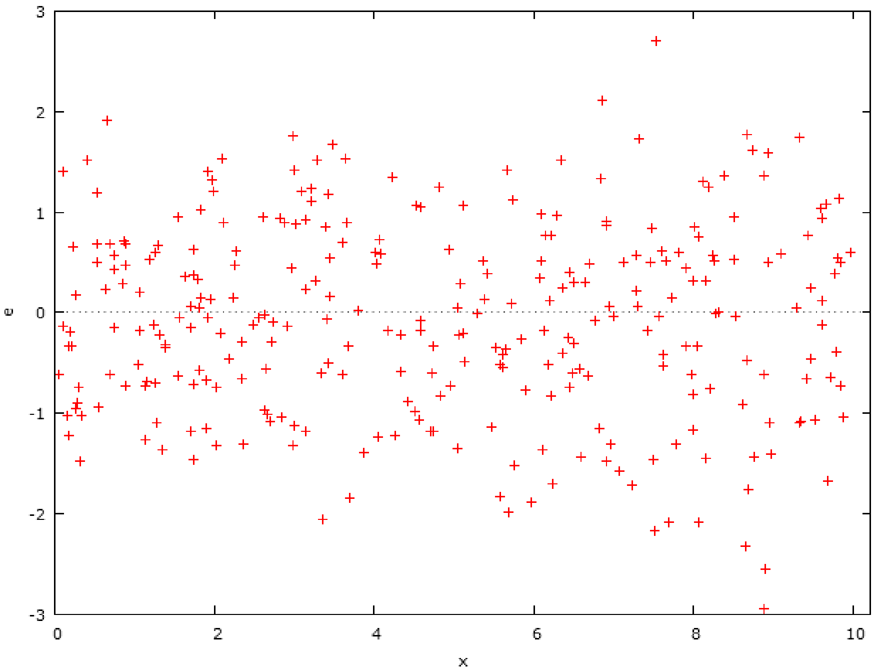


Figure 4.11: Randomly distributed residuals

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.

### Log-Linear Model

The next plot is of the least squares residuals from the **log-linear** food expenditure model (Figure 4.13). These may be mildly heteroskedastic, less so than in the linear-log model. The script to produce this is:

```
1 open "@workdir\data\food.gdt"
2 logs food_exp
3 ols l_food_exp const income
4 series uhat = $uhat
5 gnuplot uhat income --output=display
```

Notice that in line 4 the accessor `$uhat` has been used to store the residuals into a new variable. Here they are assigned to the series `ehat`. Then, they can be plotted using **gnuplot**.

Now consider residuals of a misspecified model shown in Figure 4.14. The errors are supposed

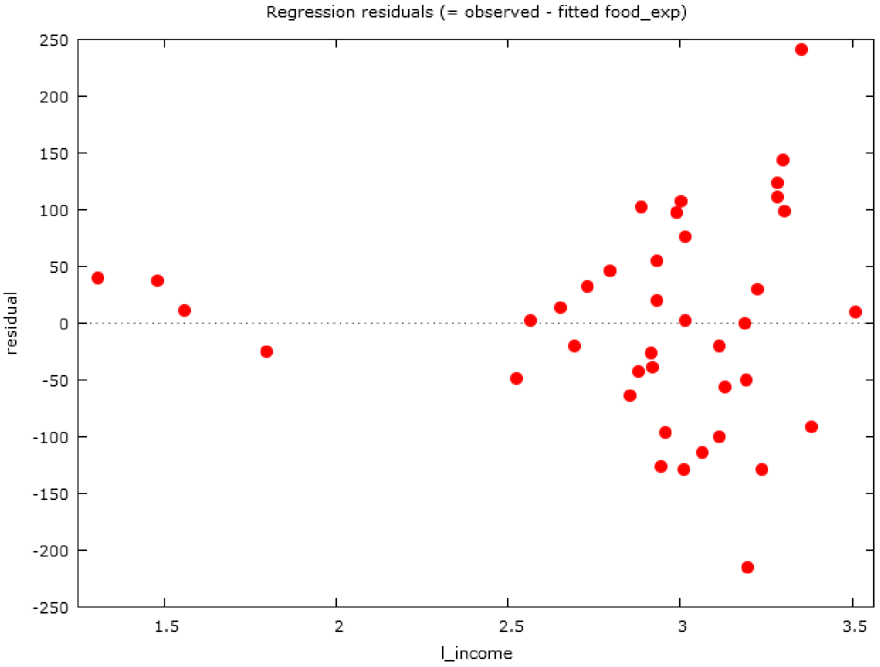


Figure 4.12: Heteroskedastic residuals from the linear-log model of food expenditures.

to look like a random scatter around zero. There are clearly parabolic and the functional form of the model is NOT correctly specified.

### 4.4.3 Testing for Normality

*POE5* 5 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 the model using least squares and save the residuals to the dataset.

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 them into a new series called `uhat1`.[5] You could use the point-and-click method to add the residuals to the data set. This is accomplished from the **models** window. Simply choose **Save>Residuals**

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

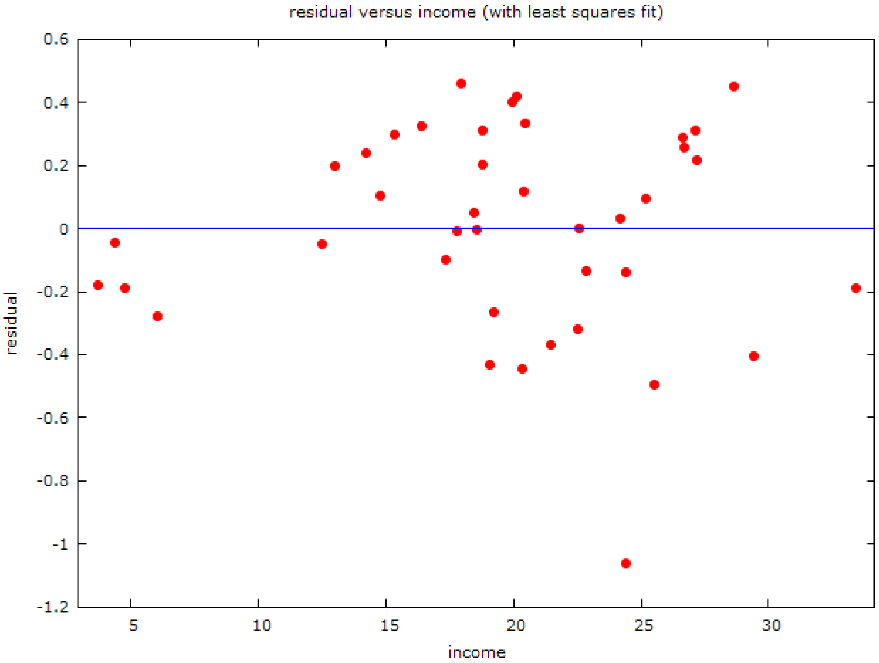


Figure 4.13: After some editing, residuals from the log-linear food expenditure model.

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 4.15. 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, the computation is

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

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

Normally distributed random variables have no skewness nor 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.3 **hansl** is used to compute skewness and excess kurtosis and could be used 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      # save the residuals
normtest uhat1 --jbera    # compute Jarque-Bera Test
normtest uhat1 --all      # show all normality tests
```

This yields a value of Jarque-Bera test = 0.0633401, with $p$-value 0.968826, which is exactly what

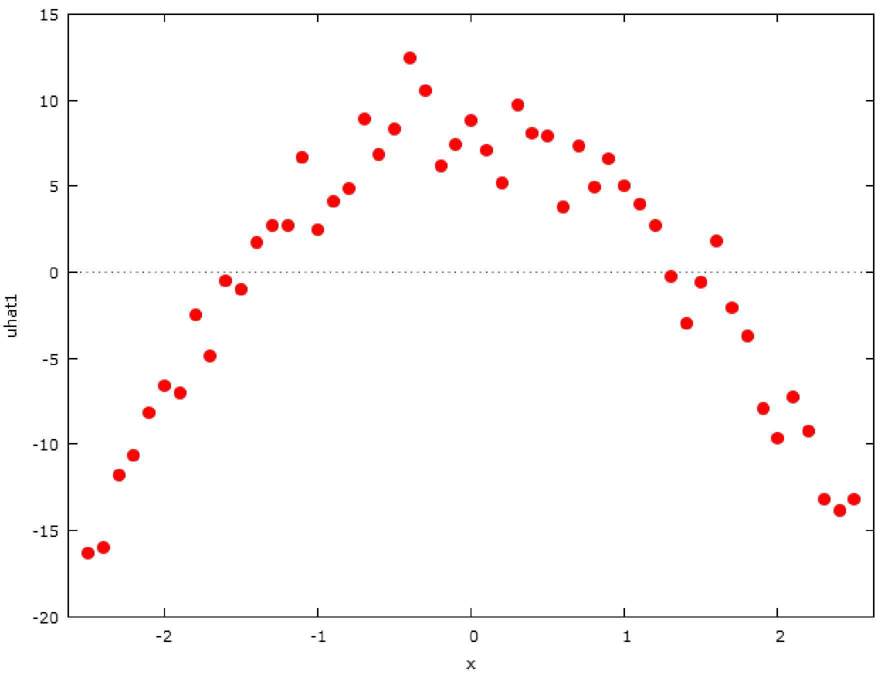


Figure 4.14: Correlated residuals from estimating a quadratic relationship using a line.

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 $\chi^2$ 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 4.16. Obviously, one of the advantages of using normtest is 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.

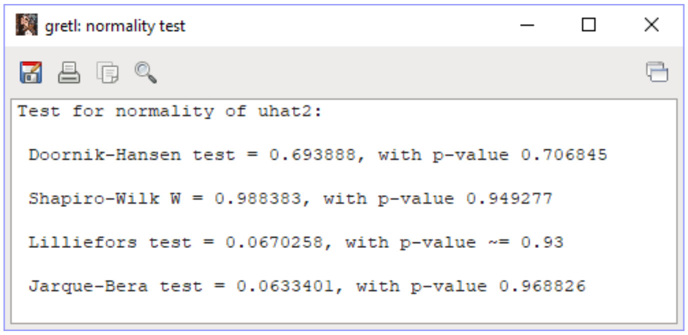
```
ols food_exp const income
modtest --normality
```

The modtest command is 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.
```

```
            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 4.15: The summary statistics for the least squares residuals.



```
Test for normality of uhat2:

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

 Shapiro-Wilk W = 0.988383, with p-value 0.949277

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

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

Figure 4.16: Using `normtest` *residual* `--all` tests the variable *residual* for normality after running a linear regression.

```
          < -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 as shown in Figure 4.17. You can also reach this graph by highlighting the variable you wish to test in the main window, right-clicking, and selecting **frequency distribution** from the pull-down menu. That opens the frequency distribution dialog box that plots the series and has an option that performs the normality test.

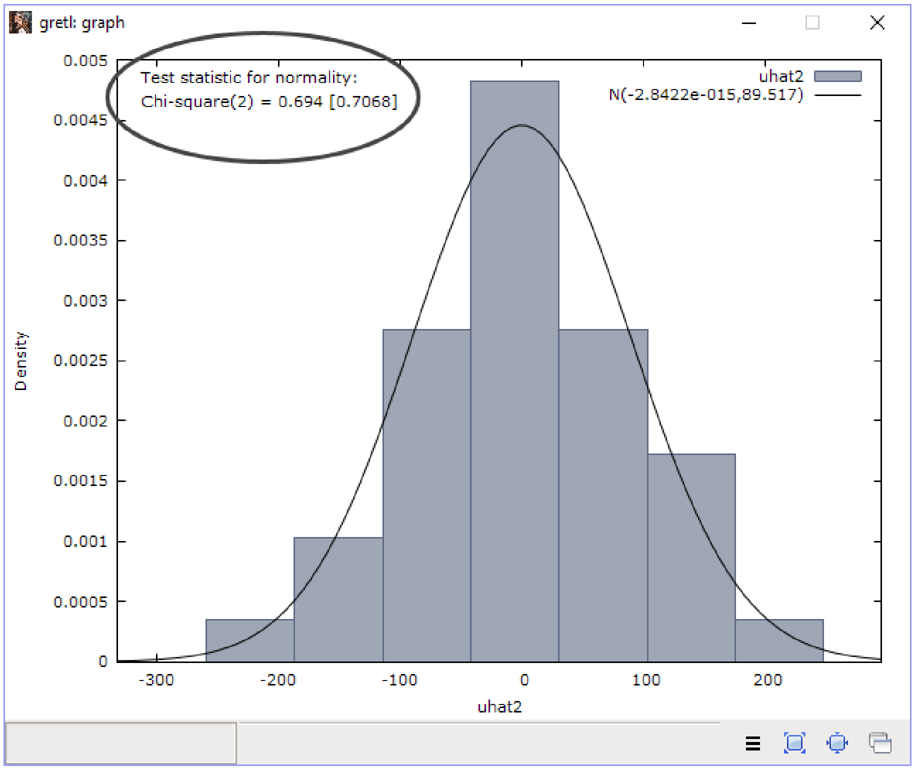


Figure 4.17: From the models window, use **Tests>Normality of residual** from the pull-down menu. This produces this histogram and reports the Doornik-Hansen test from `modtest --normality`.

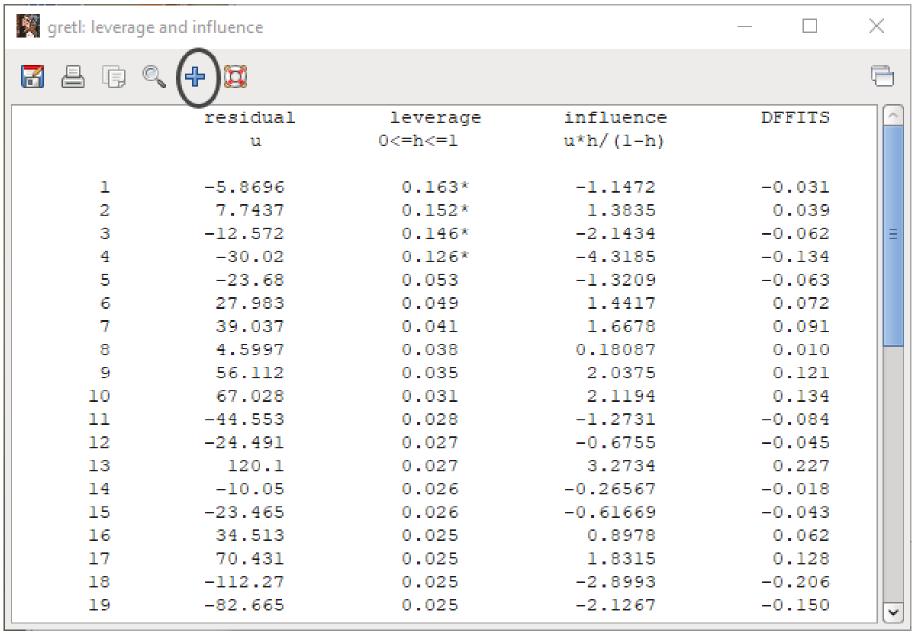
## 4.5 Influential Observations

### Example 4.7 in *POE5*

There are a number of statistics used to help identify influential observations in the data. An influential observation is one whose omission from the data has a large impact on the results. The statistics considered include leverage, studentized residuals, sensitivity of a coefficient estimate to omission of the $t^{th}$ observation (DFBETAs), and the sensitivity of predictions to the omission of the $t^{th}$ observation (DFFITS).

Gretl includes a set of diagnostics that are available from either the GUI or as a **hansl** command. The **gretl** `leverage` command produces some, but not all of the measures considered here. Consequently, we will be writing programs to compute the missing statistics.

To see what **gretl** can do easily, estimate a linear regression model and open its models window (Figure 2.9). From the menu bar choose **Analysis>Influential observations** from the menu bar. This produces two windows of output. The first window, **leverage and influence**, is shown in Figure 4.18. It lists the estimated residual, leverage, influence, and DFFITS for each observation in the sample. Clicking on the plus sign on the menu bar enables you to save any of the last three to the current dataset. A high leverage point is distant from $\bar{x}$. It has the potential to be influential if it is also distant from the regression line compared to similar points. An influential point exerts a relatively large influence on the intercept and slope of the regression.



```
                residual         leverage        influence         DFFITS
                   u             0<=h<=1         u*h/(1-h)

       1        -5.8696           0.163*          -1.1472          -0.031
       2         7.7437           0.152*           1.3835           0.039
       3        -12.572           0.146*          -2.1434          -0.062
       4         -30.02           0.126*          -4.3185          -0.134
       5         -23.68           0.053           -1.3209          -0.063
       6         27.983           0.049            1.4417           0.072
       7         39.037           0.041            1.6678           0.091
       8         4.5997           0.038           0.18087           0.010
       9         56.112           0.035            2.0375           0.121
      10         67.028           0.031            2.1194           0.134
      11        -44.553           0.028           -1.2731          -0.084
      12        -24.491           0.027           -0.6755          -0.045
      13          120.1           0.027            3.2734           0.227
      14         -10.05           0.026          -0.26567          -0.018
      15        -23.465           0.026          -0.61669          -0.043
      16         34.513           0.025            0.8978           0.062
      17         70.431           0.025            1.8315           0.128
      18        -112.27           0.025           -2.8993          -0.206
      19        -82.665           0.025           -2.1267          -0.150
```

Figure 4.18: From the models window, choose **Analysis>Influential observations** from the menu bar to produce these statistics that can be added to your data.

To be influential, an observation will generally have a large estimated residual and will also have a high leverage. Thus, the first two columns contain these two components. The average value of leverage is $k/n$. If a leverage statistic is significantly larger than this number, **gretl** places a star * by it in the list.

Influence (see Exercise 2.26 in Davidson and MacKinnon (2004)) measures the change in the $t^{th}$ residual when observation $t$ is omitted from the model. If the residual changes by a large amount when the observation is omitted then that is evidence that it is exerting a large influence on the estimated slopes. The last column contains DFFits, which measures how much the predicted value of the dependent variable changes when an observation is omitted. Below, we will discuss how each of these is actually computed.

Missing from the possible statistics listed by **gretl** are the studentized residuals, DFBETAs, and the delete one variance estimate that is used in the computation of these. In the next section, some of the details on how these statistics can be computed are presented. Also, a short script is given to produce each of these statistics.

### 4.5.1 Leverage, Influence, and DFFits

In this section, we derive a handful of useful diagnostics for detecting the presence of influential observations. The statistics we consider are based on a comparison of two regressions. One regression uses all of the observations and the other omits one observation. If the residual, prediction, or slope changes substantially when an observation is excluded then we conclude that the observation in question is influential.

Consider the following linear model.

$$y_i = \beta_1 + x_i\beta_2 + u_i \tag{4.8}$$

When equation (4.8) is estimated using all observations, the least squares estimators are $b_1$ and $b_2$. The estimated variance is $\hat{\sigma}^2$. When estimated using least squares with the $t^{th}$ observation omitted from the sample the estimates are $b_1^{(t)}$ and $b_2^{(t)}$ and the estimated variance is $\hat{\sigma}^2(t)$. We'll refer to $\hat{\sigma}^2(t)$ as the delete-one variance. These must be obtained in order to compute some of the other influence statistics of interest.

A trick that can be used to drop the $t^{th}$ observation is to create a variable that has a zero in every observation except for the $t^{th}$. We call this variable $e_t$.

Now the model can be written:

$$y_i = \beta_1 + x_i\beta_2 + e_t\alpha + u_i \tag{4.9}$$

where $i = 1, 2, \cdots, n$. Including this variable in the model and estimating the parameter by least squares yields this produces $b_1^{(t)}$ and $b_2^{(t)}$.

Useful measures of the influence the $t^{th}$ observation has on the estimation of the model's parameters is $b_1 - b_1^{(t)}$ and $b_2 - b_2^{(t)}$. Properly scaled by its standard deviation, this becomes the basis of the DFBETA statistics.

There are a few other statistics that require the computation of these. The easiest way to do this use matrix algebra. If that doesn't thrill you, then feel free to skip this section.

### 4.5.2 The Hat and Residual Maker matrices

Linear regression using least squares is an exercise in geometry. Least squares finds the shortest distance between the dependent variable and the space defined by the regressors. In Euclidean geometry the shortest route is orthogonally from the point y to the space defined by $x_1, x_2, \cdots, x_k$. Expressing the linear model in matrix form

$$y = X\beta + e \tag{4.10}$$

where $y$ is an $n \times 1$ vector containing all $n$ observations on the dependent variable, $X$ is $n \times k$ and each row contains a observation on each of the explanatory variables, the $k \times 1$ vector $\beta$ contains the intercept and slopes to be estimated, and $e$ is $n \times 1$ containing the residuals. The $rank(X) = k \leq n$.

The least squares estimator is

$$b = (X^T X)^{-1} X^T y.$$

Note,

$$Xb = \hat{y} = X((X^T X)^{-1} X^T y = Hy$$

The matrix $H$ is called the Hat matrix because it creates least squares predictions for any variable that it multiples, in this case, $y$. $H$ is the orthogonal projection onto the space defined by X. Usually it is denoted $P_x$ and I'll follow that convention going forward.

The residual maker creates the least squares residuals.

$$\hat{e} = y - \hat{y} = y - Xb = y - P_x y = (I_n - P_x)y = M_x y$$

The diagonal elements of the Hat matrix, $P_x$ are $h_i$, $i = 1, 2, \cdots, n$. The $h_i$ is referred to as **leverage** of observation $i$. It is $0 < h_i < 1$. The variance of the $i^{th}$ least squares residual, $\hat{e}_i = \sigma^2(1 - h_i)$. This implies that the least squares residual is smaller than the actual variance of $e_i$. It also implies that the least squares residuals depend on $i$ and are heteroskedastic.

The most straightforward way to compute the leverage measure in **gretl** is using these matrices. Below is a simple function that produces these:

```
1 function series h_t (list xvars)
2     matrix X = { xvars }
3     matrix Px = X*inv(X'X)*X'
4     matrix h_t = diag(Px)
5     series hats = h_t
6     return hats
7 end function
```

This function is quite simple. It takes a `list` of variables as arguments. In line 2 these are converted to a `matrix`, line 3 computes the hat matrix, line 4 takes the diagonal elements of $P_x$, line 5 puts those into a `series` and `return` sends the series out of the function when called.

To use the function, create a `list` for the independent variables and use:

```
list xlist = const income
series lev_t = h_t(xvars)
```

This puts a variable called `lev_t` into your dataset that contains the leverage statistics.

### Delete one variance computation

Another building block that must be computed is the delete-one variance, $\hat{\sigma}^2(t)$. There are a number of approaches one could take to compute these. I have chosen one that uses a few matrices and that relies on internal **gretl** functions for the most part. Inevitably, some matrices are created to facilitate variable collection and creation.

The function created to compute and collect the delete-one variances is:

```
 1 function series delete_1_variance(series y, list xvars)
 2     matrix sig = zeros($nobs,1)
 3     loop i=1..$nobs
 4         matrix e_t = zeros($nobs,1)
 5         matrix e_t[i,1]=1
 6         series et = e_t
 7         ols y xvars et
 8         matrix sig[i,1]=$sigma^2
 9     endloop
10     series sig_t = sig
11     return sig_t
12 end function
```

The function is called `delete_1_variance` and returns a `series`. It takes two arguments: a `series` for the dependent variable and a `list` for the regression's independent variables. In line 2 a matrix of `zeros` is created that will hold the variances as they are computed within the loop. It loops over the number of observations started in line 3. In line 4 another matrix of zeros is created at each new iteration that becomes the variable that will be added to the model, i.e., $e_t$. In the next line a 1 is placed in the $i^{th}$ row of the zero vector and then converted to a series in line 6. In line 7 a regression is estimated that augments the model with the created regressor that will omit the $i^{th}$ observation. The accessor `$sigma` is used to compute the variance. The loop ends and the matrix `sig` is converted to a series and returned.

To use the function, create a `list` for the independent variables and use:

```
list xlist = const income
series sig_t = delete_1_variance(food_exp, xlist)
```

This puts a variable called `sig_t` into your dataset that contains the delete-one variances.

These functions are used to compute studentized residuals and the DFFITS (See Table 4.3): The $h_t$ are the diagonal elements of the hat matrix, $\hat{\sigma}(t)$ is the square root of the $t^{th}$ delete-one variance, and $\hat{e}_t$ is the $t^{th}$ least squares residual using the entire sample.

| Statistic | Formula |
|---|---|
| Leverage | $h_t$=diag$(P_x)$ |
| Studentized Residual | $\hat{e}_t^{stu} = \hat{e}_t/(\hat{\sigma}(t)\sqrt{1-h_t})$ |
| DFFITS | $\hat{e}_t^{stu}\sqrt{h_t/(1-h_t)}$ |

Table 4.3: Influence Diagnostics

Once $\hat{e}_t$, $h_t$, $\hat{\sigma}(t)$ are in the data, , $\hat{e}_t^{stu}$ and DFFITS can be computed as series in turn. The complete set of influence statistics are generated using:

```
list xvars = const income
ols food_exp xvars

series ehat = $uhat
series lev_t = h_t(xvars)
series sig_t = delete_1_variance(food_exp, xvars)
series stu_res = ehat/sqrt(sig_t*(1-lev_t))
series DFFits=stu_res*sqrt(lev_t/(1-lev_t))
```

and added to your dataset. The advantage of having these in the data is that they can be further manipulated to identify minima, maxima, as plots, etc. Notice that these match the ones computed using the leverage command shown in Figure 4.18 above.

### 4.5.3 DFBETA

The leverage, delete-one variance, and DFFITS computations are straightforward. The computation of DFBETA is less so. This comes from the fact that there are several different ways to express this measure of influence on the estimation of the parameters. In principle this is what you want to estimate for each coefficient at each observation:

$$\text{DFBETA}_{j,t} = \frac{b_j - b_j^{(t)}}{\sqrt{var(b_j^{(t)})}}$$

One representation of this from *POE5* is

$$\text{DFBETA}_{j,t} = \frac{b_j - b_j^{(t)}}{\hat{\sigma}(t)/\hat{\sigma} \times se(b_j)}$$

Stata use another calculation that uses the studentized residual and the outcomes of auxiliary regressions:

$$\text{DFBETA}_{j,t} = \frac{\hat{e}_t^{stu}\hat{u}_{j,t}/(1-h_t)}{\sqrt{\sum_{t=1}^{n}\hat{u}_{j,t}^2}}$$

The statistic $\hat{u}_j$ is a least squares residual of $x_j$ regressed onto all of the other independent variables in the model. So for instance, $j = 2$ and $k = 4$, the regression is

$$x_2 = \alpha_1 + \alpha_3 x_3 + \alpha_4 x_4 + res.$$

Then $\hat{u}_{2t} = \hat{\alpha}_1 + \hat{\alpha}_3 x_{3t} + \hat{\alpha}_4 x_{4t}$. The algebraic form is harder than the computation.

```
 1 list x1 = const income
 2 scalar k = nelem(x1)
 3 matrix results = zeros(k,1)
 4 loop i=1..k --quiet
 5     list y1 = x1[1]
 6     list y2 = x1[2:k]
 7     ols y1 y2
 8     series dfb$i=stu_res*$uhat/sqrt($ess*(1-lev_t))
 9     list x1 = y2 y1
10 endloop
```

This script requires that you have the studentized residuals in the data as well as the leverages by observation.[6] The logic of the program deserves a defense (yes, it is ugly). The `list` of regressors in the model is populated and the number of elements it contains counted using `nelem()`. A $k \times 1$ matrix of zeros is initialized before the `loop` starts in line 4. The loop iterates over each of the $k$ variables in the list. The variables in the list are divided into two sets. The first set, `y1`, contains only the first variable from the list and the other set, `y2`, contains the remaining ones. The `y1` variable is regressed onto the remaining ones contained in `y2`.

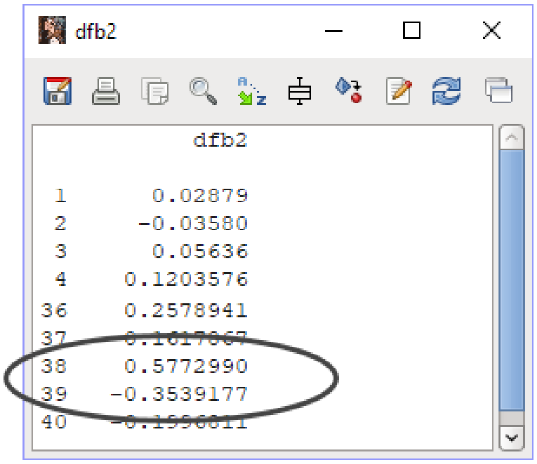
After the regression, use the accessors (`$uhat`) for the residuals, $\hat{u}_{jt}$, and `$ess` for the sum of their squares ($\sum_{t=1}^{n} \hat{u}_{j,t}^2$). These are used to compute the series for `dfb`.

Finally, we rearrange the list by moving the first variable, `y1`, to the end of the list.

| Iteration | Dependent | Independent | | |
|---|---|---|---|---|
| | y1 | y2 | | |
| i=1 | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
| i=2 | $x_2$ | $x_3$ | $x_4$ | $x_1$ |
| i=3 | $x_3$ | $x_4$ | $x_1$ | $x_2$ |
| i=4 | $x_4$ | $x_1$ | $x_2$ | $x_3$ |

The loop increments, the first variable is now the second regressor and the first regressor has moved to the end of the list. n the revised list will be the second variable in the original list and becomes `y1`; `y2` contains all of the others. Let's just say it works. When the routine has finished you'll have two new variables in the data: `dfb1` and `dfb2`. Some results for the DFBETA for income are shown in Figure 4.19. Notice that for observations 38 and 39, the result matches those shown in *POE5* Example 4.7. This also matches the result from Stata 15.1.

[6] Putting all of this together in a **gretl** bundle would be a fine idea.



```
          dfb2

 1      0.02879
 2     -0.03580
 3      0.05636
 4    0.1203576
36    0.2578941
37    0.1617067
38    0.5772990
39   -0.3539177
40   -0.1996811
```

Figure 4.19: Output from the DFBETA(income) in the food expenditure model.

## 4.6 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 *POE5* 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

$$y = \beta_1 + \beta_2 x^2$$
$$dy/dx = 2\beta_2 x$$

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

The cubic model and its slope are

$$y = \beta_1 + \beta_2 x^3$$
$$dy/dx = 3\beta_2 x^2$$

Since $x$ is squared in the slope, the algebraic sign of $\beta_2$ determines whether the slope is positive or negative. Both of these models are considered using examples below.

### 4.6.1 Wheat Yield

#### Example 4.8 in *POE5*

Figure 4.23 contains a plot of 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 *POE5* are 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 loads the data file, estimates the model using least squares, and generates a graph of the actual and fitted values of yield (greenough) from the model.

```
open "@workdir\data\wa-wheat.gdt"
ols greenough const time
gnuplot greenough time --output=display
```

The resulting plot appears below in Figure 4.20. Right-clicking on the graph brings up a menu of choices shown in Figure 4.21. Choose **Edit** and the **plot controls** dialog box appears as shown in Figure 4.22. 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 'Time.' After a little editing, the new graph (Figure 4.23) looks even better.

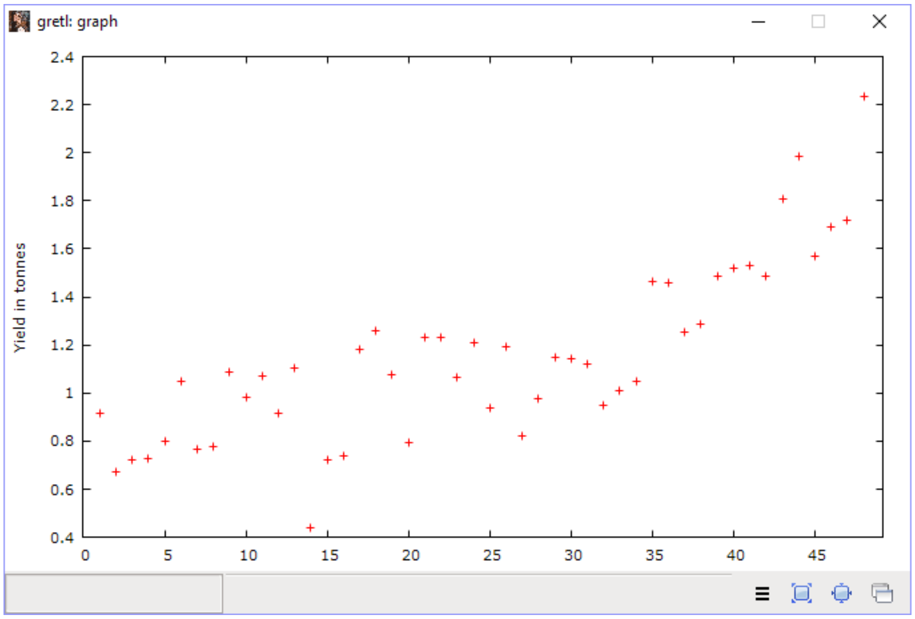


Figure 4.20: Plots wheat yield in Greenough Shire over time.

The simple **gnuplot** command works well enough. Adding information from the console or a

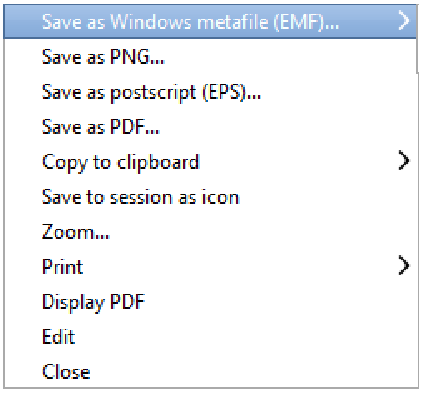


Figure 4.21: This fly-out menu is initiated by right-clicking on a **gretl** graph. To save it for further manipulation, choose **save to session as icon**.

script is easy to do as well. I added a description and a label to be used in the graph using the `-d` and `-n` switches for `setinfo`.[7] The commands are

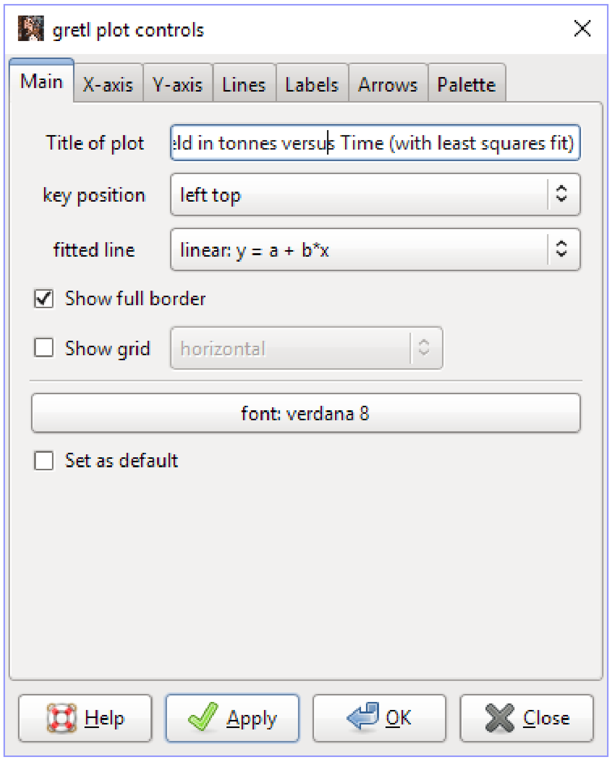
```
1 setinfo greenough -d "Wheat yield in tonnes" -n "Yield in tonnes"
2 setinfo time -d "Time" -n "Time"
3 g1 <- gnuplot greenough time --fit=linear --output=display
```

The command in line 3 is the first use of the assignment feature that is can be used with some **gretl** commands. The expression `g1 <- gnuplot greenough time` takes the graph produced by **gnuplot** and places it into a session, which is discussed in section (1.3.3), as an icon labeled `g1`. Once this is in your session, it can be displayed and edited in the usual way using **gretl**, or it can be edited using **gnuplot** commands. It also enables the `graphpg` commands to be used in a script or from the console.

Two options to `gnuplot` are used. The first option (`--fit=linear`) tells **gnuplot** to plot a least squares regression line that is fitted using the data. This option also adds the ability to apply different types of fit from the graph controls (see Figure 4.22). The second option plots the output to the screen. Once the graph window is displayed, it can be added to a session as an icon. A right-click on the icon in the session window allows you to edit the **gnuplot** commands and to make professional looking publication quality graphics using **gnuplot**. The icons can be dragged to the Graph page icon to combine several graphs onto a single page. This will be explored further below.

To make the graph look like Figure 4.24 some further manipulation was done using the plot controls.

[7] `-d` is an abbreviation of `--description=` and `-n` of `--graph-name=`.



Title of plot: ld in tonnes versus Time (with least squares fit)

key position: left top

fitted line: linear: y = a + b*x

☑ Show full border

□ Show grid: horizontal

font: verdana 8

□ Set as default

Figure 4.22: 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. Note: the fitted line box shown here only appears if you have used the `--fit=` option.

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

### 4.6.2 Combining graphs

As mentioned above, graphs (or plots) can be done from the session window or using a script. To combine graphs from a session, save your graphs to a session as icons and then drag them to the graph page icon. Opening this icon reveals the combined graph. The beauty of this method is that each graph can be edited and saved before adding it to the graph page. This is a case where the GUI is the preferred method of working in **gretl**. This is because manipulating graphs provides immediate feedback and each one can be fine-tuned to suit your needs.

However, the **graphpg** commands can also be used. This is illustrated with the following example. First, the setup. The *wa_wheat.gdt* data are loaded and a new series, `t3`, is generated in order to estimate a cubic polynomial model of wheat yield for Greenough. The rescaling of time cubed merely changes the scale of the coefficient by a corresponding amount and has no effect on

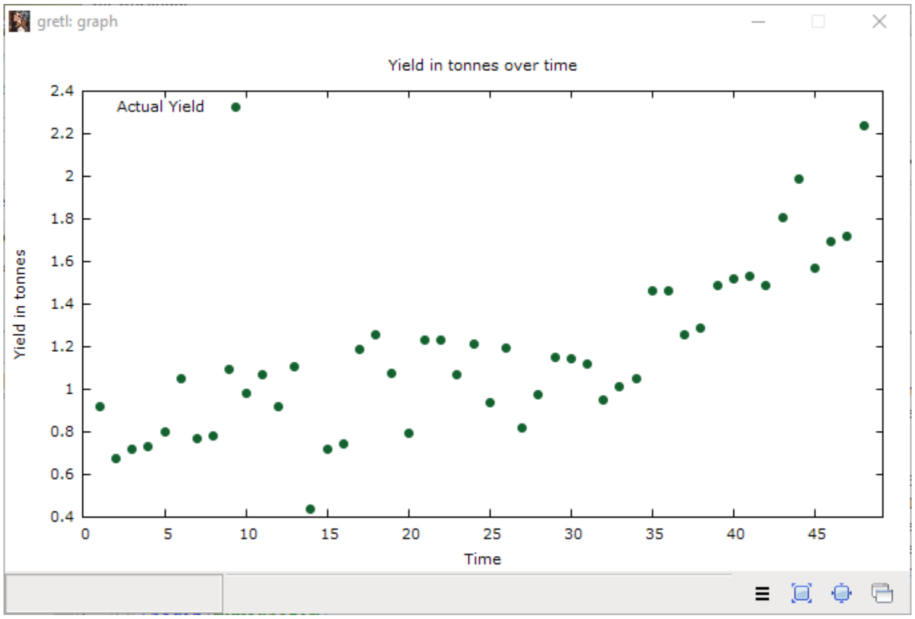


Figure 4.23: Plots wheat yield in Greenough Shire over time.

the shape or fit of the model. It 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). Then, each of the series are relabeled using the setinfo command.

```
open "@workdir\data\wa_wheat.gdt"
series t3=time^3/1000000

setinfo greenough -d "Wheat yield in tonnes" -n "Yield in tonnes"
setinfo time -d "Time" -n "Time"
setinfo t3 -d "(Time^3)/1,000,000" -n "(Time^3)/1,000,000"
```

The first graph generated is simply the yield in Greenough against time. It is added to the current session as g1 and the graphpg add command puts the graph into a **graph page**. The residuals from the regression are save as a series called ehat and setinfo is again used to provide a meaningful label.

```
ols greenough const time
g1 <- gnuplot greenough time --fit=linear --output=display
graphpg add

series ehat = $uhat
setinfo ehat -d "Residual, linear model" -n "Linear Residual"
```

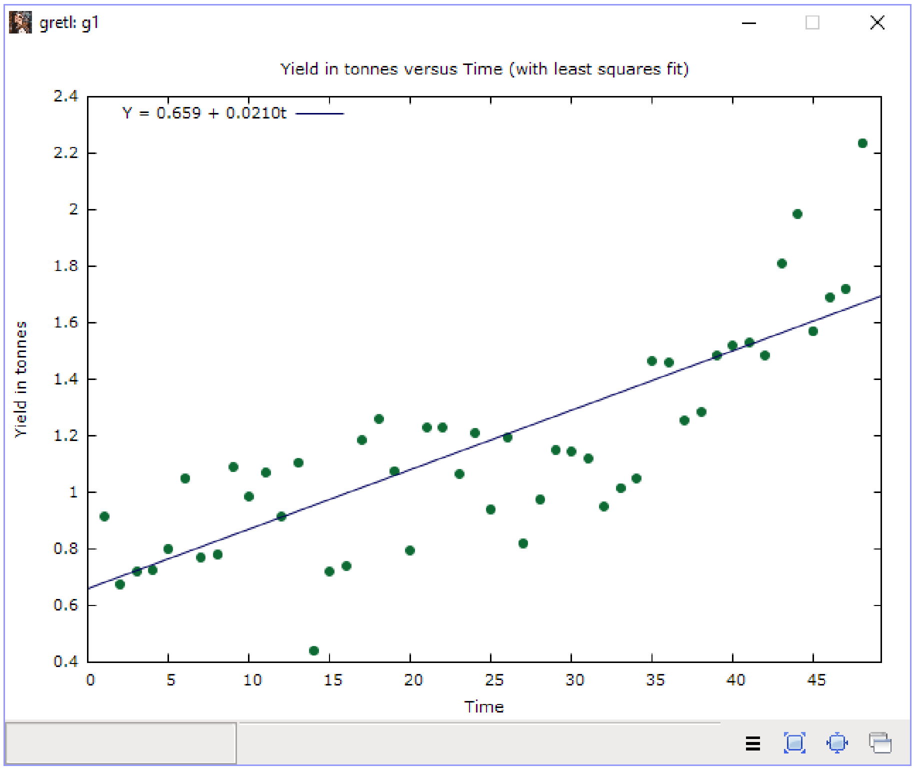


Figure 4.24: Plots wheat yield in Greenough Shire over time. The `--fit=linear` option is used and the **graph controls** were employed to change colors and observation markers.

The next graph plots the residuals from this linear regression against time and adds that to the session as `g2`. In the second line a title is added to the graph using a **gnuplot** command. The syntax is fussy. **gnuplot** commands can be issued within a **gretl** script if they are syntactically correct **and if they are enclosed in braces { }**. A **gnuplot** plot command ends with a semi-colon. This graph contains two **gnuplot** commands; one adds a title and the other labels the x-axis. Lines 7 and 8 both contain a continuation command, which means that lines 7-9 of the script makeup a single **gretl** command. Lines 8 and 9 consist of two **gnuplot** commands, the totality of which is enclosed in the braces.

The resulting graph is added to the graph page with the `graphpg add` command.

```
 7 g2 <- gnuplot ehat time --output=display \
 8   { set title "Plot of least squares residuals from linear model";  \
 9     set xlabel 'Time'; }
10 graphpg add
```

Then, the Greenough yield against time is plotted again, but this time yield is fit using a cubic function of time. This is put into the session as `g3`.

```
g3 <- gnuplot greenough time --fit=cubic --output=display
graphpg add
```

Finally, a simple linear regression is estimated:

$$yield_t = \beta_1 + \beta_2 t^3/1000000 + e_t$$

The residuals are saved, plotted against time, and added to the session as g4 and to the graph page.

```
g4 <- gnuplot ehat_3 time --output=display \
  { set title "Plot of least squares residuals from cubic model";  \
    set xlabel 'Time'; }
graphpg add
graphpg show
```

The graphpg show command produces a pdf graph shown in Figure 4.25.

## 4.7 Log-Linear Models

### 4.7.1 Growth Model

**Example 4.9 in *POE5***

Below you will find a script that reproduces the results from the growth model example in section 4.5.1 of *POE5*. 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

$$yield_t = yield_0(1 + g)^t \tag{4.11}$$

Taking the natural log

$$\ln(yield_t) = \ln(yield_0) + t\ln(1 + g) = \beta_1 + \beta_2 t \tag{4.12}$$

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.

```
open "@workdir\data\wa-wheat.gdt"
logs greenough
ols l_greenough 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.[8]

### 4.7.2 Wage Equation

**Example 4.10 in *POE5***

Below you will find a script that reproduces the results from the wage equation example in section 4.5.2 of *POE5*. In this example the log-linear model is used to measure the approximate return to another year of education. The example uses a thousand observations from the CPS monthly survey from 2008.

```
open "@workdir\data\cps5_small.gdt"
logs wage
ols l_wage const educ
t_interval($coeff(educ), $stderr(educ), $df, .95)
```

The regression results are:

$$\widehat{\text{l\_wage}} = \underset{(0.070180)}{1.59684} + \underset{(0.0048422)}{0.0987534}\,\text{educ}$$

$$T = 1200 \quad \bar{R}^2 = 0.2571 \quad F(1,1198) = 415.93 \quad \hat{\sigma} = 0.48470$$

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

This suggests that another year of schooling is expected to increase average wage by 9.88%.

The output from our `t_interval` command is:

```
The 0.95 confidence interval centered at 0.099 is (0.0893, 0.1083)
```

which suggests that an additional year of education is worth between 8.9% and 10.8% wage increases annually. Sign me up!

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

### 4.7.3 Generalized R-square

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 4.5.2 of *POE5*.

```
open "@workdir\data\cps5_small.gdt"
logs wage
ols l_wage const educ
series y = exp($yhat)
scalar corr1 = corr(y, wage)
scalar Rsquare = corr1^2
printf "\nThe correlation is %.3f and the Generalized R-square = %.3f\n",  corr1, Rsqu
```

This yields an estimated correlation of 0.465 and a squared correlation of 0.216.

### 4.7.4 Predictions in the Log-Linear Model

#### Example 4.11 in *POE5*

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 "@workdir\data\cps5_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 =  2.7818762
```

```
        nat_pred =  16.149292
  corrected_pred =  18.162196
```

That means that for a worker with 12 years of schooling the predicted wage is \$16.15/hour using the natural predictor and \$18.16/hour using the corrected one. In large samples we would expect the corrected predictor to be a bit better. Among the 1200 individuals in the sample, 307 of them have 12 years of schooling. Among those, the average wage is \$17.31. Hence the corrected prediction overshoots by about 85 cents/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 --simple
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 72 - 378
for the variable 'wage' (307 valid observations)

  Mean                         17.305
  Minimum                      4.1700
  Maximum                      45.650
  Standard deviation           7.9198
  Missing obs.                      0
```

which shows that the mean for the 307 observations is almost \$17.30. The last line `smpl full` restores the full sample.

## 4.8 Prediction Intervals

In this section, a function that computes in-sample prediction standard errors is proposed. It is based on the formulation from section 4.1 above. This formulation is generalized based on results found in Davidson and MacKinnon (2004, pp 103-104). They find the error variance for a given observation, $x_t$ to be

$$Var(y_t - x_t^T b) = \sigma_0^2 + \sigma_0^2 x_t (X^T X)^{-1} x_t^T$$

The entire set for a given sample is

$$diag(\sigma_0^2 + \sigma_0^2 X(X^T X)^{-1} X^T)$$

where the $t^{th}$ row of the $n \times k$ matrix $X$ is $x_t$.

The function to compute this is:

```
1 function series in_sample_fcast_error(series y, list xvars)
2     ols y xvars
3     scalar sig = $sigma^2
4     matrix X = { xvars }
5     matrix f_e = sig*I($nobs)+sig*X*inv(X'X)*X'
6     series se = sqrt(diag(f_e))
7     return se
8 end function
```

The function, called in_sample_fcast_error, returns a series to the dataset and takes two arguments. The first is a series that will serve as the dependent variable in a regression. The second is a list of regressors.

The first step is to estimate the model and save the estimated variance (sig). Then, the variable list is converted to a matrix and in line 5 the forecast error variance is computed. The next line takes the square root of the diagonal elements as a series and the return sends these out of the program.

To use the program, simply execute:

```
list xvars = const educ
series se_p = in_sample_fcast_error(l_wage, xvars)
```

### 4.8.1 The **fcast** Command

Gretl contains a forecast command, fcast, that returns the predictions and standard errors to a series. After the regression simply issue the following commands:

```
ols l_wage xvars
fcast f --static
series pred = $fcast
series se = $fcse
```

Since the last model estimated is a single equation, an optional variable name can be added as an argument to 1) suppress printing forecasts to the screen and to 2) place them in the dataset under the given name. In this case, a variable `f` is created to hold the forecasts in the dataset.

### Example 4.11 using `fcast`

Another way to add the forecasts to the data is through the `$fcast` accessor. It holds the forecasts, which in this case is simply `$yhat` in a static linear regression model. The other accessor, `$fcse`, returns the forecast standard error and reproduces the results from our program exactly. To print predictions, standard errors, and 95% prediction intervals to the screen, omit the optional variable name, `f`.

```
open "@workdir\data\cps5_small.gdt"
logs wage
list xvars = const educ
ols l_wage xvars
fcast f --static
series pred = $fcast
series se = $fcse

series corrected = exp(f)*exp($sigma^2/2)
series nat = exp(f)
dataset sortby educ
g6 <- gnuplot wage nat lb_p ub_p educ --output=display
```

After a little editing the To find the predicted values, standard errors, and 95% bounds for only those with 12 years of schooling use the smpl command to restrict the sample and use the simple summary statistics.

```
smpl educ==12 --restrict
summary educ wage lb_p nat ub_p se_p se --simple
smpl full
```

This produces:

```
                    Mean      Median        S.D.         Min         Max
educ               12.00       12.00      0.0000       12.00       12.00
wage               17.31       15.00       7.920       4.170       45.65
lb_p               6.236       6.236      0.0000       6.236       6.236
nat                16.15       16.15      0.0000       16.15       16.15
ub_p               41.82       41.82      0.0000       41.82       41.82
```

```
se_p            0.4850     0.4850     0.0000     0.4850     0.4850
se              0.4850     0.4850     0.0000     0.4850     0.4850
```

For individuals with 12 years of schooling, the average wage is \$17.31/hour and the median is only \$15. The natural prediction lies within the interval (\$6.24, \$16.15) with 95% frequency. That is not very informative, is it?

Another reasonable way 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 *POE5*.

```
1  open "@workdir\data\cps5_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]
22
23 g7 <- gnuplot wage lb ub ed --output=display --with-lines
```

Although there are probably more elegant ways to do this, the 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, and the overall variance of the model is saved to a scalar, `sig2`.

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 the contents of `i` placed on the $i^{th}$ row of the first column. The next line puts the prediction in the second column. The forecast standard error is put into column three and in the next two columns 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, it is easier to put the columns into a dataset and use 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. Using the `--with-lines` option prints out lines rather than dots to mark the observation. The graph (with a little editing) is found in Figure 4.27.

## 4.9 Log-Log Model

### Example 4.13 in *POE5*

Finally, a log-log model is estimated. 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* dataset 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 l_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 "@workdir\data\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

setobs 1 1 --cross-section
dataset sortby p
gnuplot corrected_pred q p --output=display
```

The results are

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

and the corresponding graph is found in Figure 4.28.

Notice that the series structure was changed from time series to a cross-section. Ordinarily, this is a terrible idea, but necessary in order to sort the data using the `dataset sortby` command. Once data are declared to be time series **gretl** will wisely not sort them. Sorting by the variable on the X-axis, however tends to make line graphs much more useful. Note, the default graph type in **gretl** uses dots, making the sort unnecessary.

The plot was edited to add titles, legends, and to change the markers and colors. The figure looks good. The nonlinear relationship between weight and price is quite evident and the fit is reasonable good.

## 4.10 Script

```
set echo off
set messages off
# function computes prediction standard errors
    function series in_sample_fcast_error(series y, list xvars)
    ols y xvars
    scalar sig = $sigma^2
    matrix X = { xvars }
    matrix f_e = sig*I($nobs)+sig*X*inv(X'X)*X'
    series se = sqrt(diag(f_e))
    return se
end function

# function estimates confidence intervals based on the t-distribution
function void t_interval(scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %.2f confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p, b, lb, ub
end function

# function to compute diagonals of hat matrix
function series h_t (list xvars)
    matrix X = { xvars }
    matrix Px = X*inv(X’X)*X’
    matrix h_t = diag(Px)
    series hats = h_t
    return hats
end function

# delete-one variance function
function series delete_1_variance(series y, list xvars)
    matrix sig = zeros($nobs,1)
    loop for i=1..$nobs --quiet
        matrix e_t = zeros($nobs,1)
        matrix e_t[i,1]=1
        series et = e_t
        ols y xvars et --quiet
        matrix sig[i,1]=$sigma^2
    endloop
    series sig_t = sig
    return sig_t
end function

# estimate model by LS and predict food_exp
open "@workdir\data\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) + $coeff(income)*20
scalar f=8013.2941+(8013.2941/40)+4.3818*(20-19.6047)^2
t_interval(yhat0,sqrt(f),$df,0.95)

# 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)
t_interval(yhat0,sqrt(f),$df,0.95)

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

# log-linear model
logs food_exp income
ols l_food_exp const income
```

```
series yhat2 = $yhat
gnuplot yhat2 food_exp income --output=display

# linear-log model
logs food_exp income
ols food_exp const l_income
series yhat2 = $yhat
gnuplot yhat2 food_exp income --output=display

# normality tests
open "@workdir\data\food.gdt"
ols food_exp const income
series uhat2 = $uhat
summary uhat2
normtest uhat2 --jbera
normtest uhat2 --all
modtest --normality

# Example 4.7 Influential Observations
open "@workdir\data\food.gdt"
genr index
set echo off
list xvars = const income
ols food_exp xvars
leverage --save

series uhat = $uhat
series lev_t = h_t(xvars)
series sig_t = delete_1_variance(food_exp, xvars)
series stu_res = uhat/sqrt(sig_t*(1-lev_t))
series DFFits=stu_res*sqrt(lev_t/(1-lev_t))

list x1 = const income
scalar k = nelem(x1)
matrix results = zeros(k,1)
loop i=1..k --quiet
    list y1 = x1[1]
    list y2 = x1[2:k]
    ols y1 y2
    series dfb$i=stu_res*$uhat/sqrt($ess*(1-lev_t))
    list x1 = y2 y1
endloop

print sig_t lev_t stu_res DFFits dfb2 --byobs

# Example 4.8
# polynomial
open "@workdir\data\wa_wheat.gdt"
series t3=time^3/1000000

setinfo greenough -d "Wheat yield in tonnes" -n "Yield in tonnes"
setinfo time -d "Time" -n "Time"
setinfo t3 -d "(Time^3)/1,000,000" -n "(Time^3)/1,000,000"

ols greenough const time
gnuplot greenough time --output=display

ols greenough const time
g1 <- gnuplot greenough time --fit=linear --output=display
graphpg add
series ehat = $uhat
setinfo ehat -d "Residual, linear model" -n "Linear Residual"

g2 <- gnuplot ehat time --output=display \
  { set title "Plot of least squares residuals from linear model";  \
  set xlabel ’Time’; }
graphpg add

g3 <- gnuplot greenough time --fit=cubic --output=display
graphpg add

ols greenough const t3
series ehat_3 = $uhat
setinfo ehat_3 -d "Residual, cubic model" -n "Cubic Residual"

g4 <- gnuplot ehat_3 time --output=display \
  { set title "Plot of least squares residuals from cubic model";  \
  set xlabel ’Time’; }
graphpg add
graphpg show

# Example 4.9
open "@workdir\data\wa_wheat.gdt"
logs greenough
ols l_greenough const time

# Example 4.10
# log-linear model
open "@workdir\data\cps5_small.gdt"
logs wage
ols l_wage const educ
t_interval($coeff(educ), $stderr(educ), $df, .95)

open "@workdir\data\cps5_small.gdt"
logs wage
ols l_wage const educ
series l_yhat = $yhat
series y = exp(l_yhat)
scalar corr1 = corr(y, wage)
scalar Rsquare = corr1^2
printf "\nThe correlation is %.3f and the Generalized\
R-square = %.3f\n", corr1, Rsquare
```

```

# Example 4.11
# simple prediction in log-linear model
open "@workdir\data\cps5_small.gdt"
logs wage
list xvars = const educ
ols l_wage xvars

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

# Predictions using fcast
open "@workdir\data\cps5_small.gdt"
logs wage
list xvars = const educ
ols l_wage xvars
fcast f --static
series pred = $fcast
series se = $fcse

series corrected = exp(f)*exp($sigma^2/2)
series nat = exp(f)

series se_p = in_sample_fcast_error(l_wage, xvars)
series lb_p = exp(f - critical(t,$df,0.025)*se)
series ub_p = exp(f + critical(t,$df,0.025)*se)

dataset sortby educ
g6 <- gnuplot wage nat lb_p ub_p educ --output=display

smpl educ==12 --restrict
summary wage --simple
summary educ wage lb_p nat ub_p se_p se --simple
smpl full

# prediction intervals using a loop
open "@workdir\data\cps5_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]

g7 <- gnuplot wage lb ub ed --output=display --with-lines

# Example 4.13
# corrected predictions in log-linear model
open "@workdir\data\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

setobs 1 1 --cross-section
dataset sortby p
gnuplot corrected_pred q p --output=display
```

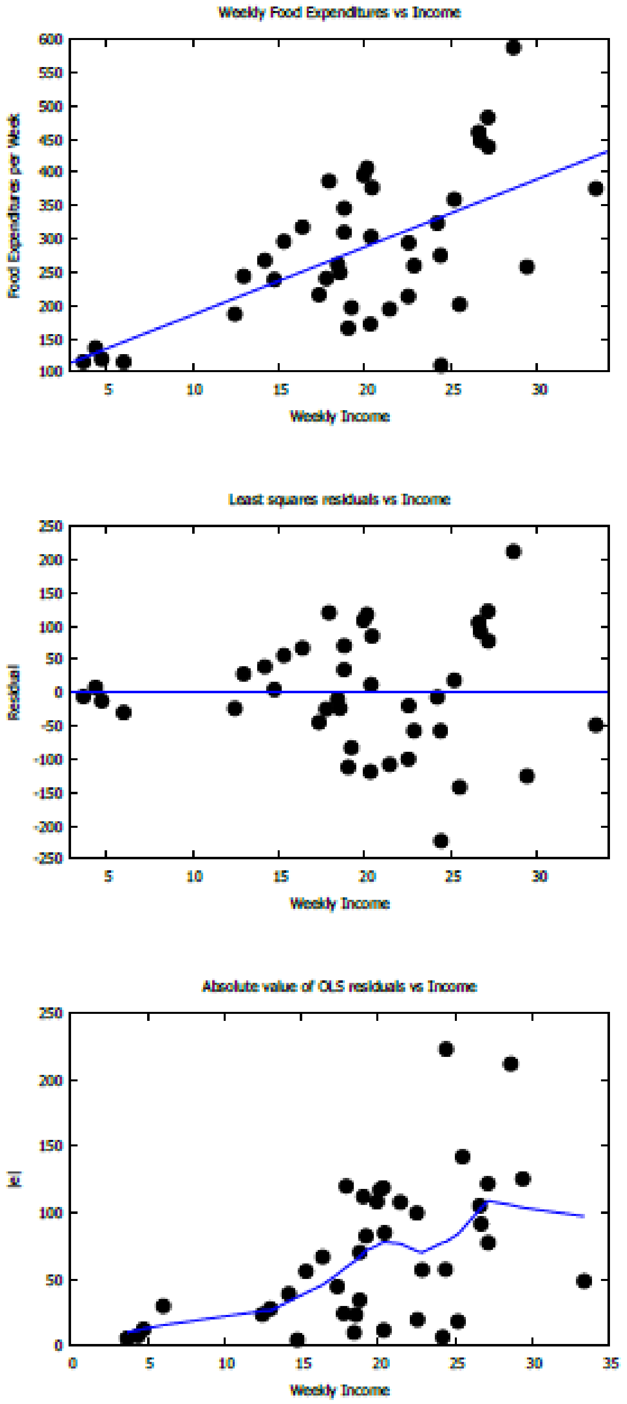


Figure 4.25: Plots of linear and cubic models of wheat yield in Greenough Shire over time.

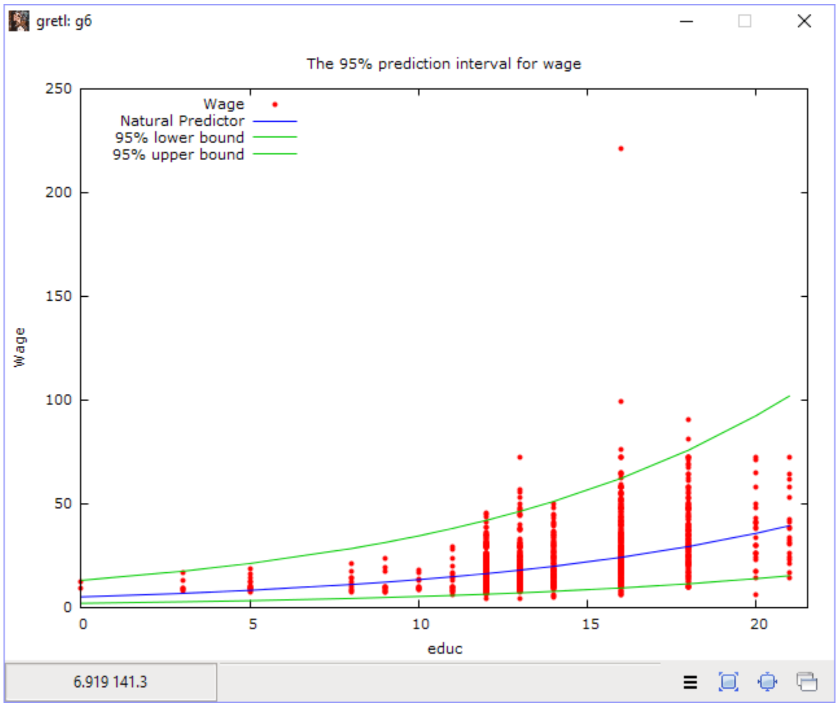


Figure 4.26: This is a plot generated using statistics from `fcast`.

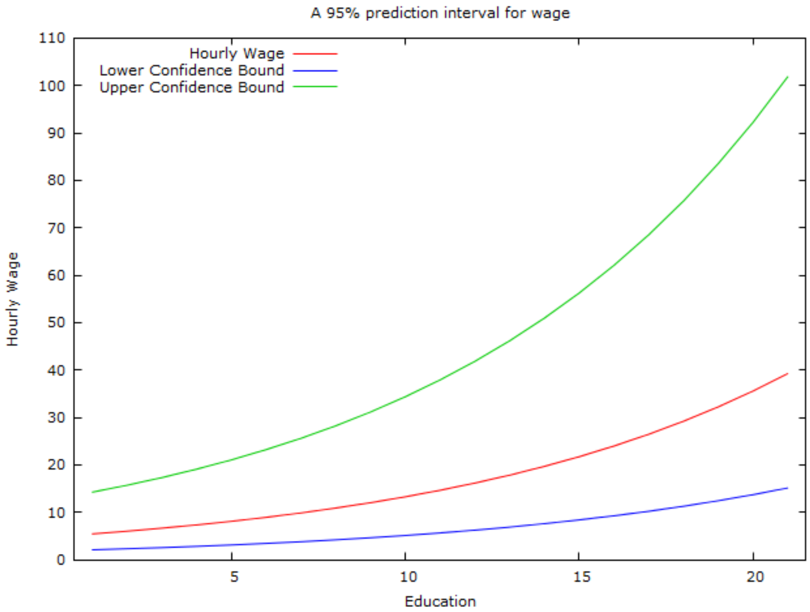


Figure 4.27: This is a plot generated using a loop to estimate forecast standard errors.

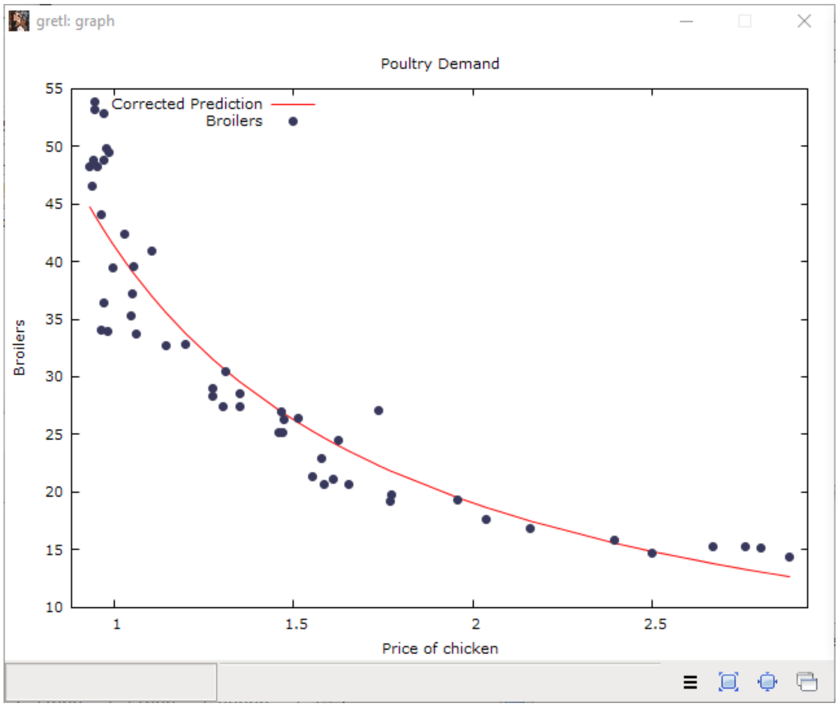


Figure 4.28: This is a plot generated from a log-log model of chicken demand.

# 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 imposes an additional requirement upon the data. 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, $x_{ij}$ is the $i^{th}$ observation on the $j^{th}$ independent variable, $j = 2, 3, \ldots, k$, $e_i$ 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, $e_i|x_{ij}$, has an average value of zero for each value of the $j$ independent variables; each has the same variance, $\sigma^2$, and are uncorrelated with any of the other errors.

To estimate each of the $\beta s$, none of the independent variables can be an exact linear combination of the others. This serves the same purpose as the requirement that the independent variable of the simple linear regression take on at least two different values in the sample. 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 $x_{ij}$ 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 first example used in this chapter is a sales model 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 Preliminary Chores

### Example 5.1 in *POE5*

Before estimating the model, relabel the data and find the summary statistics. Data labels are used in much of the output produced by **gretl**. If the data you are working with are not labeled satisfactorily, then this output will have to be further manipulated when assembling it for inclusion for reports or papers.

```
setinfo sales --description="Monthly sales revenue ($1000)" \
              --graph-name="Monthly Sales ($1000)"
setinfo price --description="Price in dollars" --graph-name="Price"
setinfo advert --description="Monthly Advertising Expenditure ($1000)" \
               --graph-name="Monthly Advertising ($1000)"
# print the new labels to the screen
labels
```

The output from the `labels` command is:

```
Listing labels for variables:
 sales: Monthly sales revenue ($1000)
 price: Price in dollars
 advert: Monthly Advertising Expenditure ($1000)
```

Editing variable attributes is also available via **Variables>Edit attributes** from the main menu or as a right-click pop-up from the main **gretl** window. Simply highlight the desired series, right-click, and choose **Edit attributes** from the fly-out menu.

Next, find the summary statistics using:

```
summary sales price advert
```

which produces:

```
                    Mean      Median        S.D.         Min         Max
sales              77.37       76.50       6.489       62.40       91.20
price              5.687       5.690      0.5184       4.830       6.490
advert             1.844       1.800      0.8317      0.5000       3.100
```

Average sales, because they are measured in \$1000, is \$77,370. Average price is \$5.69 and average advertising expenditure is \$1844. It is always wise to keep track of the actual units that you are working with. This is critical to understanding the economic meaning of the coefficient magnitudes from the regression.

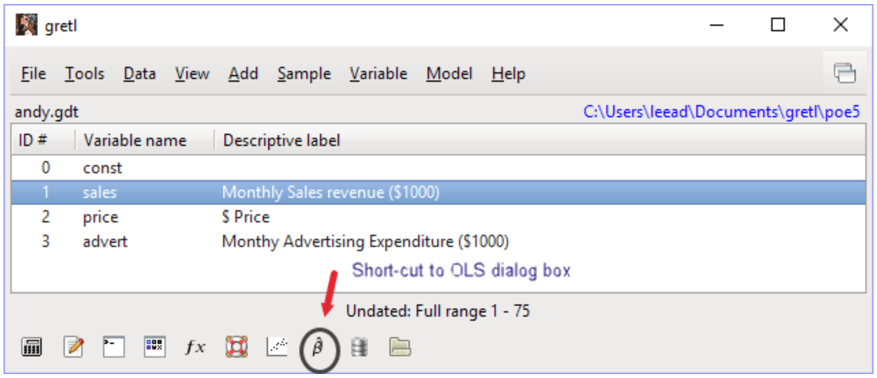
## 5.2 Linear Regression

The parameters of the model are estimated by least squares using the pull-down menus and dialog boxes (GUI) or **gretl**'s handy scripting language (**hansl**). Although this was discussed in some depth in Chapter 2, both of these will be demonstrated again below.

There are two ways to open the dialog box. As in Chapter 2, one can use the pull-down menu. Select **Model>Ordinary Least Squares** from the main **gretl** window as shown in Figure 2.6.

This brings up the dialog box shown in Figure 2.7. As in Chapter 2 you must put the dependent variable, in this case `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 also a shortcut on the toolbar that opens the specify model (Figure 2.7 dialog box. Recall that the toolbar is located at the bottom of the main **gretl** window, There you will find a button labeled $\hat{\beta}$. Clicking on this button opens the OLS **specify model** dialog.



andy.gdt C:\Users\leead\Documents\gretl\poe5

| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | sales | Monthly Sales revenue ($1000) |
| 2 | price | $ Price |
| 3 | advert | Monthy Advertising Expenditure ($1000) |

Short-cut to OLS dialog box

Undated: Full range 1 - 75

Figure 5.1: The OLS shortcut button on the toolbar.

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

## 5.3 Big Andy's Burger Barn

### Example 5.2 in *POE5*

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 "@workdir\data\andy.gdt"
m1 <- ols sales const price advert
```

This assumes that the **gretl** data set *andy.gdt* has been installed in a data folder[1] located in the **gretl** working directory. The model is estimated and the result is saved to your current session as `m1`. m1 contains the output of a models window, giving you full access to the GUI after running a script. From the session window, you can click on `m1` to revisit the results.

The results were copied using **LaTeX>Copy>Tabular** from the models window, pasted into the source code for this chapter, and appear in Table 5.1. This illustrates what these look like in use. Keep in mind, once pasted into a text file for LaTeX compilation, you can edit the format and contents as you wish. Omit statistics, change titles, combine with other results. The output appears in Table 5.1 and match those in *POE5*.

[1] Depending on your OS, a folder may be referred to as a *directory*.

### Example 5.3 in *POE5*

Next, a prediction of sales for meals priced at \$5.50 and advertising expenditures of \$1200 is made. Again, the accessors for the estimated regression coefficients are used to create the scalar prediction.

```
scalar S_hat = $coeff(const) + $coeff(price)*5.5 + $coeff(advert)*1.2
printf "\nPredicted sales when price=\
$5.50 and advertising=1200 is $%.3f\n", S_hat
```

This produces:

```
Predicted sales when price=$5.50 and advertising=$1200 is $77655.51
```

## 5.4 Goodness-of-Fit

### Example 5.4 in *POE5*

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

$$\hat{\sigma}^2 = \frac{SSE}{n-k} = \frac{1718.94}{75-3} = 23.874 \tag{5.3}$$

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, use **Analysis>ANOVA** from the model's pull-down menu to produce the ANOVA table. Refer to section 4.2 for details.

To compute $R^2$ from the standard **gretl** output recall that

$$\hat{\sigma}_y = \sqrt{\frac{SST}{n-1}} \tag{5.4}$$

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

$$SST = (n-1)\hat{\sigma}_y^2 = 74 * 6.48854 = 3115.485 \tag{5.5}$$

Then,

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{1718.94}{3115.485} = 0.448 \tag{5.6}$$

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 imposed from adding variables. Thus, adjusted $R^2$ may become smaller as variables are added. The formula is:

$$\bar{R}^2 = 1 - \frac{SSE/(n-k)}{SST/(n-1)} \tag{5.7}$$

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.

### 5.4.1 Variances and Covariances of Least Squares

**Example 5.5 in *POE5***

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, $\sigma^2$, 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 collinearity among the independent variables 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.

$$cov(b_1, b_2, b_3) = \begin{bmatrix} var(b_1) & cov(b_1, b_2) & cov(b_1, b_3) \\ cov(b_1, b_2) & var(b_2) & cov(b_2, b_3) \\ cov(b_1, b_3) & cov(b_2, b_3) & var(b_2) \end{bmatrix} \tag{5.8}$$

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 the 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, $\widehat{se}(b_2) = \sqrt{\widehat{var}(b_2)}$. Assign the contents of the variance-covariance accessor to a matrix. Take the square roots of the diagonal elements to obtain the estimated standard errors.

```
matrix covmat = $vcv
matrix se = sqrt(diag(covmat))
printf "Least squares standard errors:\n%.3f\n", se
```

These are printed:

```
Least Squares standard errors:
6.352
1.096
0.683
```

These match those found in the output table (Table 5.1) in-between the least squares estimates and $t$-ratios.

### 5.4.2 Confidence Intervals

### Example 5.6 in *POE5*

Confidence intervals can be obtained using the `scalar` command in the same way as in Chapter 3. In this section we reuse our `t_interval` function. A 95% confidence interval for $\beta_2$, 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
```

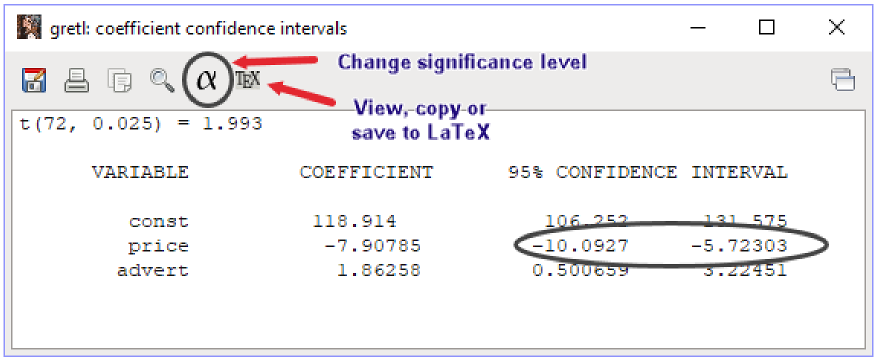
or using the function:

```
t_interval($coeff(price), $stderr(price), $df, 0.95)
```

The output produced by the `t_interval` function is:

```
The 0.95 confidence interval centered at -7.908 is (-10.0927, -5.7230)
```

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 $\beta_2$ is shown below in Figure 5.2.



Change significance level

View, copy or save to LaTeX

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

### Example 5.7 in *POE5*

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

$$\lambda = E(sales_1) - E(sales_0) = -0.4\beta_2 + 0.8\beta_3 \tag{5.9}$$

The estimate of $\lambda$ 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:

```
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)*$vcv[2,3])
t_interval(chg,se_chg,$df,.95)
```

This produces the expected result:

```
The 95% confidence interval centered at 4.653 is (3.2386, 6.0678)
```

## 5.4.3 *t*-Tests, Critical Values, and *p*-values

In section 3.5 the GUI was used to obtain test statistics, critical values and $p$-values. However, it is often much easier to use 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.

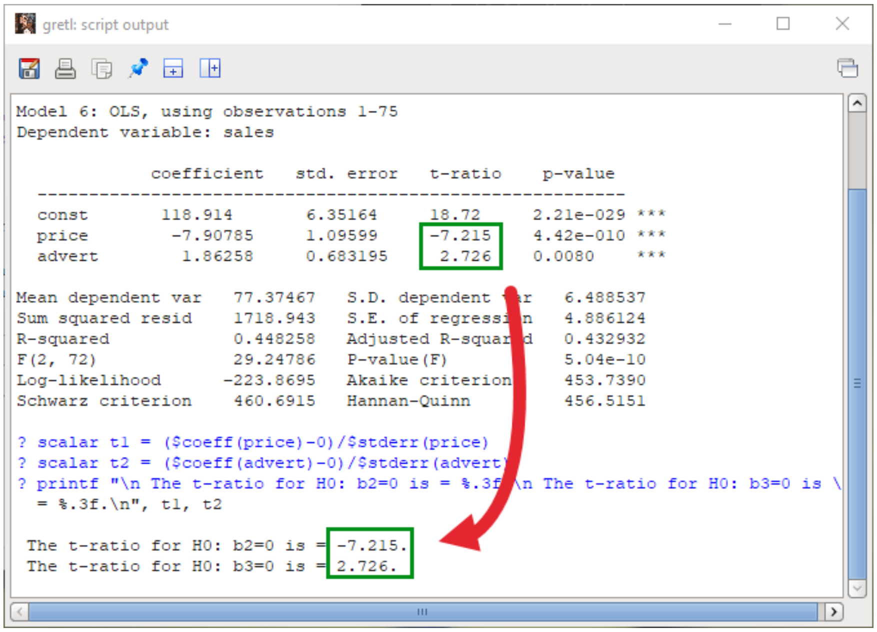
### Significance Tests

### Examples 5.8 and 5.9

Multiple regression models include 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 $\beta_j = 0$ against the alternative $\beta_j \neq 0$ 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 5.3 As you can see, the automatic results and the manually generated ones match perfectly.



```
Model 6: 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

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

Figure 5.3: 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.

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.

Rather than comparing the statistic to a critical value one could compare the $p$-value to the desired level of significance. If $p > \alpha$ then do not reject $H_0$. If $p < \alpha$, reject $H_0$. Gretl includes a `pvalue` function that computes $p$-values from various probability distributions. The syntax is very similar to that of critical. The difference is that instead of using $\alpha/2$ as the third argument, use the computed statistic.

`pvalue` computes the area to the right of `stat` in the specified distribution (`z` for Gaussian, `t` for Student's t, `X` for chi-square, `F` for F, `G` for gamma, `B` for binomial, `P` for Poisson, `exp` for Exponential, `W` for Weibull). So, to compute a $p$-value for a $t$-statistic use:

```
pvalue(t,$df,stat)                  # prints to the screen
scalar pval = pvalue(t,$df,stat)   # saves Prob(stat>p) to scalar pval
```

The argument(s) in the middle is (are) the shape parameter(s). In our case it should be $n - k$, which is the residual degrees of freedom from the Big Andy regression. Some distributions like the $F_{J,n-k}$ have two parameters. Refer to the **gretl** help for details on how to use pvalue in those situations.

For the examples we have

```
scalar t2 = ($coeff(advert)-0)/$stderr(advert)
scalar t3 = ($coeff(advert)-1)/$stderr(advert)
pvalue t $df t1
pvalue t $df t3
```

which produces:

```
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)
```

You can see that the function computes and prints areas to the right, left and the two-tailed $p$-values for the computed values of `t2` and `t3`, respectively. Advertising is significantly different from zero at the 5% level. It is not significantly different from 1 at 5%.

When used as a function, `pvalue` returns the area to the right of the statistic as a scalar.

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

which produces:

```
print pval

              p =  0.10540831
```

### One-tail Alternatives

### Example 5.10 in *POE5*

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 $H_1$: $\beta_2 < 0$, the test statistic is the usual $t$-ratio.

```
ols sales const price advert
scalar t = ($coeff(price)-0)/$stderr(price)
scalar crit = -critical(t,$df,0.05)
scalar pval = 1-pvalue(t,$df,t)
printf "\n Ho: b2=0 vs Ha: b2<0 \n \
the t-ratio is = %.3f. \n \
the critical value = %.3f \n \
and the p-value = %.3f\n", t, crit, pval
```

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

```
Ho: b2=0 vs Ha: b2<0
  the t-ratio is = -7.215.
  the critical value = -1.666
  and the p-value = 0.000
```

You can see that the $t$-ratio $-7.21524$ lies to the left of the critical value $-1.666$. The $p$-value is close to zero. That is less than 5% nominal level of the test and therefore we reject that $\beta_2$ is non-negative.

### Example 5.11 in *POE5*

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.

```
ols sales const price advert
scalar t = ($coeff(advert)-1)/$stderr(advert)
scalar crit = critical(t,$df,0.05)
scalar pval = pvalue(t,$df,t)
printf "\n Ho: b3=1 vs Ha: b3>1 \n \
    the t-ratio is = %.3f \n \
    the critical value = %.3f \n \
    and the p-value = %.3f\n", t, crit, pval
```

The results are

```
Ho: b3=1 vs Ha: b3>1
  the t-ratio is = 1.263
  the critical value = 1.666
  and the p-value = 0.105
```

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

### Example 5.12 in *POE5*

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:

$$H_0\colon -0.2\beta_2 - 0.5\beta_3 \leq 0$$
$$H_1\colon -0.2\beta_2 - 0.5\beta_3 > 0$$

The test statistic is

$$t = \frac{-0.2b_2 - 0.5b_3}{se(-0.2b_2 - 0.5b_3)} \sim t_{72} \tag{5.10}$$

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])

printf "\n Ho: d=-0.2b2-0.5b3=0 vs Ha: d > 0 \n \
the t-ratio is = %.3f \n \
the critical value = %.3f \n \
and the p-value = %.3f\n", \
chg/se_chg, critical(t,$df,0.05), pvalue(t,$df,t_ratio)
```

which generates the needed information to perform the test. Notice that the computations for the $t$-ratio, critical value and $p$-value were carried out within the `printf` statement.

```
Ho: d=-0.2b2-0.5b3=0 vs Ha: d > 0
  the t-ratio is = 1.622
  the critical value = 1.666
  and the p-value = 0.055
```

The results matches the ones in *POE5* 5. The hypothesis is not rejected at the 5% level. We conclude that the proposed changes will not increase sales.

An alternate way to obtain the variance of the linear combination is to use matrix algebra. The main advantage of this is that it reduces the opportunity to make a coding error in the computation. The linear combination of parameters,

$$-0.2b_2 - 0.5b_3 = \begin{bmatrix} 0 & -0.2 & -0.5 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \mathbf{d}^T\mathbf{b}$$

where $\mathbf{d}$ and $\mathbf{b}$ are $3 \times 1$ vectors. As the least squares estimator $\mathbf{b} \sim (\beta, Cov(\mathbf{b}))$. Estimating $Cov(\mathbf{b})$ with $\widehat{Cov(\mathbf{b})}$, the estimated $Cov(\mathbf{d}^T\mathbf{b})$ is

$$\mathbf{d}^T\widehat{Cov(\mathbf{b})}\mathbf{d}$$

In **gretl**

```
ols sales const price advert
matrix covmat = $vcv
matrix d = { 0; -0.2; -0.5 }
matrix covest = d'*covmat*d
scalar se = sqrt(covest)
printf "\nThe estimated standard error of the\
linear combination is %.3f\n", se
```

This yields the same result as previously obtained:

```
The estimated standard error of the linear combination is 0.4010
```

The benefits of using this method increase exponentially as the number of coefficients in the linear combination increases. It takes 3 lines of code no matter how many coefficients are used in the linear combination. Just change coefficients in the vector $\mathbf{d}$ accordingly.

## 5.5 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:

$$sales_i = \beta_1 + \beta_2 price_i + \beta_3 advert_i + \beta_4 advert_i^2 + e_i \quad i = 1, 2, \ldots, n \tag{5.11}$$

To estimate the parameters of this model, one creates the new variable, $advert_i^2$, 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 the magnitude of 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.

```
square advert price
```

### 5.5.1 Marginal Effects

**Example 5.14 in *POE5***

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;

$$\frac{\partial E(sales)}{\partial advert} = \beta_3 + 2\beta_4 advert \tag{5.12}$$

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:

```
series a2 = advert^2
ols sales price advert a2
scalar me1 = $coeff(advert)+2*(0.5)*$coeff(a2)
scalar me2 = $coeff(advert)+2*2*$coeff(a2)
printf "\n The marginal effect at \$500 (advert=.5) is %.3f\n\
and at \$2000 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
```

### 5.5.2 Interaction in a Wage Equation

**Example 5.15 in *POE5***

In this example experience and education are interacted. The idea is that the level of experience affects the return to another year of schooling (or, another year of education affects the return to another year of experience). The model becomes:

$$wage = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 educ \times exper + e$$

The marginal effects depend on levels of education and experience. These are measured for workers having 8 and 16 years of schooling and for workers having 20 years experience.

$$\frac{\partial E(wage|educ, exper)}{\partial exper} = \beta_1 + \beta_4 educ$$
$$\frac{\partial E(wage|educ, exper)}{\partial educ} = \beta_1 + \beta_4 exper$$

This is estimated using the *cps5_small.gdt* data using the following script:

```
open "@workdir\data\cps5_small.gdt"
series educ_exper = educ*exper
ols wage const educ exper educ_exper

scalar me_8year =  $coeff(exper)+$coeff(educ_exper)*8
scalar me_16year = $coeff(exper)+$coeff(educ_exper)*16
scalar me_20year = $coeff(exper)+$coeff(educ_exper)*20
scalar me_ed_20exper = $coeff(educ)+$coeff(educ_exper)*20

printf "\nMarginal effect of another year of schooling when:\n\
experience is 0 = %.3f\n\
experience is 20 = %.3f\n", $coeff(educ), me_ed_20exper
printf "\nMarginal effect of experience when:\n\
education is 8 = %.3f \n\
education is 16 = %.3f \n\
education is 20 = %.3f \n", me_8year, me_16year, me_20year
```

The results are:

```
Marginal effect of another year of schooling when:
 experience is 0 = 2.656
 experience is 20 = 2.601

Marginal effect of experience when:
 education is 8 = 0.216
 education is 16 = 0.194
 education is 20 = 0.183
```

### Example 5.16 in *POE5*

In this example a log-quadratic model is estimated and marginal effects computed. The model becomes

$$\ln(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 educ \times exper + \beta_5 exper^2 + e$$

The marginal effects are:

$$\frac{\partial E(\ln(wage)|educ, exper)}{\partial exper} = \beta_3 + \beta_4 educ + 2\beta_5 exper$$
$$\frac{\partial E(\ln(wage)|educ, exper)}{\partial educ} = \beta_2 + \beta_4 exper$$

There are quite a few combination of 0 and 20 years of experience and 8 and 16 years of schooling to consider. To facilitate this, I have written functions that allow me to consider these and other combinations easily.

The function for the first marginal effect (the % change in avg wage from another year of experience, given years of schooling) is:

```
function void me_1(list vars "all variables, including dep var first",
      scalar ed "set years of schooling",
      scalar expr "set years of experience")
    ols vars --quiet
    scalar me = $coeff(exper) + $coeff(educ_exper)*ed +\
      2*$coeff(sq_exper)*expr
    printf "\nMarginal effect of another year of experience:\n \
 Education = %.3g years and Experience = %.3g years\n \
 Marginal effect is %.3f percent \n", ed, expr, me*100
end function
```

The function saves a lot of typing since the equation for the marginal effect only depends on two scalar inputs (`educ`, and `exper`). Hence the function will work for whatever combination you choose to enter. It also economizes on the programming of the somewhat fussy to program `printf` statement. The function returns nothing (void) and takes 3 inputs. The variables from the regression, the desired number of years of education, and the desired years of experience. Including the regression in the function is not a great idea since the marginal effect will change depends on the presence of the education, experience, their interaction, and squared experience in the model. Other variable could be added without trouble.[2] That said, here is how we call it. First list the variables for the model starting with the dependent variable, `l_wage`. Be sure to include a constant and `educ`, `exper`, `educ_exper`, and `sq_exper`. The second argument is years of schooling and the third is years of experience at which the marginal effect will be measured.

```
list regression = l_wage const educ exper educ_exper sq_exper

me_1(regression,  8, 0)
me_1(regression, 16, 0)
me_1(regression,  8, 20)
me_1(regression, 16, 20)
```

[2] However, **this function is a not meant to be used generally**, but only as a time saver in this specific context. Don't try to use this on another model without properly modifying the code.

This yields:

```
Marginal effect of another year of experience:
  Education = 8 years and Experience = 0 years
  Marginal effect is 3.875 percent

Marginal effect of another year of experience:
  Education = 16 years and Experience = 0 years
  Marginal effect is 2.861 percent

Marginal effect of another year of experience:
  Education = 8 years and Experience = 20 years
  Marginal effect is 1.979 percent

Marginal effect of another year of experience:
  Education = 16 years and Experience = 20 years
  Marginal effect is 0.965 percent
```

A similar function can be written for the marginal effect of another year of schooling. Since its marginal effect is simpler it is likely to be more trouble that its worth, however once the other marginal effect is programmed modifying it for the second one is trivial. Here is the function:

```
1 function void me_2(list vars "all variables, including dep var first",
2       scalar ed "set years of schooling",
3       scalar expr "set years of experience")
4     ols vars --quiet
5     scalar mw = $coeff(educ) + $coeff(educ_exper)*expr
6     printf "\nMarginal effect of another year of schooling:\n \
7  Education = %.3g years and Experience = %.3g years\n \
8  Marginal effect is %.3f percent \n", ed, expr, mw*100
9 end function
```

Notice that only line 5 is different. It can be called similarly,

```
list regression = l_wage const educ exper educ_exper sq_exper

me_2(regression,  8, 0)
me_2(regression, 16, 0)
me_2(regression,  8, 20)
me_2(regression, 16, 20)
```

and the results:

```
Marginal effect of another year of schooling:
   Education = 8 years and Experience = 0 years
```

```
     Marginal effect is 13.595 percent

  Marginal effect of another year of schooling:
     Education = 16 years and Experience = 0 years
     Marginal effect is 13.595 percent

  Marginal effect of another year of schooling:
     Education = 8 years and Experience = 20 years
     Marginal effect is 11.059 percent

  Marginal effect of another year of schooling:
     Education = 16 years and Experience = 20 years
     Marginal effect is 11.059 percent
```

Obviously, the marginal effect no longer depends on the years of schooling, only on the years of experience. Hence the repetition of results.

## 5.6 Nonlinear Combinations of Parameters

### 5.6.1 Optimal level of advertising

**Example 5.17 in *POE5***

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,

$$\beta_3 + 2\beta_4 advert_o = 1 \tag{5.13}$$

Solving *advert* in terms of the parameters

$$advert_o = g(\boldsymbol{\beta}) = \frac{1-\beta_3}{2\beta_4} \tag{5.14}$$

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 depends on the parameters of the model. Let $\boldsymbol{\beta}$ be a $2 \times 1$ 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:

$$g(\boldsymbol{b}) \overset{a}{\sim} N(g(\boldsymbol{\beta}), G(\boldsymbol{\beta})VG(\boldsymbol{\beta})^T) \tag{5.15}$$

where $G(\boldsymbol{\beta}) = \partial g(\boldsymbol{\beta})/\partial \boldsymbol{\beta}^T$. Hence, to use the delta method requires that you 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, $\beta_1$, $\beta_2$, $\beta_3$, and $\beta_4$ yields:

$$\begin{aligned}
d_1 &= \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_1} = 0 \\
d_2 &= \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_2} = 0 \\
d_3 &= \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_3} = -\frac{1}{2\beta_4} \qquad (5.16) \\
d_4 &= \frac{\partial g(\boldsymbol{\beta})}{\partial \beta_4} = -\frac{1 - \beta_3}{2\beta_4^2} \qquad (5.17)
\end{aligned}$$

Note that the derivatives with respect to $\beta_1$ and $\beta_2$ are 0. To use the delta method, simply replace the unknown parameters in equation (5.14) with least squares estimates. Then to get the estimated standard error of $\widehat{g(\boldsymbol{b})}$, substituted estimates into the derivatives $d_3$ and $d_4$, and compute

$$Var(\widehat{g(\boldsymbol{b})}) = \begin{pmatrix} 0 & 0 & \hat{d}_3 & \hat{d}_4 \end{pmatrix} [\widehat{Cov}(b_1, b_2, b_3, b_4)] \begin{pmatrix} 0 \\ 0 \\ \hat{d}_3 \\ \hat{d}_4 \end{pmatrix} \tag{5.18}$$

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

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

The first line estimates the model using least squares and the `--vcv` option is used to print the covariance matrix. In line 2 the entire set of coefficients 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 of $g(\boldsymbol{\beta})$. In lines 5 and 6 the analytical derivatives are evaluated at the estimates. The matrix `d` is $1 \times 4$ and contains the derivatives of the hypothesis with respect to each of the parameters. The next line computes variance in equation (5.18). 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 14 and 15.

```
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).

### 5.6.2 How much experience maximizes wage?

#### Example 5.18 in *POE5*

Consider the log-wage equation estimated using the *cps5_small.gdt* dataset.

$$\ln(wage) = \beta_1 + \beta_2 educ + \beta_3 exper + \beta_4 educ \times exper + \beta_5 exper^2 + e$$

To determine the level of schooling that maximizes average log-wage (and hence, the wage) differentiate the mean of the model with respect to education and set the result equal to zero. Then, solving for experience you get:

$$exper_o = g(\boldsymbol{\beta}) = \frac{-\beta_3 - \beta_4 educ}{2\beta_5}$$

Estimating this point is simple. Estimate the linear model's five parameters using least squares, choose a value of eduction at which it will be evaluated (e.g., *educ*=16) and plug these into the formula. This is nonlinear function of the least squares estimates and the delta method is used to obtain its variance.

The partial derivatives of the function with respect to each of the parameters are:

$$\frac{\partial exper_o}{\partial \beta_3} = -1/2\beta_5$$

$$\frac{\partial exper_o}{\partial \beta_4} = -16/2\beta_5$$

$$\frac{\partial exper_o}{\partial \beta_5} = (\beta_3 + 16\beta_4)/2\beta_5^2$$

The estimated vector of partial derivatives becomes

$$\widehat{\mathbf{d}} = \begin{pmatrix} 0 & 0 & -1/2b_5 & -16/2b_5 & (b_3 + 16b_4)/2b_5^2 \end{pmatrix}$$

The estimated variance is

$$\widehat{Var}(exper_o) = \widehat{\mathbf{d}}^T \widehat{Cov}(\mathbf{b})\widehat{\mathbf{d}}$$

where $\widehat{Cov}(\mathbf{b})$ is the estimated least squares covariance matrix from the linear model.

### Numerical derivatives

The analytic derivatives in this example are not hard to obtain, 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 fdjac function in **gretl** takes numeric derivatives. fdjac stands for **first difference Jacobian**. The fdjac function requires two arguments: a function, $g(\beta)$, for which a derivative is desired and a vector of parameters, $\beta$, with which the derivatives will be taken. To illustrate its use, consider the new matrix function for marginal effects below.

```
function matrix G(matrix *param, list x)
    matrix X = { x }
    matrix r1 = (-param[3].*ones($nobs,1)- param[4]*X)./(2*param[5])
    return r1
end function
```

The * that prefixes the param argument is a **pointer**, the use of which is discussed below. Before discusing its use, another function is written to evaluate $g(\cdot)$ at the least squares estimates for a specific number of schooling years.

```
# Function computes the optimal experience for a given x=education
function matrix exper_0(matrix param, scalar x)
    matrix exper = (-param[3]-param[4]*x)/(2*param[5])
    return exper
end function
```

This looks very similar to the G function. Both evaluate the function $g(\boldsymbol{b})$. The difference lies in the fact that G evaluates the function at each observation and exper_0 only evaluates $g(\boldsymbol{b})$ at a specific point, $x$. Once the function is defined, fdjac operates on it to as prescribed by the delta method.

```
 1 open "@workdir\data\cps5_small.gdt"
 2 set echo off
 3 logs wage
 4 square exper
 5 series educ_exper = educ * exper
 6 
 7 ols l_wage const educ exper educ_exper sq_exper
 8 matrix covmat = $vcv
 9 matrix b = $coeff
10 list ed = educ
11 
12 matrix jac = fdjac(b, G(&b, ed))     # Numerical derivatives at each obs
13 matrix d = meanc(jac)                # The sum of the derivatives = d
14 matrix variance = qform(d,covmat)    # Var = d’ COV d
15 matrix se = sqrt(variance)           # Std Error = sqrt(Var)
16 
17 printf "\nThe optimal experience given %2g years of schooling is =\
18 %.2f\n", 16, exper_0(b,16)
19 printf "\nThe estimated standard error of experience_0 = %.3f\n", se
20 t_interval(exper_0(b,16),se,$df,.95)
```

The main difference in this version of the example lies in lines 12-14. In line 12 `fdjac` is used on the function `G(&b, ed)`. `&b` points to the contents of the current parameter vector `b`, `ed` is a `list` that contains all observations on education. This returns an $n \times 5$ matrix of derivatives. The next line takes the column sums and is the $1 \times 5$ vector `d`. The quadratic form is computed using the `qform(d,covmat)` command. The vector `d` is the first argument and the center of the quadratic form, `covmat`, is the second argument. From there the script looks like its manually calculated predecessor.

A pointer is used to supply the parameter vector to the function `(matrix *param)`. When the function is called, the vector of parameters provided by the user in `param` are held in a specific memory address. The `*` tells **gretl** to hold the contents of `param` in a memory address that can later be recalled. To recall the current contents of that (sometimes referred to as dereferencing) you must use the ampersand (`&`) in front of the `param` matrix being passed to the function, i.e., `G(&param, x)`. Thus, pointers require a pair of markers, `*` and `&`, when used.

Using pointers avoids having to make copies of objects within the program, and whatever is passed around by it can be modified in the process. That may sound like a bad idea, but it makes programs more modular. In the `fdjac` function, pointers allow the numerical derivative to be solved for recursively. See section 13.4 of the Gretl Users Guide (Cottrell and Lucchetti, 2018) for more details.

The script is run and the interval computed using our `t_interval` function. Note that we use the `exper_0` function evaluated at the least squares coefficients and an education level of 16.

```
t_interval(exper_0(b,16),se,$df,.95)
```

The result is:

```
The 95% confidence interval centered at 30.173 is (26.6721, 33.6738)
```

which is the same at shown in *POE5*. The delta method using numercial derivatives appears to have worked as intended.

## 5.7 *POE5* Appendix 5

### 5.7.1 Confidence interval using the delta method

In this example the food expenditure model is estimated via least squares and a nonlinear function of its parameters is computed. The standard errors are estimated via the delta method.

The function estimated is

$$g_1 \equiv g(b_2) = \exp(b_2)/10$$

Evaluated at the estimates $g(b_2) = \exp(b_2)/10 = \exp(10.784/10) = 2.91$. The derivative of $g(b_2)$ is $g(b_2)/10$. The script to estimate the 95% confidence interval that uses these is:

**Example 5.19 in *POE5***

```
open "@workdir\data\mc20.gdt"
ols y const x
scalar g0 = exp($coeff(x)/10)          # Function
scalar d0 = (g0/10)                    # Derivative
scalar se = d0*$stderr(x)              # Delta method std error
t_interval(g0,se,$df,.95)              # Confidence Interval
```

This produces:

```
The 95% confidence interval centered at 2.911 is (1.6006, 4.2212)
```

which matches the values in *POE5*.

### Example 5.20 in *POE5*

In this example the nonlinear function depends on two parameters. The function is

$$g_2 \equiv g(b_1, b_2) = b_1/b_2$$

This requires two derivatives that we refer to as $d_1$ and $d_2$. The following script estimates the model, a 95% confidence interval centered at $g(b_1, b_2) = b_1/b_2$, and a standard error computed via the delta method.

```
open "@workdir\data\mc20.gdt"
ols y const x
matrix covmat = $vcv
scalar g = $coeff(const)/$coeff(x)          # Function
scalar d1 = 1/$coeff(x)                     # Derivative b1
scalar d2 = -$coeff(const)/$coeff(x)^2      # Derivative b2
matrix d = d1 ~ d2                          # Vector d
matrix variance = qform(d,covmat)           # Delta method std error
scalar se = sqrt(variance)                  # Standard Error
t_interval(g,se,$df,.95)                    # Confidence Interval
```

The result is:

```
The 95% confidence interval centered at 8.184 is (-1.8077, 18.1758)
```

which matches *POE5*.

### Monte Carlo: Simulation with $\chi^2$ errors

This simulation is designed to illustrate the repeated sampling properties of least squares. The experimental design is the same at that used in section 3.7.1. In this case, errors are not normally distributed, but generated by a $\chi^2(4)$. The variates are centered at the mean ($E[\chi^2(4)] = 4$). These are normalized by dividing by the standard error, $\sqrt{8}$. The variance of the overall errors is set to $\sigma^2 = 2500$. This appears in line 11. The rest of the script should be familiar. Confidence bounds are computed in lines 19 and 20. A scalar `p1` is computed that takes the value 1 whenever the statement in parenthesis is true, i.e., when $\beta_2 = 10$ falls within the estimated interval; `p2` will be 1 when the test statistic falls within the 0.05 rejection region of the test; and `close` is 1 when $\beta_2$ is between 9 and 10.

The `print` statement in line 31 has the progressive loop compute summary statistics for the listed scalars and the `store` command writes the scalars to a new dataset named *mc_5.1.gdt*.

```
matrix sizes = { 20, 40, 100, 200, 500, 1000}
scalar size = sizes[3]
print size
nulldata size --preserve
    genr index                          # variable for obs number
    series x = (index>size/2) ? 20 : 10  # Create X =10 and X=20
    series ys = 100 + 10*x              # Systematic part of model
    scalar nu = 4                       # Deg-of-freedom for chi-square
    scalar s = 50                       # Standard deviation of errors
loop 10000 --progressive --quiet
    series e =  s * (randgen(c,nu)-nu)/sqrt(2*nu) # Normed Chi-square
    series y = ys + e                    # sample of y
    ols y const x                        # Regression
    scalar b1 = $coeff(const)            # Save intercept
    scalar b2 = $coeff(x)                # Save slope
    scalar s2 = $sigma^2                 # Save sigma-squared

    #Interval bounds
    scalar c2L = $coeff(x) - critical(t,$df,.025)*$stderr(x)
    scalar c2R = $coeff(x) + critical(t,$df,.025)*$stderr(x)

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

    # Compute Rejection of test
    scalar p2 = (($coeff(x)-10)/$stderr(x))>critical(t,$df,.05)

    # Compute whether slope is between 9 and 11.
    scalar close = (9>c2L && 11<c2R)

    print            b1 b2 s2 p1 p2 close
    store mc_5.1.gdt b1 b2 s2 p1 p2 close
endloop
```

The output from the progressive loop is:

```
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       99.7949        15.8964        15.7221        1.73307
         x       10.0113        1.00646       0.994351       0.109609

Statistics for 10000 repetitions

                   mean       std. dev
      b1       99.7949        15.8964
```

```
       b2        10.0113         1.00646
       s2        2501.87         557.302
       p1       0.949400        0.219179
       p2      0.0523000        0.222631
    close       0.661500        0.473199
```

Based on the summary statistics, the average value of $\bar{b}_1 = 99.79$ and of $\bar{b}_2 = 10.01$. Estimated variance averages 2501.87. The confidence interval covers in 9494/10000 times and the true hypothesis rejected in 523/10000 samples. These would be predicted in a linear model with homoscedastic, linearly independent error terms.

Now load the results that were written to *mc_5.1.gdt* and plot the histogram of $b_2$. In the top panel of Figure 5.4 you'll find its histogram plotted along with a normal distribution curve. The histogram appears to be approximately normally distributed (n=100), implying that the asymptotic normal approximation for the least squares coefficient starts at a very modest sample size. In a later example, we examine whether this holds for the delta method approximations.

```
open "@workdir\mc_5.1.gdt"
grb2 <- freq b2 --normal --plot=display
```

### Simulation of the delta method

In this example, we study the performance of the delta method. Using the same design as used in the previous example we also compute functions $g_1 = \exp(b_2/10)$ and $g_2 = b_1/b_2$. The histogram for the function $g_1 = \exp(b_2/10)$ based on 10000 Monte Carlo samples is shown in the bottom of Figure 5.4. The distribution of $g_1$ is skewed to the left, and does not look normally distributed (the Doornik-Hansen test confirms this).

In Figure 5.5 histograms based on 10000 Monte Carlo samples for estimates of $g_2 = b_1/b_2$ are shown for sample sizes of 40 and 200. At $n = 40$ the function is decidedly skewed. As the sample size increases, the statistic is converging towards normality, though it is still badly skewed.

### Monte Carlo: Simulation of the delta method

```
matrix sizes = { 20, 40, 100, 200, 500, 1000}
scalar size = sizes[2]
print size
nulldata size --preserve
    genr index
    series x = (index>size/2) ? 20 : 10
    series ys = 100 + 10*x
    scalar s = 50
    scalar nu = 4
loop 10000 --progressive --quiet
    series e =  s * (randgen(c,nu)-nu)/sqrt(2*nu)
    series y = ys + e
    ols y const x
    scalar b1 = $coeff(const)
    scalar b2 = $coeff(x)
    scalar s2 = $sigma^2
    matrix covmat = $vcv
    # first function
    scalar g1 = exp(b2/10)
    scalar d1 = (g1/10)
    scalar se_g1 = d1*$stderr(x)
    scalar p_g1 = abs((g1-2.71828)/se_g1)>critical(t,$df,.025)
    # second function
    scalar g2 = b1/b2
    scalar d1 = 1/b1
    scalar d2 = -b1/b2^2
    matrix d = d1 ~ d2
    matrix vmat = qform(d,covmat)
    scalar se_g2 = sqrt(vmat)
    scalar c2L = g2 - critical(t,$df,.025)*se_g2
    scalar c2R = g2 + critical(t,$df,.025)*se_g2
    # the coverage probabilities of the Confidence Intervals
    scalar p1_g2 = (10>c2L && 10<c2R)
    scalar p2_g2 = (($coeff(x)-10)/$stderr(x))>critical(t,$df,.05)
    scalar close = (9>c2L && 11<c2R)
    print            g1 se_g1 g2 se_g2 p_g1 p1_g2 p2_g2 close
    store mc_5.2.gdt g1 se_g1 g2 se_g2 p_g1 p1_g2 p2_g2 close
endloop
```

In this example the nonlinear functions are estimated along with their standard errors, coverage probabilities of a 95% confidence interval and the rejection rate of a $t$-test. The results for $n = 40$, $n = 200$, and $n = 1000$ are shown below:

```
Statistics for 10000 repetitions
n=40
                  mean       std. dev
      g1       2.74238       0.428058
   se_g1      0.426886       0.103466
      g2       10.7440        4.50758
   se_g2       4.34602        1.94085
    p_g1     0.0479000       0.213555
   p1_g2      0.949500       0.218974
   p2_g2     0.0442000       0.205539
   close      0.912700       0.282274
```

```
Statistics for 10000 repetitions
n=200
                  mean        std. dev
      g1       2.72402        0.192129
   se_g1      0.192045       0.0203195
      g2       10.1357         1.84695
   se_g2       1.82731        0.311876
    p_g1     0.0499000        0.217738
   p1_g2      0.949400        0.219179
   p2_g2     0.0495000        0.216910
   close      0.846100        0.360853

Statistics for 10000 repetitions
n=1000
                  mean        std. dev
      g1       2.72025       0.0848627
   se_g1     0.0859604      0.00406127
      g2       10.0210        0.798622
   se_g2      0.807781       0.0592096
    p_g1     0.0485000        0.214820
   p1_g2      0.953200        0.211210
   p2_g2     0.0482000        0.214189
   close      0.536900        0.498637
```

In all samples the average values of the two functions are very close to their theoretical values, 2.71828 and 10, though things improve slightly as $n$ increases. The rejection rate for a $\alpha = .05$ $t$-ratio for `g1` is `p_g1`= .0485. The rejection rate for the $t$-ratio associated with `g2` is 0.0482. The 95% confidence interval for $\beta_1/\beta_2$ covers 95.43% of the time in repeated samples.

To view frequency plots of the simulated functions load the results that were written to *mc_5.2.gdt*.

```
open "@workdir\mc_5.2.gdt"
gr1 <- freq g1 --normal --plot=display
gr2 <- freq g2 --normal --plot=display
```

### Bootstrap using the empirical distribution function

In this example, a $t$-statistic is bootstrapped based on the empirical distribution of the model errors. This is achieved by resampling residuals. *POE5* uses a different method for bootstrapping that will be considered in the next section.

Resampling in **gretl** is done using the `resample(x,blocksize)` command. This resamples from $x$ (a series or a matrix) with replacement. In the case of a series argument, each value of the returned series, $x_{boot}$, is drawn from among all the values of $x_t$ with equal probability. When a matrix argument is given, each row of the returned matrix is drawn from the rows of $x$ with equal probability. The `blocksize` argument is optional and is used if you want to resample in data chunks larger than 1 observation.

The first example the model is estimated and bootstrap samples are drawn, with replacement, from the estimated residuals. This amounts to using the empirical distribution of the errors. This happens in line 11. A $t$-ratio is bootstrapped (a pivotal statistic) and stored to an external dataset, *tsim.gdt*, for further analysis. The results are based on 1999 bootstrap samples.

```
1  # Bootstrap using EDF (Residuals)
2  open "@workdir\data\mc20.gdt"
3  ols y const x
4  matrix b=$coeff
5  series u=$uhat
6  series yhat = $yhat
7  scalar replics=1999
8  scalar tcount=0
9  series ysim
10 loop replics --progressive
11     ysim = yhat + resample(u)
12     ols ysim const x --quiet
13     scalar tsim = abs(($coeff(x)-b[2])/$stderr(x))
14     tcount += (tsim>critical(n,.025))
15     print tsim
16     store tsim.gdt tsim
17 endloop
18 printf "Proportion of cases with |t|>2.5 = %g\n", tcount/replics
```

To find the $t$ critical value from the empirical distribution, load *tsim.gdt* and find the desired percentile ( $95^{th}$ in this case).

```
open tsim.gdt
scalar critv=quantile(tsim,.95)
print critv
```

which is 2.0815 in this example.

### Pairwise Bootstrap

In this type of bootstrapping, rows of the entire data are resampled with replacement, so-called $(y_i, x_i)$ pairs. To resample more than 1 variable at a time by observation **gretl** requires conversion of multiple series into a matrix. Resample from the matrix, then disassemble the matrix columns back into series. A script for this example is:

```
open "@workdir\data\mc20.gdt"
scalar n_bootsamples = 1999        # set number of bootstrap samples

ols y const x
scalar beta1=$coeff(const)         # save original coeff b1
scalar beta2=$coeff(x)             # save original coeff b2
scalar g1_beta = exp(beta2/10)     # g1 function at original est
scalar g2_beta = beta1/beta2       # g2 function at original est

list allvars = y const x            # list of all variables
matrix X = { allvars }              # data into matrix for resampling

# start bootstrap loop
loop i=1..n_bootsamples --progressive --quiet
    matrix m1 = resample(X)         # resample rows of variables
    matrix y1 = m1[,1]              # extract dependent var
    matrix x1 = m1[,3]              # extract independent var
    series y =  y1                  # convert data back to series
    series X1 = x1
    ols y const X1                  # run regression
    scalar b2=$coeff(X1)            # save slope & intercept estimates
    scalar b1=$coeff(const)
    matrix covmat = $vcv            # save the covariance estimate
    # first function
    scalar g1 = exp(b2/10)          # first function
    scalar d1 = (g1/10)             # derivative of function
    scalar se_g1 = d1*$stderr(X1)   # delta method se
    scalar bias1 = g1-g1_beta       # bias
    scalar t1 = bias1/se_g1         # t-ratio, Ho true
    # second function
    scalar g2 = b1/b2               # second function
    scalar d1 = 1/b2                # derivative dg/db1
    scalar d2 = -b1/b2^2            # derivative dg/db2
    matrix G = d1 ~ d2              # vector of derivatives
    matrix vmat = G*covmat*G'       # Delta method variance
    scalar se_g2 = sqrt(vmat)       # std error
    scalar bias2 = (g2-g2_beta)     # bias
    scalar t2 = (bias2)/se_g2       # t-ratio, Ho true
    # print and store
    print                  b1 b2 g1 se_g1 g2 se_g2 bias1 bias2 t1 t2
    store bootsample40.gdt b1 b2 g1 se_g1 g2 se_g2 bias1 bias2 t1 t2
endloop
```

Within the loop we compute both functions ($g_1$ and $g_2$), their biases, delta method standard errors, and $t$-ratios. These are stored to a dataset *bootsample40.gdt*.

To analyze results, open the *bootsample40.gdt* and construct the desired statistics as series. The summary statistics reveal the quantities of interest.

```
open bootsample20.gdt
summary
# freq b2 --normal --plot=display
summary bias1 bias2  --simple
summary g1 g2 --simple
scalar q_025 = quantile(g1,.025)
scalar q_975 = quantile(g1,.975)
scalar c_t1_05 = quantile(abs(t1),.95)
print q_025 q_975 c_t1_05
```

Here we take summary statistics for the bias and the functions. The 0.025 and 0.975 quantiles are taken of $g_1$ to obtain a percentile bootstrap confidence interval. Finally, the 0.95 quantile of the $t$-ratio is taken to reveal the bootstrap critical value for the $t$-test based on the sample size.

The results are:

```
                  Mean      Median        S.D.         Min         Max
bias1          0.05813    -0.04279      0.6544      -1.382       7.135
bias2           0.8788      0.2036       4.489      -10.18       34.22

                  Mean      Median        S.D.         Min         Max
g1               2.969       2.868      0.6544       1.529       10.05
g2               9.063       8.388       4.489      -1.995       42.41

          q_025 =  1.9977185
          q_975 =  4.4955391
        c_t1_05 =  3.1386137
```

The *mc20.gdt* dataset is based on samples of size 20. So, the first rows of Tables 5D.4b and 5D.5 are the ones we use for comparison. In fact, the biases are quite close. Our bootstrap bias measure for $g_1$ measured 0.058 and *POE5*'s measured 0.068. For $g_2$ ours measures 0.88 and *POE5*'s measured 0.79. The 95% confidence interval is $(1.998, 4.496)$ and the 5% $t$ critical value for sample size 20 is 3.14. The bootstrap standard error is 4.489 for $g_2$ which is slightly larger than that in *POE5*, which is 4.442. This accounts for the slightly larger $t$ critical value and confidence intervals produced from our script.

## 5.8 waldTest

Finally,[3] since the delta method received quite a bit of attention in this chapter, it is worth mentioning the user written package, *waldTest.gfn* that is available on the **gretl** function package

[3] This section is optional and uses a **gretl** add-on from its function package server. The server and its use is discussed later is used extensively in Chapter 16 below and is discussed in section 16.3.3.

server (see section 16.3.3 for some details on what this contains and how to use it). This function package was written by Oleh Komashko and is able to test nonlinear hypotheses in just about any **gretl** estimated model. It also can be used to estimate confidence intervals for nonlinear functions as well. For the preceding example we could use:

```
1 include waldTest                 # grab this package from the server
2 open "@workdir\data\andy.gdt"
3 square advert
4 ols sales const price advert sq_advert
5
6 nlwaldtest("(1-b[3])/(2*b[4])",$coeff,$vcv)
7 nlconfint("(1-b[3])/(2*b[4])",$coeff,$vcv,null,.95,$df)
```

The first step is to download and install the *waldTest* function package from the **gretl** function package server. This process is described in section 16.3.3. Then, open the data, create the square of advertising and estimate the linear regression as in line 4. Two of the functions from *waldTest* are shown in lines 6 and 7. The syntax is realtively forgiving, but consult the help that comes with the function package for guidance if the function returns something unexpected. In the first instance, `nlwaldtest` computes a nonlinear wald test using the delta method. The first argument, which is enclosed in double quotes, is the expression for the nonlinear combination of parameters you want to test. In this case, $(1 - \beta_3)/2\beta_4 = 0$. The next arguments are the coefficient vector from the estimated model, and the estimated variance-covariance matrix. Additional options can be added to the argument list. For instance, you can specify either the chi-square or $F$ form of the Wald test (see section 6.1.3).

The second command estimates a confidence interval centered at the nonlinear combination, again using the delta method to obtain a standard error. This command uses five inputs. As in `nlwaldtest`, the first two are the coefficient vector and the variance-covariance matrix. The next argument is for the variable list (normally `$xlist`) which in this case is set to `null`, meaning that we give no value for it. Next is the desired coverage probability of the confidence interval, and the last is the relevant degrees of freedom to use for the $t$-distribution.

The output produced by these nifty functions is:

```
Wald test of a (non)linear restriction:

(1-b[3])/(2*b[4]) = 0

Chi(1) = 244.88, with p-value = 3.39431e-055


Confidence interval for function of model parameters:
       (1-b[3])/(2*b[4])

   t(71, 0.025) = 1.994
```

```
  value     std. err        95% conf. interval

2.01434     0.128723      1.75767      2.27101
```

The confidence interval matches the results obtained manually in Example 5.17 above.

## 5.9 Script

```
set verbose off

# function estimates confidence intervals based on the t-distribution
function void t_interval(scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# Example 5.1
open "@workdir\data\andy.gdt"
#Change the descriptive labels and graph labels
setinfo sales --description="Monthly sales revenue ($1000)" \
              --graph-name="Monthly Sales ($1000)"
setinfo price --description="Price in dollars" --graph-name="Price"
setinfo advert --description="Monthly Advertising Expenditure ($1000)" \
               --graph-name="Monthly Advertising ($1000)"
# print the new labels to the screen
labels

# summary statistics
summary sales price advert --simple

# Example 5.2
# regression, prediction, variable rescaling
m1<-ols sales const price advert

# Example 5.2
# Predict sales when price is 5.50 and adv is 1200
scalar yhat = $coeff(const) + $coeff(price)*5.50 + $coeff(advert)*1.2
printf "\nPredicted sales when price=$5.50 and advertising=$1200 is\
$%.2f\n", yhat*1000

# Rescale variables
series sales_star = sales * 1000
series price_star = price * 100
ols sales_star const price_star advert

# Example 5.3
# Calculate sigma-hat square
open "@workdir\data\andy.gdt"
list xvars = const price advert
ols sales xvars
scalar sighat2 = $ess/$df
scalar sig2 = $sigma^2
print sighat2 sig2
```

```

# eval function
eval($sigma^2)

# Example 5.4
# Goodness-of-fit
printf "\nR-square = %.3f\n", $rsq

# FWL
open "@workdir\data\andy.gdt"
list xvars = const price advert
ols sales xvars
series ehat=$uhat
printf "\nSum-of-Squared Errors from full regression: %.3f\n", $ess

ols sales const price --quiet
series sales_resid=$uhat
ols advert const price --quiet
series advert_resid=$uhat
ols sales_resid advert_resid
series ehat_fwl=$uhat

smpl 1 5
print ehat_fwl ehat
printf "\nSum-of-Squared Errors from FWL: %.3f\n", $ess
smpl full

# Example 5.5
ols sales const price advert --vcv
matrix covmat = $vcv
matrix se = sqrt(diag(covmat))
printf "Least Squares standard errors:\n%.3f\n", se
/*---POE5 Example 5.6---*/
open "@workdir\data\andy.gdt"
m1 <- ols sales const price advert
t_interval($coeff(price),$stderr(price),$df,.95)
t_interval($coeff(advert),$stderr(advert),$df,.95)

/*---POE5 Example 5.7---*/
# 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)*$vcv[2,3])
t_interval(chg,se_chg,$df,.95)

# Examples 5.8 and 5.9
# 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

scalar t3 = ($coeff(advert)-1)/$stderr(advert)
pvalue t $df t1
scalar p=pvalue( t, $df, t3)
print p

/*---POE5 Example 5.10---*/
scalar t = ($coeff(price)-0)/$stderr(price)
scalar crit = -critical(t,$df,0.05)
scalar pval = 1-pvalue(t,$df,t)
printf "\n Ho: b2=0 vs Ha: b2<0 \n \
the t-ratio is = %.3f. \n \
the critical value = %.3f \n \
and the p-value = %.3f\n", t, crit, pval

/*---POE5 Example 5.11---*/
scalar t = ($coeff(advert)-1)/$stderr(advert)
scalar crit = critical(t,$df,0.05)
scalar pval = pvalue(t,$df,t)
printf "\n Ho: b3=1 vs Ha: b3>1 \n \
the t-ratio is = %.3f \n \
the critical value = %.3f \n \
and the p-value = %.3f\n", t, crit, pval

# Example 5.12
# 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])

printf "\n Ho: d=-0.2b2-0.5b3=0 vs Ha: d > 0 \n \
the t-ratio is = %.3f \n \
the critical value = %.3f \n \
and the p-value = %.3f\n", \
chg/se_chg, critical(t,$df,0.05), pvalue(t,$df,chg/se_chg)

# Using matrices to compute linear combination variance
ols sales const price advert
covmat = $vcv
d = { 0; -0.2; -0.5 }
covest = d'*covmat*d
se = sqrt(covest)
printf "\nThe estimated standard error of the linear combination is\
%.4f\n", se

# Example 5.14
# interaction creates nonlinearity
```

```
open "@workdir\data\andy.gdt"
series a2 = advert*advert
square a2
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 "\n The marginal effect at \$500 (advert=.5) is %.3f\n\
and at \$2000 is %.3f\n",me1,me2

# Example 5.15
open "@workdir\data\cps5_small.gdt"
series educ_exper = educ*exper
ols wage const educ exper educ_exper

scalar me_8year =  $coeff(exper)+$coeff(educ_exper)*8
scalar me_16year = $coeff(exper)+$coeff(educ_exper)*16
scalar me_20year = $coeff(exper)+$coeff(educ_exper)*20
scalar me_ed_20exper = $coeff(educ)+$coeff(educ_exper)*20

set echo off
printf "\nMarginal effect of another year of schooling when:\n\
experience is 0 = %.3f\n\
experience is 20 = %.3f\n", $coeff(educ), me_ed_20exper
printf "\nMarginal effect of experience when:\n\
education is 8 = %.3f \n\
education is 16 = %.3f \n\
education is 20 = %.3f \n", me_8year, me_16year, me_20year

# Example 5.16
open "@workdir\data\cps5_small.gdt"
logs wage
square exper
series educ_exper = educ * exper

ols l_wage const educ exper educ_exper sq_exper

function void me_1(list vars "all variables, including dep var first",
      scalar ed "set years of schooling",
      scalar expr "set years of experience")
    ols vars --quiet
    scalar me = $coeff(exper) + $coeff(educ_exper)*ed +\
      2*$coeff(sq_exper)*expr
    printf "\nMarginal effect of another year of experience:\n \
Education = %.3f years and Experience = %.3f years\n \
Marginal effect is %.3f percent \n", ed, expr, me*100
end function
list regression = l_wage const educ exper educ_exper sq_exper

me_1(regression,  8, 0)
me_1(regression, 16, 0)
me_1(regression,  8, 20)
```

```
me_1(regression, 16, 20)

function void me_2(list vars "all variables, including dep var first",
      scalar ed "set years of schooling",
      scalar expr "set years of experience")
    ols vars --quiet
    scalar mw = $coeff(educ) + $coeff(educ_exper)*expr
    printf "\nMarginal effect of another year of schooling:\n \
 Education = %.3g years and Experience = %.3g years\n \
 Marginal effect is %.3f percent \n", ed, expr, mw*100
end function

list regression = l_wage const educ exper educ_exper sq_exper

me_2(regression,  8, 0)
me_2(regression, 16, 0)
me_2(regression,  8, 20)
me_2(regression, 16, 20)

# Example 5.17
# delta method for nonlinear hypotheses
open "@workdir\data\andy.gdt"
square advert

ols sales const price advert sq_advert --vcv
matrix b = $coeff
matrix cov = $vcv
scalar g_beta = (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 = g_beta - critical(t,$df,.025)*se
scalar ub = g_beta + critical(t,$df,.025)*se
printf "\nThe estimated optimal level of advertising is $%.2f.\n",\
    1000*g_beta
printf "\nThe 95%% confidence interval is ($%.2f, $%.2f).\n",\
    1000*lb,1000*ub

t_interval(g_beta,se,$df,.95)

# Bonus: waldTest.gfn   # first, install package from the funct server
include waldTest        # once installed, this will work
open "@workdir\data\andy.gdt"
square advert
ols sales const price advert sq_advert

nlwaldtest("(1-b[3])/(2*b[4])",$coeff,$vcv)
nlconfint("(1-b[3])/(2*b[4])",$coeff,$vcv,null,.95,$df)
```

```

# Example 5.18 Optimal Experience--the delta method
# Function computes the optimal experience for a given x=education
function matrix exper_0(matrix param, scalar x)
    matrix exper = (-param[3]-param[4]*x)/(2*param[5])
    return exper
end function

# This function computes experience for all observations in sample
function matrix G(matrix *param, list x)
    matrix X = { x }
    matrix r1 = (-param[3].*ones($nobs,1)- param[4]*X)./(2*param[5])
    return r1
end function

open "@workdir\data\cps5_small.gdt"
set echo off
logs wage
square exper
series educ_exper = educ * exper

ols l_wage const educ exper educ_exper sq_exper
matrix covmat = $vcv
matrix b = $coeff
list ed = educ

matrix jac = fdjac(b, G(&b, ed))       # Numerical derivatives at each obs
matrix d = meanc(jac)                  # The sum of the derivatives = d
matrix variance = qform(d,covmat)      # Var = d' COV d
matrix se = sqrt(variance)             # Std Error = sqrt(Var)

printf "\nThe optimal experience given %2g years of schooling is = %.2f\n",\
     16, exper_0(b,16)
printf "\nThe estimated standard error of experience_0 = %.3f\n", se
t_interval(exper_0(b,16),se,$df,.95)

# Example 5.19
open "@workdir\data\mc20.gdt"
ols y const x
scalar g0 = exp($coeff(x)/10)          # Function
scalar d0 = (g0/10)                    # Derivative
scalar se = d0*$stderr(x)              # Delta method std error
t_interval(g0,se,$df,.95)              # Confidence Interval

# Example 5.20
open "@workdir\data\mc20.gdt"
ols y const x
matrix covmat = $vcv
scalar g = $coeff(const)/$coeff(x)        # Function
scalar d1 = 1/$coeff(x)                   # Derivative b1
scalar d2 = -$coeff(const)/$coeff(x)^2    # Derivative b2
```

```
matrix d = d1 ~ d2                        # Vector d
matrix variance = qform(d,covmat)         # Delta method std error
scalar se = sqrt(variance)                # Standard Error
t_interval(g,se,$df,.95)                  # Confidence Interval

# Monte Carlo simulation of linear model with chi-square errors
matrix sizes = { 20, 40, 100, 200, 500, 1000}
scalar size = sizes[3]
print size
nulldata size --preserve
    genr index                           # Generate index for obs numbers
    series x = (index>size/2) ? 20 : 10  # Create X =10 and X=20
    series ys = 100 + 10*x               # Systematic part of model
    scalar nu = 4                        # Degrees of freedom for chi-square
    scalar s = 50                        # Standard deviation of errors
loop 10000 --progressive --quiet
    series e =  s * (randgen(c,nu)-nu)/sqrt(2*nu) # Normalized Chi-square rv
    series y = ys + e                     # sample of y
    ols y const x                         # Regression
    scalar b1 = $coeff(const)             # Save intercept
    scalar b2 = $coeff(x)                 # Save slope
    scalar s2 = $sigma^2                  # Save sigma-squared
    #Interval bounds
    scalar c2L = $coeff(x) - critical(t,$df,.025)*$stderr(x)
    scalar c2R = $coeff(x) + critical(t,$df,.025)*$stderr(x)
    # Compute the coverage probabilities of the Confidence Intervals
    scalar p1 = (10>c2L && 10<c2R)
    # Compute Rejection of test
    scalar p2 = (($coeff(x)-10)/$stderr(x))>critical(t,$df,.05)
    # Compute whether slope is between 9 and 11.
    scalar close = (9>c2L && 11<c2R)
    print b1 b2 s2 p1 p2 close
    store mc_5.1.gdt b1 b2 s2 p1 p2 close
endloop

open "@workdir\mc_5.1.gdt"
grb2 <- freq b2 --normal --plot=display

# Monte Carlo simulation of delta method
matrix sizes = { 20, 40, 100, 200, 500, 1000}
scalar size = sizes[4]
print size
nulldata size --preserve
    genr index
    series x = (index>size/2) ? 20 : 10
    series ys = 100 + 10*x
    scalar s = 50
    scalar nu = 4
loop 10000 --progressive --quiet
    series e =  s * (randgen(c,nu)-nu)/sqrt(2*nu)
    series y = ys + e
    ols y const x
    scalar b1 = $coeff(const)
    scalar b2 = $coeff(x)
    scalar s2 = $sigma^2
    matrix covmat = $vcv
    # first function
    scalar g1 = exp(b2/10)
    scalar d1 = (g1/10)
    scalar se_g1 = d1*$stderr(x)
    scalar p_g1 = abs((g1-2.71828)/se_g1)>critical(t,$df,.025)
    # second function
    scalar g2 = b1/b2
    scalar d1 = 1/b2
    scalar d2 = -b1/b2^2
    matrix d = d1 ~ d2
    matrix vmat = qform(d,covmat)
    scalar se_g2 = sqrt(vmat)
    scalar c2L = g2 - critical(t,$df,.025)*se_g2
    scalar c2R = g2 + critical(t,$df,.025)*se_g2
    # the coverage probabilities of the Confidence Intervals
    scalar p1_g2 = (10>c2L && 10<c2R)
    scalar p2_g2 = (($coeff(x)-10)/$stderr(x))>critical(t,$df,.05)
    scalar close = (9>c2L && 11<c2R)
    print            g1 se_g1 g2 se_g2 p_g1 p1_g2 p2_g2 close
    store mc_5.2.gdt g1 se_g1 g2 se_g2 p_g1 p1_g2 p2_g2 close
endloop

open "@workdir\mc_5.2.gdt"
gr1 <- freq g1 --normal --plot=display
gr2 <- freq g2 --normal --plot=display

# Use large df to approximate N(0,1) intervals
open "@workdir\data\mc20.gdt"
ols y const x
scalar g0 = exp($coeff(x)/10)
scalar d0 = (g0/10)
scalar se = d0*$stderr(x)
printf "\nco is %.3f, and se is %.3f\n", g0, se
t_interval(g0,se,120000,.95)

# Bootstrap using EDF (Residuals)
open "@workdir\data\mc20.gdt"
ols y const x
matrix b=$coeff
series u=$uhat
series yhat = $yhat
scalar replics=1999
scalar tcount=0
series ysim
loop replics --progressive
    ysim = yhat + resample(u)
    ols ysim const x --quiet
    scalar tsim = abs(($coeff(x)-b[2])/$stderr(x))
    tcount += (tsim>critical(n,.025))
    print tsim
    store tsim.gdt tsim
endloop
printf "Proportion of cases with |t|>2.5 = %g\n", tcount/replics

open tsim.gdt
scalar critv=quantile(tsim,.95)
print critv

# Pairwise Bootstrap
open "@workdir\data\mc20.gdt"
scalar n_bootsamples = 1999        # set number of bootstrap samples

ols y const x
scalar beta1=$coeff(const)         # save original coeff b1
scalar beta2=$coeff(x)             # save original coeff b2
scalar g1_beta = exp(beta2/10)     # g1 function at original est
scalar g2_beta = beta1/beta2       # g2 function at original est

list allvars = y const x            # list of all variables
matrix X = { allvars }              # put data into matrix for resampling

# start bootstrap loop
loop i=1..n_bootsamples --progressive --quiet
    matrix m1 = resample(X)         # resample rows of variables
    matrix y1 = m1[,1]              # extract dependent var
    matrix x1 = m1[,3]              # extract independent var
    series y =  y1                  # convert data back to series
    series X1 = x1
    ols y const X1                  # run regression
    scalar b2=$coeff(X1)            # save slope and intercept estimates
    scalar b1=$coeff(const)
    matrix covmat = $vcv            # save the covariance estimate
    # first function
    scalar g1 = exp(b2/10)          # first function
    scalar d1 = (g1/10)             # derivative of function
    scalar se_g1 = d1*$stderr(X1)   # delta method se
    scalar bias1 = g1-g1_beta       # bias
    scalar t1 = bias1/se_g1         # t-ratio, Ho true
    # second function
    scalar g2 = b1/b2               # second function
    scalar d1 = 1/b1                # derivative dg/db1
    scalar d2 = -b1/b2^2            # derivative dg/db2
    matrix G = d1 ~ d2              # vector of derivatives
    matrix vmat = G*covmat*G’       # Delta method variance
    scalar se_g2 = sqrt(vmat)       # std error
    scalar bias2 = (g2-g2_beta)     # bias
    scalar t2 = (bias2)/se_g2       # t-ratio, Ho true
    # print and store
    print                  b1 b2 g1 se_g1 g2 se_g2 bias1 bias2 t1 t2
    store bootsample40.gdt b1 b2 g1 se_g1 g2 se_g2 bias1 bias2 t1 t2
endloop

open bootsample40.gdt
summary
# freq b2 --normal --plot=display
summary bias1 bias2  --simple
summary g1 g2 --simple
scalar q_025 = quantile(g1,.025)
scalar q_975 = quantile(g1,.975)
scalar c_t1_05 = quantile(abs(t1),.95)
print q_025 q_975 c_t1_05
```

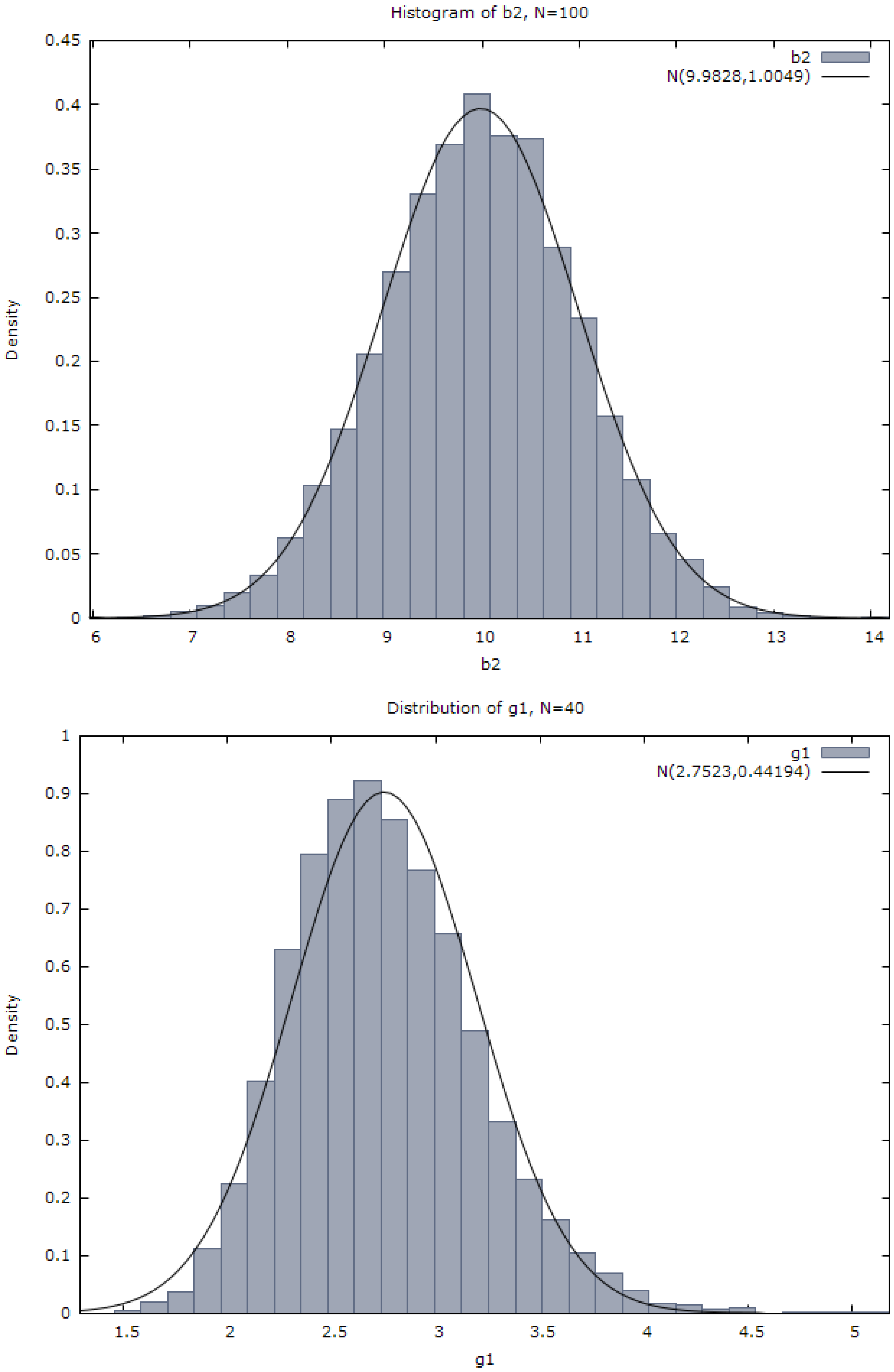


Figure 5.4: Histogram of estimates of $b_2$ for $n = 100$ and $g_1$ for $n = 40$. 10000 Monte Carlo samples.

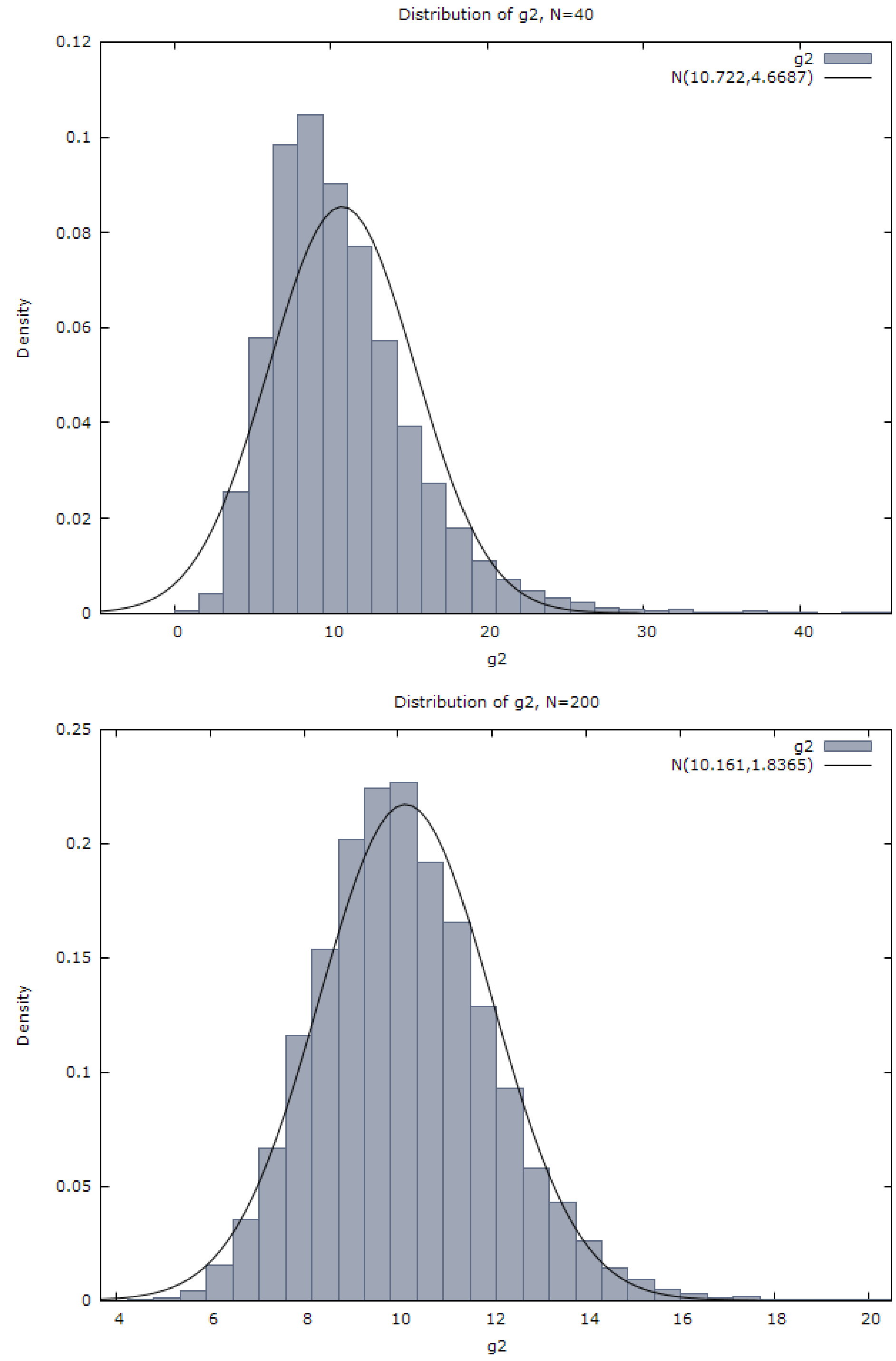


Figure 5.5: Histogram of estimates $g_2$ for $n = 40$ and $n = 200$. 10000 Monte Carlo samples.

# 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. Model specification is considered using model selection rules, out-of-sample forecasting, and a test for functional form. Collinearity and the detection of influential observations are discussed and nonlinear least squares is introduced.

## 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

$$sales = \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 + e \tag{6.1}$$

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.

$$\begin{aligned} H_0 \;\; E[sales|price] &= \beta_1 + \beta_2 price \\ H_1 \;\; E[sales|price, advert] &= \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 \end{aligned}$$

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.

### Examples 6.1 and 6.2 in *POE5*

The script to estimate the models under $H_0$ and $H_1$ and to compute the test statistic is given below.

```
1 open "@workdir\data\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 `square` command is used to square any variable or variables that follow. The string `sq_` is appended as a prefix to the original variable name, so that squared advertising ($\texttt{advert}^2$) 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 4 `scalar sseu = $ess`). In line 5 the degrees of freedom in the unrestricted model are saved so that you can use it in the computation of the $p$-value for the $F$-statistic. 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, estimate the model using least squares. From the pull-down menu (see Figure 2.6) select **Model>Ordinary Least Squares**, specify the unrestricted model (Figure 2.7), and run the regression. This opens the models window (Figure 2.9).

Choose **Tests** from the menu bar of the **models** window, to open the fly-out menu shown in Figure 6.1. The first four options in 6.1 are the most pertinent to the discussion here. These

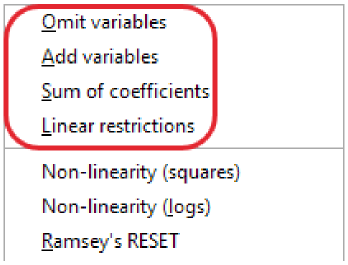


Figure 6.1: Choosing **Tests** from the pull-down menu of the model window reveals several testing options

allow one to test hypotheses by omitting variables from the model, adding variables to the model, summing coefficients, or by imposing arbitrary linear restrictions on the parameters of the model.

Since the test in this example involves imposing a zero restrictions on the coefficients of advertising and squared advertising, we can use the **Omit variables** option. This brings up the dialog box shown in Figure 6.2.

Notice the two radio buttons at the bottom of the window. The first is labeled **Estimate reduced model**; choose this one to compute equation 6.2. If you select the **Wald**, no harm is done. Both are computed using a Wald statistic. The advantage of the Wald test is that a 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 printed and the unrestricted model remains in **gretl**'s memory where its statistics can be accessed.

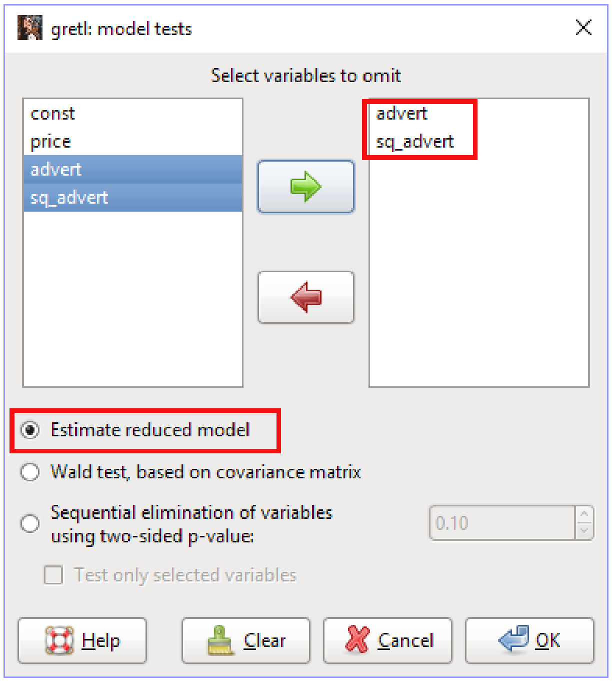
Select the variable *advert* and *sq_advert* as shown. Click **OK** to reveal the result shown in Figure 6.3.

From a script use

```
ols sales const price advert sq_advert
omit advert sq_advert --test-only
```

The `--test-only` option of the `omit` statement will produce the test statistic and $p$-value only, suppressing the printed output from the restricted model to the screen.

The `linear restrictions` option can also be summoned from the pull-down menu as shown



gretl: model tests

Select variables to omit

const
price
advert
sq_advert

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

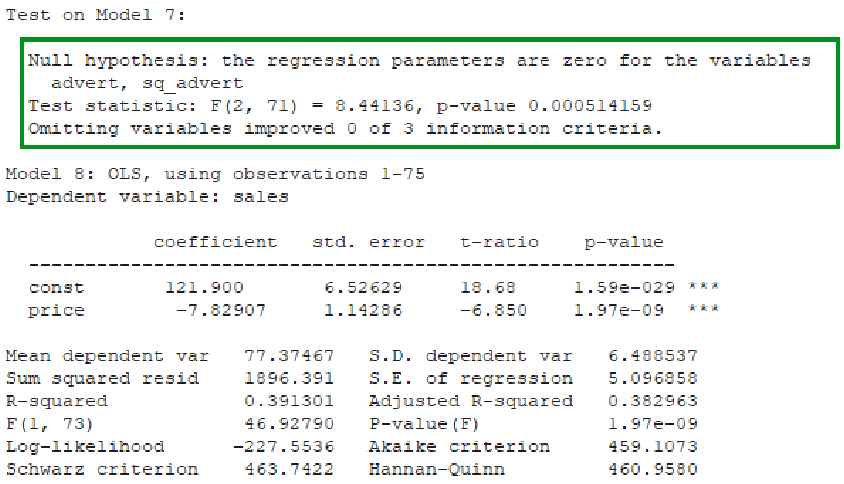
Figure 6.2: The **model tests** dialog box for using omit variables to test zero hypotheses using the fly-out menu in the models window.

in Figure 6.1. This produces a large dialog box that deserves explanation. The box appears in Figure 6.4.

Enter the hypotheses to test (or restrictions to impose) 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 independent variable list, starting with 1. This means that $\beta_3$ 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. When a restriction involves a multiple of a parameter e.g., $3\beta_3 = 2$, place the multiplier first, then the parameter, and use * to multiply. In this case the restriction $3\beta_3 = 2$ is expressed as `3*b[3] = 2`.

From the console or a script you must indicate where the restrictions start and end. The restrictions start with a `restrict` statement and end with `end restrict`. The restrict statement usage is:

```
restrict --quiet
     b[3] = 0
     b[4] = 0
end restrict
```



```
Test on Model 7:

Null hypothesis: the regression parameters are zero for the variables
  advert, sq_advert
Test statistic: F(2, 71) = 8.44136, p-value 0.000514159
Omitting variables improved 0 of 3 information criteria.

Model 8: 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
```

Figure 6.3: The results using the **Omit variables** dialog box to test zero restrictions on the parameters of a linear model.

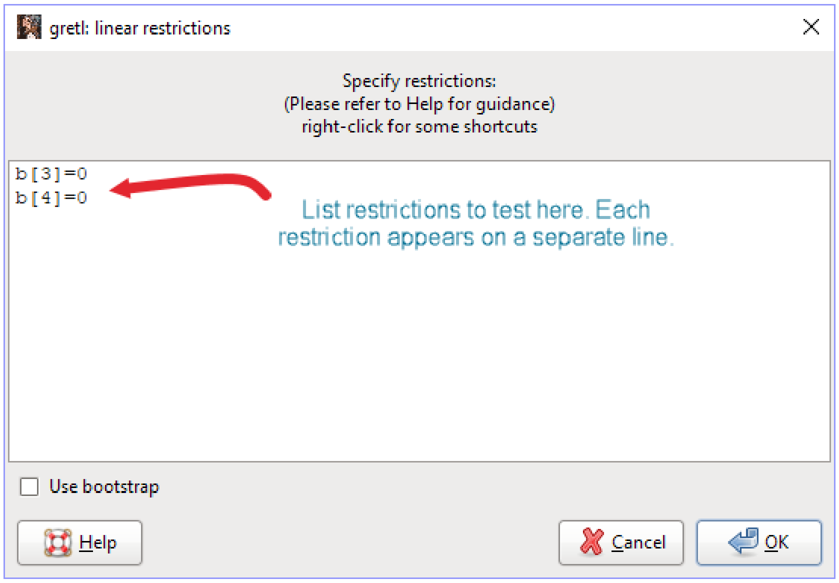
Put each restriction on its own line. The `--quiet` option suppresses the restricted regression from the results window.

Another example of a set of restrictions from a **gretl** script is:

```
restrict
    b[1] = 0
    b[2] - b[3] = 0
    b[4] + 2*b[5] = 1
end restrict
```

The `restrict` and `end restrict` statements can omitted when using the dialog box (Figure 6.4) to impose or test restrictions. The results from the `restrict` statements appear below.

```
    m1 saved
    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  ***
```



gretl: linear restrictions
Specify restrictions:
(Please refer to Help for guidance)
right-click for some shortcuts
List restrictions to test here. Each restriction appears on a separate line.
□ Use bootstrap
Help
Cancel
OK

```
b[3]=0
b[4]=0
```

Figure 6.4: The **linear restriction** dialog box obtained using the **Linear restrictions** option in the **Tests** pull-down menu.

```
advert          0.000000      0.000000      NA          NA
sq_advert       0.000000      0.000000      NA          NA

Standard error of the regression = 5.09686
```

Notice that the restricted estimates are printed; the coefficients on `advert` and `sq_advert` are zero. Use the `--quiet` option in the `restrict` line to suppress the restricted estimates. One disadvantage of using `restrict` is that there is currently no way to assign the output from the restricted model to a session icon. This is something that `omit` allows.

### 6.1.1 Regression Significance

#### Example 6.3 in *POE5*

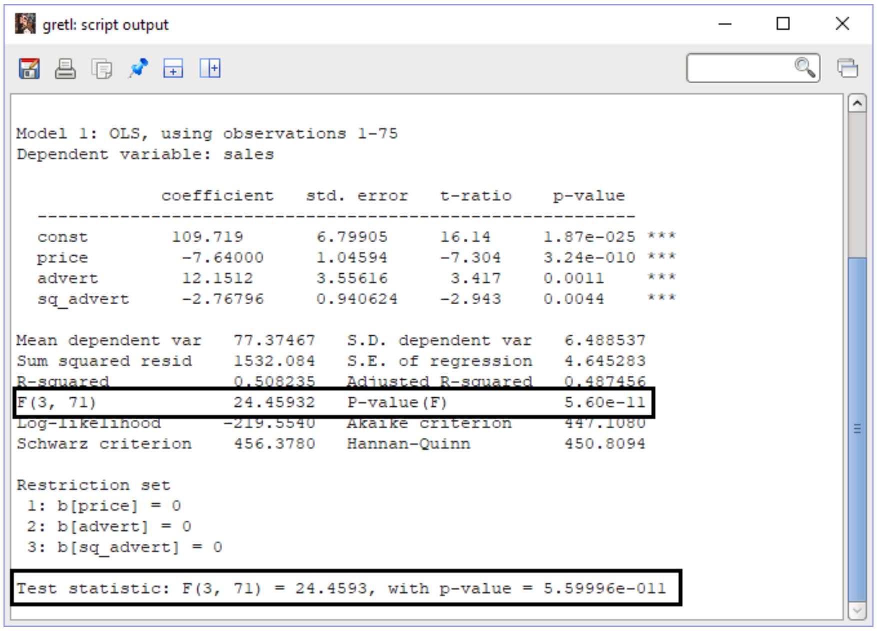
The $F$-statistic is used to statistically determine whether the variables in a model have any effect on the average value of the dependent variable. In this case, $H_0$ is the proposition that $y$ does not depend on any of the independent variables, and $H_1$ is that it does.

$$\begin{aligned} H_o: &\quad E[y_i] = \beta_1 \\ H_1: &\quad E[y_i|x_{i2},\cdots,x_{ik}] = \beta_1 + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} \end{aligned}$$

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

```
1 open "@workdir\data\andy.gdt"
2 square advert
3 ols sales const price advert sq_advert
4 restrict --quiet
5   b[2] = 0
6   b[3] = 0
7   b[4] = 0
8 end restrict
```

In lines 3 the model is estimated and in 4-8 each of the slopes is restricted to be zero. The test result is shown in Figure 6.5 below. You can see that the $F$-statistic for this test is equal to 24.4593.



```
Model 1: 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

Restriction set
 1: b[price] = 0
 2: b[advert] = 0
 3: b[sq_advert] = 0

Test statistic: F(3, 71) = 24.4593, with p-value = 5.59996e-011
```

Figure 6.5: The results obtained from using the `restrict` statements via the dialog box to conduct the overall $F$-test of regression significance.

The same number appears in the regression results as $F(3, 71)$. This is no coincidence. The test of overall 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 6.5. Since the $p$-value is less than $=0.05$, we reject the null hypothesis that the model is insignificant at the five percent level.

The command reference for `restrict` is:

```
restrict

Options:   --quiet (don't print restricted estimates)
           --silent (don't print anything)
```

```
            --wald (system estimators only  see below)
            --bootstrap (bootstrap the test if possible)
            --full (OLS and VECMs only, restricts to last model)

  Imposes a set of (usually linear) restrictions on either (a)
  the model last estimated or (b) a system of equations previously
  defined and named. In all cases the set of restrictions should be
  started with the keyword ``restrict" and terminated with ``end restrict".
```

**Omit** This is a good opportunity to use the omit statement and to show the effect of the --test-only and --chi-square options. Consider the script

```
1 open "@workdir\data\andy.gdt"
2 square advert
3 list xvars = price advert sq_advert
4 ols sales const xvars --quiet
5 omit xvars
6 ols sales const xvars --quiet
7 omit xvars --chi-square
8 ols sales const xvars --quiet
9 omit xvars --test-only
```

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 unrestricted regression is estimated in lines 4, 6 and 8. The first instance of omit in line 5 returns the restricted model and uses the $F$ version of the test statistic. The second omit xvars statement repeats the test, imposing the restrictions on the model, but using the $\chi^2$ version of the test statistic. By default, the omit command replaces the current model in memory with the restricted one. To keep the unrestricted model in memory, and thus its statistics available using accessors, use the --test-only option as in line 9. The output from the three forms is shown below.

```
omit xvars
  Null hypothesis: the regression parameters are zero for the variables
    price, advert, sq_advert
  Test statistic: F(3, 71) = 24.4593, p-value 5.59996e-011
  Omitting variables improved 0 of 3 information criteria.

omit xvars --chi-square
  Null hypothesis: the regression parameters are zero for the variables
    price, advert, sq_advert
  Wald test: Chi-square(3) = 73.3779, p-value 8.06688e-016
  (LR test: Chi-square(3) = 53.2316, p-value 1.63633e-011)
```

```
    Omitting variables improved 0 of 3 information criteria.

  omit xvars --test-only
    Null hypothesis: the regression parameters are zero for the variables
      price, advert, sq_advert
    Test statistic: F(3, 71) = 24.4593, p-value 5.59996e-011
```

The three sets of results are nearly identical. The one difference is that the `--test-only` option offers no information about whether omitting variables improves any of the information criteria (AIC, or SC). The `--test-only` option produces no regression output since a restricted model is not estimated. Finally, statistics from the unrestricted regression are available using the accessors. The regression output was suppressed using the `--quiet` option with the `ols` command.

Without the `--quiet` option, the model is restricted and the estimate of the constant (the series mean in this case) is given before printing the test result.

A summary of the `omit` syntax is given:

```
omit

Argument:   varlist
Options:    --test-only (don't replace the current model)
            --chi-square (give chi-square form of Wald test)
            --quiet (print only the basic test result)
            --silent (don't print anything)
            --vcv (print covariance matrix for reduced model)
            --auto[=alpha] (sequential elimination, see below)
Examples:   omit 5 7 9
            omit seasonals --quiet
            omit --auto
            omit --auto=0.05
```

### 6.1.2 Relationship Between *t*- and *F*-tests

#### Example 6.4 in *POE5*

Using the model for Big Andy

$$sales = \beta_1 + \beta_2 price + \beta_3 advert + \beta_4 advert^2 + e \tag{6.3}$$

and suppose we want to test whether *price* affects *sales*. Using the `omit` command produces the *F*-test and saves the computed statistic to a scalar I call `F_test` using the `$test` accessor.

```
ols sales const price advert sq_advert
omit price --test-only
scalar F_test = $test
```

The output is shown below:

```
Test on Model 2:

  Null hypothesis: the regression parameter is zero for price
  Test statistic: F(1, 71) = 53.3549, p-value 3.23648e-010
```

The $F(1,\ 71)$ statistic is equal to 53.355 and has a $p$-value that is much smaller than 0.05; the coefficient is significant at the 5% level. Compare these results to that of a $t$-test that have been squared.

```
scalar t_2 = ($coeff(price)/$stderr(price))^2
print t_2 F_test
```

This yields:

```
           t_2 =  53.354875
        F_test =  53.354875
```

This confirms that $t^2_{n-k} = F_{1,n-k}$ and therefore the $t$-ratio and the $F$-test must produce identical answers. For two-sided tests, the $p$-values will be equivalent as well.

### 6.1.3 Optimal Level of Advertising

#### Example 6.5 in *POE5*

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 "@workdir\data\andy.gdt"
square advert
ols sales const price advert sq_advert
scalar Ao =(1-$coeff(advert))/(2*$coeff(sq_advert))
printf "\nThe optimal level of advertising is $%.2f\n", Ao*1000
```

which generates the result:

```
The optimal level of advertising is $2014.34
```

To test the hypothesis that $1900 is optimal (remember, *advert* is measured in $1000) based on equation (6.4).

$$\begin{aligned} H_0&: \quad \beta_3 + 3.8\beta_4 = 1 \\ H_1&: \quad \beta_3 + 3.8\beta_4 \neq 1 \end{aligned}$$

you can use a $t$-test or an $F$-test. Following the regression, use

```
restrict --quiet
    b[3]+3.8*b[4]=1
end restrict
```

Remember that `b[3]` refers to the coefficient of the third variable in the regression (`advert`) and `b[4]` to the fourth (`sq_advert`). A coefficient can also be referred to by its variable name. So, the following statement is equivalent:

```
restrict --quiet
    b[advert]+3.8*b[sq_advert]=1
end restrict
```

This is an attractive option since one does not have to keep track of the variable number in the variable list. The disadvantage is that it requires more typing.

The output from either version of the script is:

```
Restriction:
 b[advert] + 3.8*b[sq_advert] = 1

Test statistic: F(1, 71) = 0.936195, with p-value = 0.336543
```

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.

### Example 6.6 in *POE5*

A one-tailed test is a better option in this case. Andy decides he wants to test whether the optimal amount is greater than \$1900.

$$H_0\text{:}\beta_3 + 3.8\beta_4 \leq 1$$
$$H_1\text{:}\beta_3 + 3.8\beta_4 > 1$$

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 section 5.4.3.

```
1  ols sales const price advert sq_advert --vcv
2  scalar r = $coeff(advert)+3.8*$coeff(sq_advert)-1
3  scalar v = $vcv[3,3]+((3.8)^2)*$vcv[4,4]+2*(3.8)*$vcv[3,4]
4
5  scalar tratio = r/sqrt(v)
6  scalar crit = critical(t,$df,.05)
7  scalar p = pvalue(t,$df,tratio)
8
9  printf "\n Ho: b2+3.8b3=1 vs Ha: b2+3.8b3 > 1 \n \
10 the t-ratio is = %.3f \n \
11 the critical value is = %.3f \n \
12 and the p-value = %.3f\n", tratio, crit,  p
```

The hypothesis is in line 2 and the estimated variance in line 3. This was easily done in the script. The results are:

```
Ho: b2+3.8b3=1 vs Ha: b2+3.8b3 > 1
  the t-ratio is = 0.968
  the critical value is = 1.667
  and the p-value = 0.168
```

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.

### Example 6.7 in *POE5*

Finally, Big Andy makes another conjecture about sales. He is considering a price of \$6 and buying \$1900 in advertising; he expects sales to be \$80,000. Combined with the estimated optimality of \$1900 in advertising leads to the following joint test:

$$H_0\text{: } \beta_3 + 3.8\beta_4 = 1 \text{ and } \beta_1 + 6\beta_2 + 1.9\beta_3 + 1.9^2\beta_4 = 80$$
$$H_1\text{: not } H_0$$

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:

```
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
```

Andy is disappointed with this outcome. The null hypothesis is rejected since the $p$-value associated with the test is $0.0049 < .05$. Sorry Andy!

### Examples 6.2 and 6.5 revisited

In these examples a comparison is made between the finite-sample size version of the hypotheses tests in Examples 6.2 and 6.5 of *POE5* and their asymptotic counterparts. The $\chi^2$ form used in asymptotic tests is very similar to the $F$-form; divide the $\chi^2(J)$ by its degrees of freedom, $J$, and you get the $F$. Their are slight differences in the $\chi^2(J)/J$ and the $F_{J,n-k}$ distributions, which accounts for the small difference in the reported $p$-values.

The two versions are shown below. The $F$-statistic is:

$$F = \frac{(SSE_r - SSE_u)/J}{SSE_u/(n-k)} \sim F_{J,n-k} \qquad \text{if } H_0 \text{ is true} \tag{6.6}$$

and the $\chi^2$ is:

$$C = \frac{(SSE_r - SSE_u)}{SSE_u/(n-k)} \sim \chi^2(J) \qquad \text{if } H_0 \text{ is true} \tag{6.7}$$

It is easy to see that $C/J = F$.

To illustrate this we compare $p$-values of the $F$-statistic version of the test and the $\chi^2$ version. First, the null hypothesis that $\beta_3 = \beta_4 = 0$ is tested against the two-sided alternative as in Example 6.2 (p. 175).

The script for the first hypothesis test uses the `omit` statement with the `--test-only` option. The second `omit` command adds the `--chi-square` option that computes the $\chi^2$ version of the test. This option is not available with the `restrict` version of the test.

```
ols sales const price advert sq_advert

omit advert sq_advert --test-only
scalar F_2_nk = $test

omit advert sq_advert --test-only --chi-square
scalar Chi_2 = $test
```

This produces:

```
Test on Model 2: (--test-only)

  Null hypothesis: the regression parameters are zero for the variables
    advert, sq_advert
  Test statistic: F(2, 71) = 8.44136, p-value 0.000514159

Test on Model 2: (--test-only --chi-square)

  Null hypothesis: the regression parameters are zero for the variables
    advert, sq_advert
  Wald test: Chi-square(2) = 16.8827, p-value 0.000215757
  (F-form: F(2, 71) = 8.44136, p-value 0.000514159)
```

The `--chi-square` option produces both versions of the statistic and both $p$-values. The $F$ version of the test has a larger $p$-value, but they are both well below a 5% threshold and are significant.

The second example considers a single hypothesis and compares the $F_{1,n-k}$ to a $\chi^2(1)$. The null-hypothesis is $\beta_3 + 3.8\beta_4 = 1$ against the two-sided alternative (not equal one).

```
restrict --quiet
    b[3]+3.8*b[4]=1
end restrict

scalar F_1_nk = $test
scalar Chi_1 =  $test

pvalue F 1 $df F_1_nk
pvalue C 1 Chi_1
```

This produces:

```
F(1, 71): area to the right of 0.936195 = 0.336543
(to the left: 0.663457)
```

```
Chi-square(1): area to the right of 0.936195 = 0.333258
(to the left: 0.666742)
```

As expected the $F$ version of the test has a slightly larger $p$-value, but they are very similar in magnitude and neither is significantly different from zero at 5%.

### Example 6.8 in *POE5*

In section 5.6.1 a nonlinear function of the parameters was proposed as an estimate of the optimal level of advertising. In this example we test to determine whether this optimal level of advertising is equal to $1900. The optimal level was determined to be:

$$advert_o = \frac{1-\beta_3}{2\beta_4} \tag{6.8}$$

The null hypothesis is that $advert_o = 1.9$ against the alternative $advert_o \neq 1.9$.

Gretl's `restrict` block can be used to test a nonlinear hypothesis after estimation of a single equation linear model. The basic syntax is:

```
ols y const x2 x3 x4
restrict --quiet
    rfunc = [some function of the estimates, b returned by ols]
end restrict
```

First, a linear regression is estimated by least squares. Then the `restrict` block is initiated (using `--quiet` is optional). The next line uses `rfunc` as the name given to a user written function that depends on the elements of the estimated coefficient matrix before closing the `restrict` block.

In this example the null hypothesis is $(1-\beta_3)/(2\beta_4) = 1.9$. Rearranging it becomes $v = ((1-\beta_3)/(2\beta_4)) - 1.9 = 0$. The function argument must be

```
const matrix b
```

which stands for **constraint matrix b**. This says the function is a constraint (to test), and that the argument `b`, i.e., the coefficient matrix from the previous estimation, is a matrix. The only part of this that is user defined is the function name, `restr`. Leave the rest alone![1]

Run the function and estimate the model using `restrict`.

[1] That is, unless the routine you are using calls the estimated parameters something other than `b`. `ols` refers to it as `b` so it works here.

```
function matrix restr (const matrix b)
     matrix v = (1-b[3])/(2*b[4])-1.9
return v
end function

ols sales const price advert sq_advert
restrict --quiet
    rfunc = restr
end restrict
```

The result displayed to the screen is:

```
Test statistic: chi^2(1) = 0.789008, with p-value = 0.3744
```

The hypothesis cannot be rejected at the 5% level.

## 6.2 Nonsample Information

### Example 6.9 in *POE5*

In this section a log-log beer demand model is estimated. The data are in *beer.gdt* and are in level form. The model is:

$$\ln(q) = \beta_1 + \beta_2 \ln(pb) + \beta_3 \ln(pl) + \beta_4 \ln(pr) + \beta_5 \ln(i) + e \tag{6.9}$$

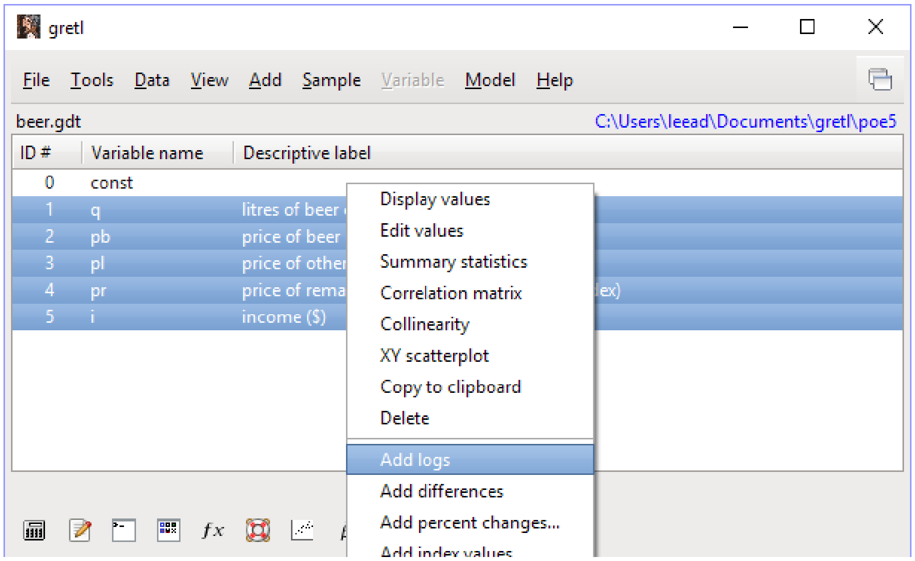
First, convert each of the variables into natural logs using the GUI or the `logs` command.

From the GUI use the cursor to highlight the variables you want transformed in the main window. Right-click the mouse and choose **Add Logs** from the pop-up menu as shown in Figure 6.6. The natural log of each of the variables is obtained and the result stored in a new variable with the prefix `l_` ("el" underscore). As shown previously this can be done in a script or from the console using the `logs` command `logs q pb pl pr i`.[2]

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 "@workdir\data\beer.gdt"
logs q pb pl pr i
ols l_q const l_pb l_pl l_pr l_i --quiet
```

[2] Recall that the there is also a menu item **Add>Add logs of selected variables** that does this too.



| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | q | litres of beer |
| 2 | pb | price of beer |
| 3 | pl | price of other |
| 4 | pr | price of rema ... ex) |
| 5 | i | income ($) |

Figure 6.6: Highlight the desired variables, right-click in the variables window, and choose **Add Logs**.

```
restrict
    b2+b3+b4+b5=0
end restrict
```

The syntax for the restrictions is undocumented. The command reference suggests referring to the coefficients by their position number in the parameter vector as in:

```
restrict
    b[2]+b[3]+b[4]+b[5]=0
end restrict
```

The abbreviated version remains undocumented in the **gretl 2018a** and whether it will continue to work is unknown. 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 appear below.

```
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
```

## 6.3 Model Specification

### Example 6.10, 6.11, and 6.12 in *POE5*

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

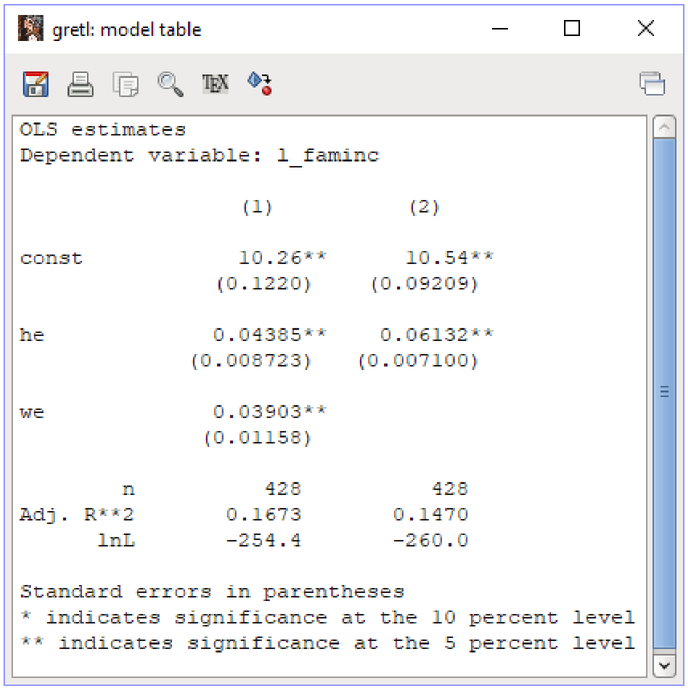
$$l\_faminc_i = \beta_1 + \beta_2 he_i + \beta_3 we + e_i \tag{6.10}$$

where *l_faminc* is the natural logarithm of family income, *he* is husband's years of schooling, *we* is woman's years of schooling. Several variations of this model are estimated that include the number of children in the household under age 6 (*kl6*) and two irrelevant variables, $x_5$ and $x_6$.

```
open "@workdir\data\edu_inc.gdt"
logs faminc
m1 <- ols l_faminc const he we
modeltab add
m2 <- omit we
modeltab add
modeltab show
modeltab --output=two_models.tex
```

The data are opened, log of family income is taken and the baseline regression is estimated. A hypothesis test of the significance of woman's schooling is conducted.

This adds the models to the current session and adds the models to a model table. This also populates the **model table** icon in **gretl**'s icon view (a.k.a. session window). The window is shown below in Figure 6.7.



```
OLS estimates
Dependent variable: l_faminc

                     (1)           (2)

const              10.26**       10.54**
                  (0.1220)     (0.09209)

he               0.04385**     0.06132**
                (0.008723)    (0.007100)

we               0.03903**
                 (0.01158)

        n           428           428
Adj. R**2        0.1673        0.1470
      lnL        -254.4        -260.0

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

Figure 6.7: The `modeltab` commands can be used to construct a model table. This can be saved as LaTeX or RTF.

The LaTeX output is shown below:

OLS estimates
Dependent variable: l_faminc

| | (1) | (2) |
|---|---|---|
| const | 10.26** | 10.54** |
| | (0.1220) | (0.09209) |
| he | 0.04385** | 0.06132** |
| | (0.008723) | (0.007100) |
| we | 0.03903** | |
| | (0.01158) | |
| $n$ | 428 | 428 |
| $\bar{R}^2$ | 0.1673 | 0.1470 |
| $\ell$ | −254.4 | −260 |

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

One interesting thing here is that the omit command accepts the assignment operator (<–) that adds the restricted model to the current session.

In the above script, we have used the modeltab function after each estimated model to add it to the model table. The next to last line tells **gretl** to display the model table in a window and the last line writes the table to a LaTeX file. You can also write it to an *.rtf* file for inclusion in a MS Word document.

The models estimated from the GUI 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 1.17) and drag each model onto the model table icon. Click on the model table icon to reveal the output shown in Figure 6.7.

In Table 6.1 of *POE5* five models of family income are estimated. I've created a variable list for each model's set of regressors:

```
list x1 = const he
list x2 = const he we
list x3 = const he we kl6
list x4 = const he we kl6 xtra_x5 xtra_x6
list x5 = const he kl6 xtra_x5 xtra_x6
```

Using these it is easy to assemble all five models into a model table.

```
modeltab free
m1 <- ols l_faminc x2 --quiet
modeltab add
m2 <- ols l_faminc x1 --quiet
modeltab add
m3 <- ols l_faminc x3 --quiet
modeltab add
m4 <- ols l_faminc x4 --quiet
modeltab add
m5 <- ols l_faminc x5 --quiet
modeltab add
modeltab show
modeltab --output=family_inc_modeltable.tex
```

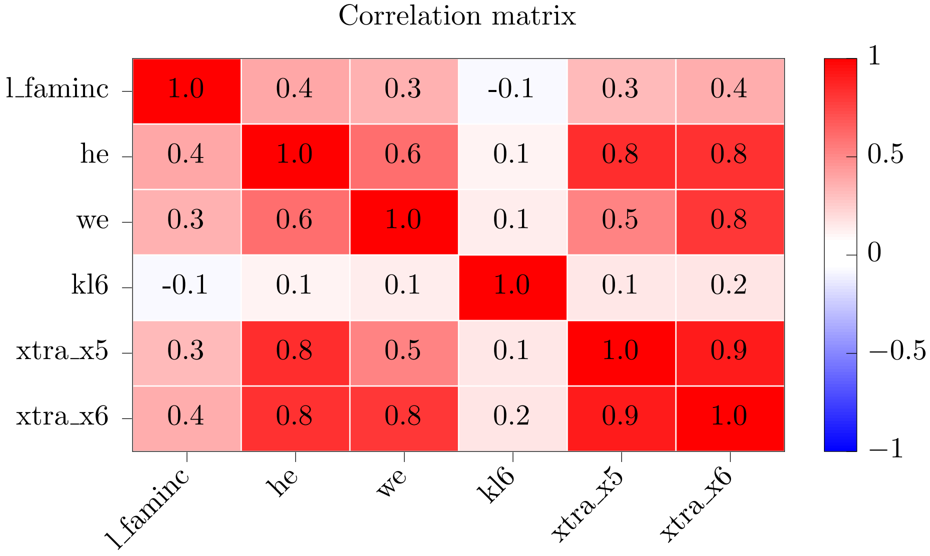
The **gretl** script to estimate these models and test the implied hypothesis restrictions follows.

OLS estimates
Dependent variable: l_faminc

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| const | 10.26$^{**}$ | 10.54$^{**}$ | 10.24$^{**}$ | 10.24$^{**}$ | 10.31$^{**}$ |
| | (0.1220) | (0.09209) | (0.1210) | (0.1214) | (0.1165) |
| he | 0.04385$^{**}$ | 0.06132$^{**}$ | 0.04482$^{**}$ | 0.04602$^{**}$ | 0.05171$^{**}$ |
| | (0.008723) | (0.007100) | (0.008635) | (0.01355) | (0.01329) |
| we | 0.03903$^{**}$ | | 0.04211$^{**}$ | 0.04922$^{**}$ | |
| | (0.01158) | | (0.01150) | (0.02470) | |
| kl6 | | | $-0.1733^{**}$ | $-0.1724^{**}$ | $-0.1690^{**}$ |
| | | | (0.05423) | (0.05468) | (0.05484) |
| xtra_x5 | | | | 0.005388 | $-0.03214^{**}$ |
| | | | | (0.02431) | (0.01543) |
| xtra_x6 | | | | $-0.006937$ | 0.03093$^{**}$ |
| | | | | (0.02148) | (0.01007) |
| $n$ | 428 | 428 | 428 | 428 | 428 |
| $\bar{R}^2$ | 0.1673 | 0.1470 | 0.1849 | 0.1813 | 0.1756 |
| $\ell$ | $-254.4$ | $-260$ | $-249.3$ | $-249.2$ | $-251.2$ |

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

Table 6.3: Model Table from LaTeX



| | l_faminc | he | we | kl6 | xtra_x5 | xtra_x6 |
|---|---|---|---|---|---|---|
| l_faminc | 1.0 | 0.4 | 0.3 | -0.1 | 0.3 | 0.4 |
| he | 0.4 | 1.0 | 0.6 | 0.1 | 0.8 | 0.8 |
| we | 0.3 | 0.6 | 1.0 | 0.1 | 0.5 | 0.8 |
| kl6 | -0.1 | 0.1 | 0.1 | 1.0 | 0.1 | 0.2 |
| xtra_x5 | 0.3 | 0.8 | 0.5 | 0.1 | 1.0 | 0.9 |
| xtra_x6 | 0.4 | 0.8 | 0.8 | 0.2 | 0.9 | 1.0 |

Table 6.4: Heatmat of the correlation matrix produced in **gretl** .

Table 6.2 in *POE5* contains the correlation matrix for variables used in the family income example. This is easily produced using the `corr` function.

```
corr l_faminc he we kl6 xtra_x5 xtra_x6 --plot=heatmap.tex
```

which produces the a heatmap  shown in Table 6.4. The LaTeX code is written to the *heatmap.tex*[3] file in the **gretl** working directory. Darker shades indicate higher correlation and red (blue) indicates positive (negative) correlation.

## 6.4 Model Selection

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.

[3] This required a little editing because the variable names included the underline character. LaTeX uses this symbol in math mode to signify a subscript. That is not what we wanted so the LaTeX code had to be modified slightly by using _ in place of _ in the LaTeX source code.

Here are a few rules of thumb:

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.

### 6.4.1 Adjusted R-square

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:

$$\bar{R}^2 = 1 - \frac{SSE/(n-k)}{SST/(n-1)} \tag{6.11}$$

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.

### 6.4.2 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:

$$AIC = \ln(SSE/n) + 2k/n \tag{6.12}$$

$$BIC = SC = \ln(SSE/n) + k\ln(n)/n \tag{6.13}$$

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 (6.12) 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.

### 6.4.3 A gretl Function to Produce Model Selection Rules

As discussed in section 3.2 **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 model selection function is designed to do two things. First, it prints values of the model selection rules for $\bar{R}^2$, $\bar{R}^2$, $AIC$ and $SC$. It also prints the sample size, number of regressors, and their names. It also sends the computed statistics to a matrix. This allows 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 given the function. The `modelsel` function has two inputs. The first is a data `series` that will be referred to inside the body of the function as `y`. The second is a variable `list` that will be referred to as `xvars`. The inputs are separated by a comma and there are spaces between the list of inputs. Feed the function a dependent variable and a list of the independent variables as inputs. The function estimates a model using ols, computes the criteria 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 = $nobs
 5     scalar k = nelem(xvars)
 6     scalar aic = ln(sse/n)+2*k/n
 7     scalar bic = ln(sse/n)+k*ln(n)/n
 8     scalar rbar2 = 1-((1-$rsq)*(n-1)/$df)
 9     matrix A = { k, n, $rsq, rbar2, aic, bic}
10     printf "\nRegressors: %s\n",varname(xvars)
11     printf " k = %d, n = %d, R2 = %.4f, Adjusted R2 = %.4f,\n\
12 AIC = %.4f, and SC = %.4f\n", k, n, $rsq, rbar2, aic, bic
13     return A
```

In line 2 the function inputs `y` and the list `xvars` are used to estimate a linear model by least squares using the `--quiet` option 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 sends the names of the regressors to the screen. Lines 11 and 12 send formatted output to the screen. Line 13 sends the matrix `A` as a return from the function. The last line closes the function.[4]

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 all_x = const he we xtra_x5 xtra_x6
matrix a = modelsel(l_faminc,all_x)
```

The output is:

[4] To get the **gretl** value of AIC: `scalar aic_g = (1+ln(2*$pi)+aic)*n`.

```
Regressors: const,he,we,kl6,xtra_x5,xtra_x6
 k = 6, n = 428, R2 = 0.1909, Adjusted R2 = 0.1813,
 AIC = -1.6452, and SC = -1.5883
```

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 $k$, $n$, $R^2$, $\bar{R}^2$, $AIC$, and $SC$.

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 kl6 xtra_x5 xtra_x6
5  list x5 = const he kl6 xtra_x5 xtra_x6
6  matrix a = modelsel(l_faminc,x1)
7  matrix b = modelsel(l_faminc,x2)
8  matrix c = modelsel(l_faminc,x3)
9  matrix d = modelsel(l_faminc,x4)
10 matrix e = modelsel(l_faminc,x5)
11
12 matrix MS = a|b|c|d|e
13 cnameset(MS,"k n R2 Adj_R2 AIC SC" )
14 printf "%10.5g", MS
```

In this example the model selection rules will be computed for five different models. Lines 1-5 construct the variable list for each of these. The next five 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, e`). The `cnameset` function is used to give each column of the matrix a meaningful name. Then, the `printf` statement prints the matrix.

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 using `function modelsel clear` 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, R2 = 0.1490, Adjusted R2 = 0.1470,
 AIC = -1.6135, and SC = -1.5945

Regressors: const,he,we
 K = 3, N = 428, R2 = 0.1712, Adjusted R2 = 0.1673,
 AIC = -1.6352, and SC = -1.6067
```

```
Regressors: const,he,we,kl6
 K = 4, N = 428, R2 = 0.1907, Adjusted R2 = 0.1849,
 AIC = -1.6543, and SC = -1.6164

Regressors: const,he,we,kl6,xtra_x5,xtra_x6
 K = 6, N = 428, R2 = 0.1909, Adjusted R2 = 0.1813,
 AIC = -1.6452, and SC = -1.5883

Regressors: const,he,kl6,xtra_x5,xtra_x6
 K = 5, N = 428, R2 = 0.1833, Adjusted R2 = 0.1756,
 AIC = -1.6405, and SC = -1.5931
```

The last part prints the matrix `MS`.

```
k          n         R2     Adj_R2        AIC         SC
2        428    0.14903    0.14704    -1.6135    -1.5945
3        428    0.17117    0.16727    -1.6352    -1.6067
4        428    0.19067    0.18494    -1.6543    -1.6164
6        428    0.19091    0.18132    -1.6452    -1.5883
5        428    0.18329    0.17557    -1.6405    -1.5931
```

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 $R^2$.

### 6.4.4 RESET

#### Example 6.14 in *POE5*

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

$$y_i = \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} + e_i \tag{6.14}$$

and the hypothesis

$$\begin{array}{ll} H_0\text{:} & E[y|x_{i2}, x_{i3}] = \beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3} \\ H_1\text{:} & \text{not } H_0 \end{array}$$

Rejection of $H_0$ implies that the functional form is not supported by the data. To test this, first estimate (6.14) 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:

$$H_0: \quad \gamma_1 = 0$$
$$H_0: \quad \gamma_1 = \gamma_2 = 0$$

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 l_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 only)
Test statistic: F = 1.738326,
with p-value = P(F(1,423) > 1.73833) = 0.188

RESET test for specification (squares and cubes)
Test statistic: F = 1.278259,
with p-value = P(F(2,422) > 1.27826) = 0.28
```

The adequacy of the functional form is not rejected at the 5% level for both tests.

## 6.5 Prediction

### Example 6.15 in *POE5*

In this example we compute a prediction interval for sales at Andy's Burger Barn. The prediction is for a price of \$6 and advertising expenditures of \$1900. This type of problem was first encountered in section 4.1 and refined using matrices in section 4.8. That latter approach is taken here.

The computation is based on the `in_sample_fcast_error` function which computes forecast errors for every observation in a sample. In this example, I only want to evaluate the prediction at one specific point and to compute its standard deviation to use in a prediction interval.

In the script below, the data are loaded, advertising squared added to the data, and a regression estimated. The coefficients are saved in a vector, `b`, and the variance-covariance saved in `covmat`.

Line 6 is the point at which the prediction will be computed. When *price* is \$6 and *advert*ising is 1.9 (\$100). Advertising squared is added as well. This requires the variables to be ordered in the same way that they are in the variable list used in the regression (`sales const price advert sq_advert`). Line 7 computes the prediction and the quadratic form for the variance computation is done in line 8 using the `qform` command. The variance is computed and the square root taken to produce the standard error.

```
1  open "@workdir\data\andy.gdt"
2  square advert
3  ols sales const price advert sq_advert
4  matrix b = $coeff
5  matrix covmat = $vcv
6  matrix x_0 = { 1, 6, 1.9, 1.9^2 }
7  matrix pred = x_0*b
8  matrix v = (qform(x_0,covmat))+$sigma^2
9  matrix se = sqrt(v)
10 t_interval(pred, se, $df, .95)
```

Finally, our `t_interval` program (see page 59) is used to compute the interval and to print the output to the screen. This produces:

```
The 95% confidence interval centered at 76.974 is (67.5326, 86.4155)
```

These are measured in \$100 and match the results in *POE5* exactly.

## Example 6.16 *POE5*

Table 6.4 in *POE5* contains model selection criteria for housing data. The data are found in *br5.gdt*. Load them, square age, and take the natural logarithm of price. A list of regressors is created and two scalars are added. The first will make the last observation in a $n_1 = 800$ observation subsample and the second will mark the last observation in the data. These will be used in a moment.

The model under study is of housing prices in Baton Rouge. The model is

$$\ln(price) = \beta_1 + \beta_2 age + \beta_3 sqft + \beta_4 age^2 + \beta_5 sqft^2 + \beta_6(age \times sqft) + e \tag{6.15}$$

```
open "@workdir\data\br5.gdt"
square age
logs price
list xvars = const sqft age sq_age
```

```
scalar t1 = 800
scalar t2 = 900
```

The model will be estimated using the first 800 observations. Based on these estimates, the 100 remaining observations, referred to as a hold-out sample, will be predicted using the estimated model. Gretl produces a number of statistics that are useful in evaluating the quality of forecasts. Among them is the Root-Mean-Square-Error:

$$RMSE = \sqrt{\frac{1}{n_2}\sum_{n_1+1}^{n}(y_i - \hat{y}_i)^2}$$

$n_2$ is the number of observations in the hold-sample, $n_1$ the number in the estimation sample.

Fortunately, the `fcast` function will compute what we need for this example.

```
smpl 1 t1
ols l_price xvars
smpl 1 t2
fcast 801 900 --static --stats-only
```

The `smpl` command restricts the sample to observations 1-800. The model is estimated, the sample restored, and the `fcast` command used to produce `--static` forecast of observations 901-900. The `--stats-only` option limits the output to the forecast quality measures shown below:

```
Forecast evaluation statistics

 Mean Error                           -0.029709
 Root Mean Squared Error               0.27136
 Mean Absolute Error                   0.19242
 Mean Percentage Error                -1.104
 Mean Absolute Percentage Error        4.1927
 Theil's U                             0.30121
 Bias proportion, UM                   0.011986
 Regression proportion, UR             0.043132
 Disturbance proportion, UD            0.94488
```

The RMSE for this model and sample is 0.27136, which matches *POE5*.

Table 6.4 in *POE5* contains model selection criteria and RMSE for eight different models. To facilitate the computation of RMSE multiple times, I wrote a crude RMSE program to compute the statistics for the table.

```
# Function to compute RMSE for t1, t2
function matrix rmse (series yvar, list xvars, scalar t1, scalar t2)
    matrix y = yvar                  # yvar into matrix
    matrix X_all = { xvars }         # xvars into matrix
    matrix y1 = y[1:t1,]             # Estimation subset y
    matrix X = X_all[1:t2,]          # Sample restricted to 1-t2
    matrix X1 = X_all[1:t1,]         # Estimation subset regressors
    matrix Px1 = X*inv(X1’X1)*X1’y1  # Yhat for entire 1:t2 sample
    matrix ehat = y[1:t2,]-Px1       # Y-Yhat for entire 1:t2 sample
    matrix ehatp = ehat[t1+1:t2,]    # Residuals for pred. sub-period
    matrix RMSE = sqrt(ehatp’ehatp/(t2-t1))# MSEP residuals
    return RMSE
end function
```

All of the computations are in matrix form and this won’t work if your data contain missing values. However, ours does not and this works fine for what we want it to do. The function returns a matrix (a scalar equal to RMSE) and uses four inputs. The dependent variable for a regression, a list of independent variables to use in the regression, and two scalars to mark the last observation in the estimation sample and the last observation in the hold-out sample.

To confirm that it works, it is used on the preceding model:

```
scalar r1 = rmse(l_price, xvars, 800, 900)
printf "RMSE for observations %g to %g = %.4f\n", 800, 900, r1
```

This produces:

```
RMSE for observations 800 to 900 = 0.2714
```

which matches the result from `fcast`.

To reproduce what is in the table you can try this rudimentary script.

```
series age_sqft = age*sqft
list x1 = const sqft age
list x2 = x1 sq_age
list x3 = x1 sq_sqft
list x4 = x1 age_sqft
list x5 = x1 sq_age sq_sqft
list x6 = x1 sq_age age_sqft
list x7 = x1 sq_sqft age_sqft
list x8 = x1 sq_sqft sq_age age_sqft

```

```
matrix a = modelsel(l_price,x1)
matrix b = modelsel(l_price,x2)
matrix c = modelsel(l_price,x3)
matrix d = modelsel(l_price,x4)
matrix e = modelsel(l_price,x5)
matrix f = modelsel(l_price,x6)
matrix g = modelsel(l_price,x7)
matrix h = modelsel(l_price,x8)

matrix ra = rmse(l_price,x1,t1,t2)
matrix rb = rmse(l_price,x2,t1,t2)
matrix rc = rmse(l_price,x3,t1,t2)
matrix rd = rmse(l_price,x4,t1,t2)
matrix re = rmse(l_price,x5,t1,t2)
matrix rf = rmse(l_price,x6,t1,t2)
matrix rg = rmse(l_price,x7,t1,t2)
matrix rh = rmse(l_price,x8,t1,t2)

matrix MS = a|b|c|d|e|f|g|h
matrix RMS = ra|rb|rc|rd|re|rf|rg|rh
matrix all_crit = MS~RMS
cnameset(all_crit,"k n R2 Adj_R2 AIC SC RMSE" )
printf "%10.5g", all_crit
```

The resulting matrix matches Table 6.4 quite well.

```
k          n       R2     Adj_R2       AIC        SC      RMSE
3        900   0.6985     0.6978    -2.534    -2.518    0.2791
4        900   0.7207     0.7198    -2.609    -2.587    0.2714
4        900   0.6992     0.6982    -2.535    -2.513    0.2841
4        900   0.6996     0.6986    -2.536    -2.515     0.279
5        900   0.7208     0.7196    -2.607     -2.58    0.2754
5        900    0.721     0.7197    -2.608    -2.581    0.2712
5        900   0.7006     0.6993    -2.537     -2.51     0.284
6        900   0.7212     0.7197    -2.606    -2.574    0.2754
```

We could clearly improve upon this by adding the actual model variables in a row, but I'll leave that as an exercise. Also, keep in mind that the column labeled n pertains to the estimation sample for the model selection rules, not the RMSE calculation.

## 6.6 Collinearity in Rice Production

The data set *rice5.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 rice production example from *POE5*. This is log-log model of production (tonnes of rice) that is a depends on area under cultivation (hectares), labor input (person-days), and fertilizer (kilograms).

$$\ln(prod) = \beta_1 + \beta_2 \ln(area) + \beta_3 \ln(labor) + \beta_4 \ln(fert) + e$$

The script is

```
open "@workdir\data\rice5.gdt"
summary --simple
corr area fert labor prod
logs area fert labor prod
corr l_area l_fert l_labor l_prod
```

The summary statistics in levels are:

```
                   Mean       Median         S.D.          Min          Max
firm              22.50        22.50        12.77        1.000        44.00
area              2.120        1.750        1.420       0.2000        5.500
fert              176.4        128.7        154.3        10.00        595.7
labor             107.4        90.50        71.12        11.00        381.0
prod              6.169        4.995        4.849       0.6000        21.07
year               1994         1994       0.5029         1993         1994
```

The correlation matrix of the levels is:

Correlation coefficients, using the observations 1:1–44:2
5% critical value (two-tailed) = 0.2096 for n = 88

| area | fert | labor | prod | year | |
|---|---|---|---|---|---|
| 1.0000 | 0.8531 | 0.9093 | 0.8347 | −0.0056 | area |
| | 1.0000 | 0.8656 | 0.8584 | 0.0461 | fert |
| | | 1.0000 | 0.8865 | −0.0002 | labor |
| | | | 1.0000 | −0.0439 | prod |
| | | | | 1.0000 | year |

The variables are quite highly correlated in the sample. For instance the correlation between area and labor input is 0.9093. Large farms use more labor. What a surprise!

Taking logarithms won't change much. The correlations among the log variables are:

Correlation coefficients, using the observations 1:1–44:2
5% critical value (two-tailed) = 0.2096 for n = 88

| l_area | l_fert | l_labor | l_prod | year | |
|---|---|---|---|---|---|
| 1.0000 | 0.8387 | 0.9320 | 0.8856 | −0.0048 | l_area |
| | 1.0000 | 0.8790 | 0.8981 | 0.0343 | l_fert |
| | | 1.0000 | 0.9130 | −0.0409 | l_labor |
| | | | 1.0000 | −0.0784 | l_prod |
| | | | | 1.0000 | year |

The correlation between ln(*area*) and ln(*labor*) actually increases slightly to 0.932.

The production model is estimated for 1994.

```
smpl (year==1994) --restrict
m_1994 <- ols l_prod const l_area l_labor l_fert
omit l_area l_labor --test-only
```

The regression result is:

m_1994: OLS, using observations 1–44
Dependent variable: l_prod

| | Coefficient | Std. Error | *t*-ratio | p-value |
|---|---|---|---|---|
| const | −1.94729 | 0.738487 | −2.637 | 0.0119 |
| l_area | 0.210607 | 0.182074 | 1.157 | 0.2543 |
| l_labor | 0.377584 | 0.255058 | 1.480 | 0.1466 |
| l_fert | 0.343335 | 0.127998 | 2.682 | 0.0106 |

| | | | |
|---|---|---|---|
| Mean dependent var | 1.457871 | S.D. dependent var | 0.852785 |
| Sum squared resid | 3.924527 | S.E. of regression | 0.313230 |
| $R^2$ | 0.874501 | Adjusted $R^2$ | 0.865089 |
| $F(3, 40)$ | 92.90939 | P-value($F$) | 4.53e–18 |
| Log-likelihood | −9.260529 | Akaike criterion | 26.52106 |
| Schwarz criterion | 33.65782 | Hannan–Quinn | 29.16771 |

The test of the individual significance of the coefficients can be read from the table of regression results. Only the coefficient of `l_fert` is significant at 5%. The overall $F$-statistic is 92.9 and its $p$-value is well below 5%. The $R^2 = 0.875$, which seems fairly large. The joint significance of $\beta_2$ and $\beta_3$ is tested using `omit`. The coefficients are jointly different from zero, since the $p$-value for this test is $0.0021 < 0.05$.

```
Null hypothesis: the regression parameters are zero for the variables
    l_area, l_labor
  Test statistic: F(2, 40) = 7.1918, p-value 0.00214705
```

Finally, collinearity is examined using the `vif` function after the regression. **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 recommendation provided by (Hill et al., 2018, p. 291) 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 regress $x_j$ on all of the other independent variables and compare the $R_j^2$ from the auxiliary regression to 10. If the $R_j^2$ exceeds 10, then there is evidence of a collinearity problem.

The $vif_j$ reports the same information, but in a less straightforward way. The *vif* associated with the $j^{th}$ regressor is computed

$$vif_j = \frac{1}{1 - R_j^2} \tag{6.16}$$

which is, as you can see, simply a function of the $R_j^2$ from the $j^{th}$ auxiliary regression. Notice that when $R_j^2 > .80$, 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 vifs for the log-log rice production model estimated for 1994 are:

```
Variance Inflation Factors
Minimum possible value = 1.0
Values > 10.0 may indicate a collinearity problem

      l_area    9.149
     l_labor   17.734
      l_fert    7.684

VIF(j) = 1/(1 - R(j)^2), where R(j) is the multiple correlation
coefficient between variable j and the other independent variables
```

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 $R_j^2$. 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.

Interval estimates for each of the slopes can be obtained using the `t_interval` function after estimation. However, since the model results were sent to the session window, it is easier to use the GUI. Navigate to the session window and double-click on the `m_1994` icon to bring up its models window. From its menu bar choose **Analysis>Confidence intervals for coefficients** to reveal

```
t(40, 0.025) = 2.021

      VARIABLE         COEFFICIENT      95% CONFIDENCE INTERVAL

         const           -1.94729         -3.43982    -0.454749
        l_area            0.210607       -0.157378     0.578592
       l_labor            0.377584       -0.137907     0.893075
        l_fert            0.343335       0.0846404     0.602029
```

One suggestion for mitigating the effects of collinearity is to impose restrictions on the parameters of the model. Suppose one knows that returns to rice production are constant. This implies that $\beta_2 + \beta_3 + \beta_4 = 1$. Using this as a restriction

```
restrict m_1994 --full
    b[2]+b[3]+b[4]=1
end restrict
```

This particular script includes two new options for `restrict`. The first allows the restrict statement to be applied to a model that you have stored. In this case it is the model `m_1994` that was saved to a session as an icon. The second is the `--full` option. This option when used in most contexts replaces the current contents of most accessors with the ones from the restricted model. So in this example we want to form confidence intervals for the restricted coefficients, we would need the restricted least squares results. Those become available from the accessors if the `--full` option is used with `restrict`. leads to:

```
Restriction:
 b[l_area] + b[l_labor] + b[l_fert] = 1

Test statistic: F(1, 40) = 1.04387, with p-value = 0.313062
```

```
Restricted estimates:

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const       -2.16830      0.706472    -3.069    0.0038   ***
  l_area       0.226228     0.181528     1.246    0.2197
  l_labor      0.483419     0.233200     2.073    0.0445   **
  l_fert       0.290353     0.117086     2.480    0.0173   **
```

The restriction as a hypothesis is not rejected at 5%. Its p-value is 0.31. From the restricted model, `l_labor` is now statistically significant at 5%.

To find the confidence intervals use the `t_interval` program:

```
t_interval($coeff(l_area),$stderr(l_area),$df,.95)
t_interval($coeff(l_labor),$stderr(l_labor),$df,.95)
t_interval($coeff(l_fert),$stderr(l_fert),$df,.95)
```

which produces:

```
The 95% confidence interval centered at 0.226 is (-0.1404, 0.5928)

The 95% confidence interval centered at 0.483 is (0.0125, 0.9544)

The 95% confidence interval centered at 0.290 is (0.0539, 0.5268)
```

Finally, we'll repeat the estimation of the rice production model using the full sample, computing vifs, and computing 95% confidence intervals.

```
smpl full
m_full <- ols l_prod const l_area l_labor l_fert
vif
t_interval($coeff(l_area),$stderr(l_area),$df,.95)
t_interval($coeff(l_labor),$stderr(l_labor),$df,.95)
t_interval($coeff(l_fert),$stderr(l_fert),$df,.95)
```

The results are:

m_full: Pooled OLS, using 88 observations
Included 44 cross-sectional units

Time-series length = 2
Dependent variable: l_prod

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | −1.86940 | 0.456543 | −4.095 | 0.0001 |
| l_area | 0.210789 | 0.108286 | 1.947 | 0.0549 |
| l_labor | 0.399672 | 0.130650 | 3.059 | 0.0030 |
| l_fert | 0.319456 | 0.0635063 | 5.030 | 0.0000 |

| | | | |
|---|---|---|---|
| Mean dependent var | 1.520993 | S.D. dependent var | 0.809932 |
| Sum squared resid | 6.898693 | S.E. of regression | 0.286579 |
| $R^2$ | 0.879121 | Adjusted $R^2$ | 0.874804 |
| $F(3, 84)$ | 203.6369 | P-value($F$) | 1.99e–38 |
| Log-likelihood | −12.84238 | Akaike criterion | 33.68476 |
| Schwarz criterion | 43.59411 | Hannan–Quinn | 37.67699 |

The confidence intervals can either be obtained using our function or from the GUI in the models window. Recall that this is available even using a script if you assign the output to a name (as we have here with `m_full`).

```
t(84, 0.025) = 1.989

      VARIABLE         COEFFICIENT      95% CONFIDENCE INTERVAL

         const           -1.86940         -2.77728    -0.961510
        l_area           0.210789      -0.00454921     0.426127
       l_labor           0.399672         0.139861     0.659483
        l_fert           0.319456         0.193166     0.445745
```

The vif output is

```
Variance Inflation Factors
Minimum possible value = 1.0
Values > 10.0 may indicate a collinearity problem

      l_area    7.705
     l_labor   10.051
      l_fert    4.455

VIF(j) = 1/(1 - R(j)^2), where R(j) is the multiple correlation
coefficient between variable j and the other independent variables
```

The largest vif is now down to around 10, which is a bit better that in the unrestricted model (where it was 17.7).

## 6.7 Influential Observations

### Example 6.18 in *POE5*

In section 4.5 we developed programs for computing diagnostics that can be used to detect the influence of an observation on the regression output. In this example, we use those to analyze the housing model estimated using the *br5.gdt* data.

$$\ln(price) = \beta_1 + \beta_2 age + \beta_3 sqft + \beta_4 age^2 + \beta_5 sqft^2 + \beta_6(age \times price) + e$$

The script to estimate the model and to collect the desired statistics is:

```
 1  open "@workdir\data\br5.gdt"
 2  genr index
 3  logs price
 4  square age sqft
 5
 6  list xvars = const sqft age sq_age
 7  ols l_price xvars
 8  leverage --save --quiet
 9
10  series uhat = $uhat
11  series lev_t = h_t(xvars)
12  series sig_t = delete_1_variance(l_price, xvars)
13  series stu_res = uhat/sqrt(sig_t*(1-lev_t))
14  series DFFits=stu_res*sqrt(lev_t/(1-lev_t))
15
16  list x1 = xvars
17  scalar k = nelem(xvars)
18  matrix results = zeros(k,1)
19  loop i=1..k --quiet
20      list y1 = x1[1]
21      list y2 = x1[2:k]
22      ols y1 y2
23      series dfb$i=stu_res*$uhat/sqrt($ess*(1-lev_t))
24      list x1 = y2 y1
25  endloop
26
27  store influential.gdt index sig_t lev_t stu_res DFFits\
28        dfb1 dfb2 dfb3 dfb4
29  series ab_dfb2 = abs(dfb2)
30  series ab_stu_res = abs(stu_res)
31  series ab_DFFits = abs(DFFits)
```

There is not much new here. In lines 10-14 we collect residuals, use our user written programs from section 4.5 `h_t` to compute leverage and `sig_t` to compute the delete-one variances. Studentized residuals and DFFits follow. The loop in lines 18-25 collect the DFBETAs for all of the regressors. Everything is stored to an external dataset *infuential.gdt* that will be located in the working directory.

Lines 29-31 create series that contain the absolute values of DFBETA(2), studentized residuals, and DFFits. We want to find the observations that are most influential and these statistics can be large negative or positive numbers.

Then, to reduce the amount of output to a manageable level we sort by each series and print only the largest five values of the statistic, along with its original observation number.

```
1  dataset sortby ab_dfb2
2  smpl $nobs-5 $nobs
3  print index dfb2 --byobs
4
5  smpl full
6  dataset sortby lev_t
7  smpl $nobs-5 $nobs
8  print index lev_t --byobs
9
10 smpl full
11 dataset sortby ab_stu_res
12 smpl $nobs-5 $nobs
13 print index stu_res --byobs
14
15 smpl full
16 dataset sortby ab_DFFits
17 smpl $nobs-5 $nobs
18 print index DFFits --byobs
```

The sorting is done based on the `full` sample using the `dataset sortby` command. Then the sample is reduced to the last five in the data using `smpl $nobs-5 $nobs`, and printed using the `--byobs` option in line 8. Here is some output for the DFBETA for the *sqrt* coefficient:

```
             index         dfb2

895            836   -0.2055396
896            472   -0.2403838
897            356   -0.2441436
898            859    0.2570806
899            787   -0.2708825
900            411   -0.6577355
```

The most influential observation on the second coefficient is 411 followed by observation 787.

For leverage, $h_t$

```
              index         lev_t

   895          420       0.04012
   896          148       0.06205
   897          392       0.06232
   898          605       0.06244
   899          150       0.06369
   900          497       0.06395
```

Observation 797 has the highest leverage.

For studentized residuals we have

```
              index       stu_res

   895          283     -3.853597
   896           51     -3.885458
   897          503     -4.258513
   898          524     -4.313883
   899          898     -4.744688
   900          411     -4.980408
```

which shows observation 411 being influential by this measure.

Finally, DFFits

```
              index        DFFits

   895          160     0.4602006
   896          831    -0.4619299
   897           94    -0.4685957
   898          150     0.5114895
   899          524    -0.5600395
   900          411    -0.9036011
```

Predictions are influenced most by observation 411 with observation 524 a close runner-up.

## 6.8 Nonlinear Least Squares

### Example 6.19

Models that are nonlinear in the parameters and an additive error term are candidates for nonlinear least squares estimation. In this example we estimate a one parameter model using nonlinear least squares.

The model is

$$y_t = \beta x_{t1} + \beta^2 x_{t2} + e_t$$

Since the parameter is squared and the error is additive, this model is a candidate for nonlinear least squares estimation. The minimum of the sum of squared errors function cannot be solved analytically for $\beta$ in terms of the data. So, a numerical solution to the least squares normal equations must be found.

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, **gretl** requires an extra step on your part. You have to type in an equation that contains parameters and variables for **gretl** to estimate. This is the way one works in EViews and other software by default, so the relative burden here is low.

Nonlinear least squares (and other nonlinear estimators) use numerical methods rather than analytical ones to minimize the sum of squared errors objective function. The routines that do this iterative until the user is satisfied that no more improvements in the sum-of-squares function can be had.

The routines require you to provide 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 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 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.

The routine in **gretl** that does this is `nls`. To use `nls` the user must specify a regression function. The function will contain variables as named in the dataset and a set of user named parameters. The parameters must be "declared" and given initial values. Optionally, one may supply analytical derivatives of the regression function with respect to each of the parameters that determine the direction of the next step. If derivatives are not given, you must give a list of the parameters to be estimated (separated by spaces or commas), preceded by the keyword `params`. The tolerance (criterion for terminating the iterative estimation procedure) can be adjusted using the `set` command. The syntax for specifying the function to be estimated is the same as for the `genr` command.

For the single parameter model we have:

```
open "@workdir\data\nlls.gdt"
scalar b=1
    nls y=b*x1+b^2*x2
    params b
end nls
```

The dataset is *nlls.gdt* and the starting value for the parameter b is set to 1. The third line is the model, and the params statement b follows (since we are not supplying analytical derivatives of the function in line 3). Run the routine to obtain:

```
Using numerical derivatives
Tolerance = 1.81899e-012
Convergence achieved after 11 iterations

Model 1: NLS, using observations 1-20
y = b*x1+b^2*x2

             estimate   std. error   t-ratio   p-value
  -----------------------------------------------------
  b          1.16121     0.130666     8.887    3.40e-08 ***

Mean dependent var   1.184900   S.D. dependent var   1.047650
Sum squared resid    16.30797   S.E. of regression   0.926452
Uncentered R-squared 0.217987   Centered R-squared  -0.000618
Log-likelihood      26.33799   Akaike criterion     54.67597
Schwarz criterion    55.67171   Hannan-Quinn         54.87035

GNR: R-squared = 0, max |t| = 1.45037e-009
Convergence seems to be reasonably complete
```

The nonlinear least squares estimate of $\beta$ is 1.612. The estimated standard error is 0.131. Notice other common regression statistics are reported as well (though a few a missing). Notice that the centered $R^2$ is negative. Obviously this statistic is not bounded between 1 and 0 in a nonlinear model.

### Example 6.20

In this example another simple nonlinear model is estimated. This one is a logistic growth curve and is estimated using data on the share of total U.S. crude steel production that is produced by electric arc furnaces. The output is a function of time, $t$.

$$y_t = \frac{\alpha}{1+\exp(-\beta-\delta t)} + e_t$$

This is interesting. There is one variable, $t$=time period, and three parameters ($\alpha$, $\beta$, and $\delta$).

The script is:

```
open "@workdir\data\steel.gdt"
 # Starting Values
scalar alpha = 1                          # Starting Values
```

```
4  scalar delta = .1
5  scalar beta = -1
6
7  nls eaf = alpha/(1 + exp(-beta-delta*t))  # Regression function
8      params alpha delta beta                # Parameters: remember the order
9  end nls
```

The starting values are given in lines 3-5. Here you have to use your judgement (or in my case, luck). The logistic growth curve is in line 7, where `eaf` is the dependent variable.

Run the routine to obtain:

```
Using numerical derivatives
Tolerance = 1.81899e-012
Convergence achieved after 29 iterations

Model 1: NLS, using observations 1970-2015 (T = 46)
eaf = alpha/(1 + exp(-beta-delta*t))

              estimate    std. error   t-ratio    p-value
  --------------------------------------------------------
  alpha       0.814375    0.0510501     15.95    1.14e-019 ***
  delta       0.0572226   0.00430389    13.30    7.85e-017 ***
  beta       1.37767     0.0563557    24.45    7.50e-027 ***

Mean dependent var   0.401087   S.D. dependent var   0.146314
Sum squared resid    0.017272   S.E. of regression   0.020042
Uncentered R-squared 0.982070   Centered R-squared  -0.000128
Log-likelihood       116.1364   Akaike criterion    226.2728
Schwarz criterion   220.7868   Hannan-Quinn        224.2177
rho                  0.794149   Durbin-Watson        0.392075

GNR: R-squared = 2.22045e-016, max |t| = 9.54129e-008
Convergence seems to be reasonably complete
```

It took 29 iterations to converge and **gretl** appears to be satisfied that convergence to a (local) minimum has been achieved.

An inflection point occurs in this model at $-\beta/\delta$. This can be computed using accessors. A parameter can no longer be referenced by the variable that accompanies it. There is no longer a one-to-one correspondence. Instead, the accessors can be used on the parameter name that you have designated. Therefore to compute the inflection and print it to the screen we use:

```
10 printf "\n Inflection Point = %.3f\n ",\
11        -$coeff(beta)/$coeff(delta) # Function
```

which yields:

```
Inflection Point = 24.076
```

Finally, I'll add a little lagniappe. Since the model is estimated we might as well go ahead and compute a confidence interval for the inflection point. It is a nonlinear function of the estimates and the Delta method can be used to compute its variance.

```
1 matrix covmat = $vcv                                # Save the covariance
2 matrix d={0;$coeff(beta)/($coeff(delta)^2);-1/$coeff(delta)} # Derivative
3 scalar se = sqrt(d'*covmat*d)                       # Std errors
4 t_interval(-$coeff(beta)/$coeff(delta),se,$df,.95)  # t_interval
5 printf "\nThe Delta estimated standard error is %.3f \n",  se
```

The accessor `$vcv` is used to save the variance-covariance matrix computed by `nls`. The second line consists of the derivatives of the function with respect to $\alpha$, $\beta$, and $\delta$. Line 3 is the standard error of the inflection point, i.e., the square root of the quadratic form. All of this is combined and used in the `t_interval` function in line 4. For good measure, I print out the estimated standard error.

## 6.9 Script

### 6.9.1 Functions

```
set verbose off
# function estimates confidence intervals based on the t-distribution
function void t_interval (scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# function computes prediction standard errors
    function series in_sample_fcast_error (series y, list xvars)
    ols y xvars
    scalar sig = $sigma^2
    matrix X = { xvars }
    matrix f_e = sig*I($nobs)+sig*X*inv(X'X)*X'
    series se = sqrt(diag(f_e))
    return se
end function

# function to compute diagonals of hat matrix
function series h_t (list xvars)
    matrix X = { xvars }
    matrix Px = X*inv(X'X)*X'
    matrix h_t = diag(Px)
    series hats = h_t
    return hats
end function

# delete-one variance function
function series delete_1_variance (series y, list xvars)
    matrix sig = zeros($nobs,1)
    loop i=1..$nobs --quiet
        matrix e_t = zeros($nobs,1)
        matrix e_t[i,1]=1
        series et = e_t
        ols y xvars et --quiet
        matrix sig[i,1]=$sigma^2
    endloop
    series sig_t = sig
    return sig_t
end function

# 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, $rsq, rbar2, aic, bic}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "k = %d, n = %d, R2 = %.4f, Adjusted R2 = %.4f, AIC = %.4f,\
and SC = %.4f\n", k, N, $rsq, rbar2, aic, bic
    return A
end function

# Function to compute RMSE for t1, t2
function matrix rmse (series yvar, list xvars, scalar t1, scalar t2)
    matrix y = yvar                  # Put yvar into matrix
    matrix X_all = { xvars }         # Put xvars into matrix
    matrix y1 = y[1:t1,]             # Estimation subset y
    matrix X = X_all[1:t2,]          # Sample restricted to 1-t2
    matrix X1 = X_all[1:t1,]         # Estimation subset regressors
    matrix Px1 = X*inv(X1'X1)*X1'y1  # Yhat for entire 1:t2 sample
    matrix ehat = y[1:t2,]-Px1       # Y-Yhat for entire 1:t2 sample
    matrix ehatp = ehat[t1+1:t2,]    # Residuals for the prediction sub-period
    matrix RMSE = sqrt(ehatp’ehatp/(t2-t1))# Mean of squared prediction residuals
    return RMSE
end function

# f-test
# Example 6.1
open "@workdir\data\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

# Example 6.2
# f-test using omit
ols sales const price advert sq_advert
omit advert sq_advert --test-only

# f-test using restrict
set echo off
set messages off
m2 <- ols sales const price advert sq_advert
restrict --quiet
b[3]=0
b[4]=0
end restrict

# Example 6.3
# overall f
set echo off
set messages off
open "@workdir\data\andy.gdt"
square advert
ols sales const price advert sq_advert
restrict --quiet
  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

# Using the Wald option with omit
open "@workdir\data\andy.gdt"
square advert
list xvars = price advert sq_advert
ols sales const xvars --quiet
omit xvars --wald
omit xvars

# Example 6.4
# t-test
ols sales const price advert sq_advert
omit price --test-only
scalar t_2 = ($coeff(price)/$stderr(price))^2
scalar F_test = $test
print t_2 F_test

# Example 6.5
# optimal advertising
open "@workdir\data\andy.gdt"
square advert
ols sales const price advert sq_advert
scalar Ao =(1-$coeff(advert))/(2*$coeff(sq_advert))
printf "\nThe optimal level of advertising is $%.2f\n", Ao*1000
# test of optimal advertising
restrict --quiet
    b[advert]+3.8*b[sq_advert]=1
end restrict

# Example 6.6
# 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)*$vcv[3,4]
scalar tratio = r/sqrt(v)
scalar crit = critical(t,$df,.05)
scalar p = pvalue(t,$df,tratio)
printf "\n Ho: b2+3.8b3=1 vs Ha: b2+3.8b3 > 1 \n \
the t-ratio is = %.3f \n \
the critical value is = %.3f \n \
and the p-value = %.3f\n", tratio, crit,  p

# Example 6.7
# joint test
ols sales const price advert sq_advert
restrict --quiet
    b[3]+3.8*b[4]=1
    b[1]+6*b[2]+1.9*b[3]+3.61*b[4]=80
```

```
end restrict

# Examples 6.2 and 6.5 revisited
ols sales const price advert sq_advert
omit advert sq_advert --test-only
scalar F_2_nk = $test
omit advert sq_advert --test-only --chi-square
scalar Chi_2 = $test
pvalue F 2 $df F_2_nk
pvalue C 2 Chi_2

restrict --quiet
    b[3]+3.8*b[4]=1
end restrict
scalar F_1_nk = $test
scalar Chi_1 =  $test
pvalue F 1 $df F_1_nk
pvalue C 1 Chi_1

# Example 6.8
# Nonlinear Hypothesis
function matrix restr (const matrix b)
     matrix v = (1-b[3])/(2*b[4])-1.9
return v
end function

ols sales const price advert sq_advert
restrict --quiet
    rfunc = restr
end restrict

# Example 6.9
# restricted estimation
open "@workdir\data\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 --quiet
    b[2]+b[3]+b[4]+b[5]=0
end restrict

# Example 6.10
# model specification -- relevant and irrelevant vars
open "@workdir\data\edu_inc.gdt"
logs faminc
m1 <- ols l_faminc const he we
modeltab add
m2 <- omit we
```

```
modeltab add
modeltab show

corr l_faminc he we kl6 xtra_x5 xtra_x6 --plot=corr.tex

# Example 6.11
ols l_faminc const he we kl6

# Example 6.12
list all_x = const he we kl6 xtra_x5 xtra_x6
ols l_faminc all_x
matrix a = modelsel(l_faminc,all_x)

list x1 = const he
list x2 = const he we
list x3 = const he we kl6
list x4 = const he we kl6 xtra_x5 xtra_x6
list x5 = const he kl6 xtra_x5 xtra_x6
matrix a = modelsel(l_faminc,x1)
matrix b = modelsel(l_faminc,x2)
matrix c = modelsel(l_faminc,x3)
matrix d = modelsel(l_faminc,x4)
matrix e = modelsel(l_faminc,x5)

matrix MS = a|b|c|d|e
cnameset(MS,"k n R2 Adj_R2 AIC SC" )
printf "%10.5g", MS

# Table 6.1 in POE5
modeltab free
m1 <- ols l_faminc x2 --quiet
modeltab add
m2 <- ols l_faminc x1 --quiet
modeltab add
m3 <- ols l_faminc x3 --quiet
modeltab add
m4 <- ols l_faminc x4 --quiet
modeltab add
m5 <- ols l_faminc x5 --quiet
modeltab add
modeltab show
modeltab --output=family_inc_modeltable.tex

# Example 6.13
# Control for ability in wage equation
open "@workdir\data\koop_tobias_87.gdt"
logs wage
square exper
ols l_wage const educ exper sq_exper score
omit score

```

```
# Example 6.14
# reset test
open "@workdir\data\edu_inc.gdt"
logs faminc

ols l_faminc const he we kl6
reset --quiet --squares-only
reset --quiet

ols l_faminc he we kl6 --quiet
reset

/*---POE5 Example 6.15---*/
# Forecasting SALES for the Burger Barn
open "@workdir\data\andy.gdt"
square advert
ols sales const price advert sq_advert
matrix b = $coeff
matrix covmat = $vcv
matrix x_0 = { 1, 6, 1.9, 1.9^2 }
matrix pred = x_0*b
matrix v = (qform(x_0,covmat))+$sigma^2
matrix se = sqrt(v)
t_interval(pred, se, $df, .95)

/*---POE5 Example 6.16---*/
# Predicting House Prices
open "@workdir\data\br5.gdt"
    set echo off
    set messages off
square age sqft
logs price
list xvars = const sqft age sq_age
scalar t1 = 800
scalar t2 = 900

smpl 1 t1
ols l_price xvars
smpl 1 t2
fcast 801 900 --static --stats-only

scalar r1 = rmse(l_price, xvars, 800, 900)
matrix m1 = modelsel(l_price,xvars)
printf "RMSE for observations %g to %g = %.4f\n", 800, 900, r1

series age_sqft = age*sqft
list x1 = const sqft age
list x2 = x1 sq_age
list x3 = x1 sq_sqft
list x4 = x1 age_sqft
list x5 = x1 sq_age sq_sqft
list x6 = x1 sq_age age_sqft
list x7 = x1 sq_sqft age_sqft
list x8 = x1 sq_sqft sq_age age_sqft
matrix a = modelsel(l_price,x1)
matrix b = modelsel(l_price,x2)
matrix c = modelsel(l_price,x3)
matrix d = modelsel(l_price,x4)
matrix e = modelsel(l_price,x5)
matrix f = modelsel(l_price,x6)
matrix g = modelsel(l_price,x7)
matrix h = modelsel(l_price,x8)

matrix ra = rmse(l_price,x1,t1,t2)
matrix rb = rmse(l_price,x2,t1,t2)
matrix rc = rmse(l_price,x3,t1,t2)
matrix rd = rmse(l_price,x4,t1,t2)
matrix re = rmse(l_price,x5,t1,t2)
matrix rf = rmse(l_price,x6,t1,t2)
matrix rg = rmse(l_price,x7,t1,t2)
matrix rh = rmse(l_price,x8,t1,t2)

matrix MS = a|b|c|d|e|f|g|h
matrix RMS = ra|rb|rc|rd|re|rf|rg|rh
matrix all_crit = MS~RMS
cnameset(all_crit,"k n R2 Adj_R2 AIC SC RMSE" )
printf "%10.5g", all_crit

/*---POE5 Example 6.17---*/
# Collinearity in a Rice Production Function
open "@workdir\data\rice5.gdt"
summary --simple
corr area fert labor prod year
logs area fert labor prod
corr l_area l_fert l_labor l_prod year

smpl (year==1994) --restrict
m_1994 <- ols l_prod const l_area l_labor l_fert
omit l_area l_labor --test-only
vif

restrict m_1994 --full
    b[2]+b[3]+b[4]=1
end restrict

t_interval($coeff(l_area),$stderr(l_area),$df,.95)
t_interval($coeff(l_labor),$stderr(l_labor),$df,.95)
t_interval($coeff(l_fert),$stderr(l_fert),$df,.95)

smpl full
m_full <- ols l_prod const l_area l_labor l_fert
vif
```

```
t_interval($coeff(l_area),$stderr(l_area),$df,.95)
t_interval($coeff(l_labor),$stderr(l_labor),$df,.95)
t_interval($coeff(l_fert),$stderr(l_fert),$df,.95)
/*---POE5 Example 6.18---*/
# Influential Observations in the House Price Equation
open "@workdir\data\br5.gdt"
genr index
logs price
square age sqft

list xvars = const sqft age sq_age
ols l_price xvars
leverage --save --quiet

series uhat = $uhat
series lev_t = h_t(xvars)
series sig_t = delete_1_variance(l_price, xvars)
series stu_res = uhat/sqrt(sig_t*(1-lev_t))
series DFFits=stu_res*sqrt(lev_t/(1-lev_t))

list x1 = xvars
scalar k = nelem(xvars)
matrix results = zeros(k,1)
loop i=1..k --quiet
    list y1 = x1[1]
    list y2 = x1[2:k]
    ols y1 y2
    series dfb$i=stu_res*$uhat/sqrt($ess*(1-lev_t))
    list x1 = y2 y1
endloop

store influential.gdt index sig_t lev_t stu_res DFFits dfb1 dfb2 dfb3 dfb4
series ab_dfb2=abs(dfb2)
series ab_stu_res = abs(stu_res)
series ab_DFFits = abs(DFFits)

dataset sortby ab_dfb2
smpl $nobs-5 $nobs
print index dfb2 --byobs

smpl full
dataset sortby lev_t
smpl $nobs-5 $nobs
print index lev_t --byobs

smpl full
dataset sortby ab_stu_res
smpl $nobs-5 $nobs
print index stu_res --byobs

smpl full
```

```
dataset sortby ab_DFFits
smpl $nobs-5 $nobs
print index DFFits --byobs

/*---POE5 Example 6.19---*/
# Nonlinear Least Squares Estimates for Simple Model
open "@workdir\data\nlls.gdt"
scalar b=1
    nls y=b*x1+b^2*x2
    params b
end nls

/*---POE5 Example 6.20---*/
# A Logistic Growth Curve
open "@workdir\data\steel.gdt"
 # Starting Values
scalar alpha = 1                          # Starting Values
scalar delta = .1
scalar beta = -1

nls eaf = alpha/(1 + exp(-beta-delta*t))  # Regression function
    params alpha delta beta               # Parameters: remember the order
end nls
matrix covmat = $vcv                                # Save the covariance
printf "\nInflection Point = %.3f\n ", -$coeff(beta)/$coeff(delta)
matrix d={0;$coeff(beta)/($coeff(delta)^2);-1/$coeff(delta)} # Derivative
scalar se = sqrt(d'*covmat*d)                       # Std errors
t_interval(-$coeff(beta)/$coeff(delta),se,$df,.95)  # confidence interval
printf "\nThe Delta estimated standard error is %.3f \n",  se
```

# 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 indicators 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** 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.

### Example 7.1 in *POE5*

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 "@workdir\data\utown.gdt"
summary --simple
smpl 1 6
print --byobs
```

```
5 smpl full
6 smpl $nobs-4 $nobs
7 print --byobs
```

The sample is limited to the first 6 observations in line 3. 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, six 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
  ...

 996   257.195      22.84          4          1          0          0
 997   338.295      30.00         11          1          0          1
 998   263.526      23.99          6          1          0          0
 999   300.728      28.74          9          1          0          0
1000   220.987      20.93          2          1          0          1
```

The sample is restored to completeness and then limited to the last five observations. These are printed as well.

The simple summary statistics for the entire sample from line 2 appear below. 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.

```
                      Mean       Median         S.D.          Min          Max
price                247.7        245.8        42.19        134.3        345.2
sqft                 25.21        25.36        2.918        20.03        30.00
age                  9.392        6.000        9.427       0.0000        60.00
utown               0.5190        1.000       0.4999       0.0000        1.000
pool                0.2040       0.0000       0.4032       0.0000        1.000
fplace              0.5180        1.000       0.4999       0.0000        1.000
```

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.

```
1 series ld = (sqft>25)
2 discrete ld
3 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.

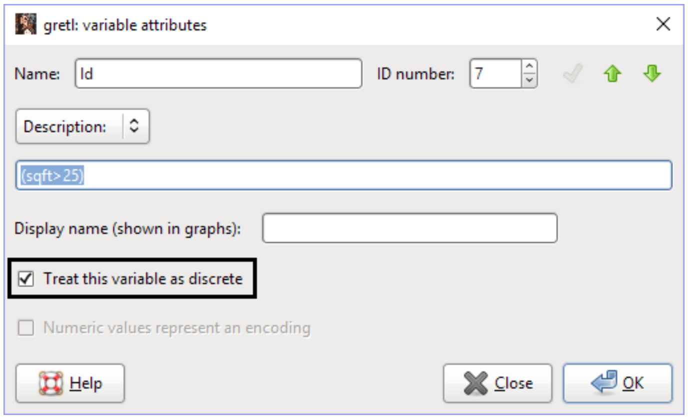
The heuristic is as follows: First, are all the values of the variable “reasonably round”, where this is taken to mean that they are all integer multiples of 0.25? If this criterion is met, we then ask whether the variable takes on a fairly small set of distinct values, where fairly small is defined as less than or equal to 8. If both conditions are satisfied, the variable is automatically considered discrete.”(Cottrell and Lucchetti, 2018, p. 84)

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
```



Name: ld
ID number: 7
Description: (sqft>25)
Display name (shown in graphs):
☑ Treat this variable as discrete
□ Numeric values represent an encoding

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.

Indicator variables can also be created using the conditional assignment operator. The variable *ld* could be created using:

```
series large = (sqft > 25) ? 1 : 0
```

The series would be called `large` and if the expression inside parentheses is true (i.e., the house has more than 2500 square feet, then it takes the value that follows the question mark (?), which is 1. If the statement is not true, it is assigned the value that follows the colon (i.e., 0). The conditional assignment operator can be used with compound logic as well. In the next example, a series called midprice is given the value 1 if the price falls between 215 and 275 using:

```
series midprice = (215 < price) && (price < 275) ? 1 : 0
```

The double ampersands means and in this case. If both are true (*price* greater than 215 **and** less than 275, `midprice` is assigned the value 1. Otherwise, it is zero. A brief printout of the result demonstrates success.

```
           price          sqft         large      midprice
```

```
 1      205.452        23.46             0             0
 2      185.328        20.03             0             0
 3      248.422        27.77             1             1
 4      154.690        20.17             0             0
 5      221.801        26.45             1             1
```

Finally, indicators can be interacted with other indicators or continuous variables using lists. Suppose we create two lists. The first contains an indicator, utown, which is 0 if the house is not located in the University Town subdivision. The second list contains both continuous and indicators (sqft, age, and pool). A set of interaction variables can be created using the following syntax:

```
1 list house = sqft age pool
2 list loc = utown
3 list inter = utown ^ house
4 print inter -o
```

The list called inter in line 3 contains the interaction of the `utown` list and the `loc` list; the operator is ^. Note, the indicator list must be to the left of ^. This produces:

```
    sqft_utown_0 sqft_utown_1  age_utown_0

 1        23.46      0.00          6
 2        20.03      0.00          5
 3        27.77      0.00          6
 4        20.17      0.00          1
 5        26.45      0.00          0


     age_utown_1 pool_utown_0  pool_utown_1

 1        0            0            0
 2        0            0            0
 3        0            0            0
 4        0            0            0
 5        0            0            0
```

Recall that none of the first five houses in the sample are in University Town. So, when interacted with *utown*=1, the interaction variables are all zero. Also, none of the houses had a pool, hence `pool_utown` is 0 for *utown*=1 and for *utown*=0. Also, notice that the `--byobs` option is abbreviated with the simple switch `-o` in the print statement.

### 7.1.2 Estimating a Regression

The following regression in Example 7.1 is 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 of 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

#### Example 7.2 in *POE5*

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

$$
E[wage|educ] = \begin{cases} \beta_1 + \beta_2 educ & White,\ Males \\ \beta_1 + \delta_1 + \beta_2 educ & Black,\ Males \\ \beta_1 + \delta_2 + \beta_2 educ & White,\ Females \\ \beta_1 + \delta_1 + \delta_2 + \gamma + \beta_2 educ & Black,\ Females \end{cases} \tag{7.1}
$$

The **reference group** is the one where all indicator variables are zero, i.e., white males. The parameter $\delta_1$ measures the effect of being black, relative to the reference group; $\delta_2$ measures the effect of being female relative to the reference group, and $\gamma$ measures the effect of being both black and female.

The model is estimated using the *cps5_small.gdt* data which is from March 2013. The script is:

```
1 open "@workdir\data\cps5_small.gdt"
2 series black_female = black * female
3 list demographic = black female black_female
4 m1 <- ols wage const educ demographic
5 omit demographic --test-only
```

The results appear below:

m1: OLS, using observations 1–1200
Dependent variable: wage

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | −9.482 | 1.958 | −4.843 | 0.0000 |
| educ | 2.474 | 0.1351 | 18.31 | 0.0000 |
| black | −2.065 | 2.162 | −0.9554 | 0.3396 |
| female | −4.223 | 0.8249 | −5.120 | 0.0000 |
| black_female | 0.5329 | 2.802 | 0.1902 | 0.8492 |

| | | | |
|---|---|---|---|
| Mean dependent var | 23.64004 | S.D. dependent var | 15.21655 |
| Sum squared resid | 214400.9 | S.E. of regression | 13.39459 |
| $R^2$ | 0.227720 | Adjusted $R^2$ | 0.225135 |
| $F(4, 1195)$ | 88.09155 | P-value($F$) | 1.21e–65 |
| Log-likelihood | −4814.042 | Akaike criterion | 9638.084 |
| Schwarz criterion | 9663.534 | Hannan–Quinn | 9647.671 |

Holding the years of schooling constant, black males earn \$2.07/hour less than white males. For the same schooling, white females earn \$4.22 less, and black females earn \$.53 more. 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 conducted using the omit command in line 5. The results are:

```
Test on Model 1:

  Null hypothesis: the regression parameters are zero for the variables
```

```
    black, female, black_female
  Test statistic: F(3, 1195) = 10.5183, p-value 7.8715e-007
  Omitting variables improved 0 of 3 information criteria.
```

The test statistic is 10.5 and the p-value from the F(3, 1195) distribution is well below 5%. The null hypothesis is rejected.

and the result is

```
Test on Model 1:

  Null hypothesis: the regression parameters are zero for the variables
    black, female, black_female
  Test statistic: F(3, 1195) = 10.5183, p-value 7.8715e-007
  Omitting variables improved 0 of 3 information criteria.
```

### 7.2.2 Regional indicators

#### Example 7.3 in *POE5*

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

$$E[wage|educ] = \begin{cases} \beta_1 + \beta_2 educ & \text{Northeast} \\ \beta_1 + \delta_1 + \beta_2 educ & \text{South} \\ \beta_1 + \delta_2 + \beta_2 educ & \text{Midwest} \\ \beta_1 + \delta_3 + \beta_2 educ & \text{West} \end{cases} \tag{7.2}$$

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.

```
list regions = south midwest west
m2 <- ols wage const educ demographic regions
omit regions --test-only
```

The results from both models appear below:

OLS estimates
Dependent variable: wage

| | (1) | (2) |
|---|---|---|
| const | −9.482** | −8.371** |
| | (1.958) | (2.154) |
| educ | 2.474** | 2.467** |
| | (0.1351) | (0.1351) |
| black | −2.065 | −1.878 |
| | (2.162) | (2.180) |
| female | −4.223** | −4.186** |
| | (0.8249) | (0.8246) |
| black_female | 0.5329 | 0.6190 |
| | (2.802) | (2.801) |
| south | | −1.652 |
| | | (1.156) |
| midwest | | −1.939 |
| | | (1.208) |
| west | | −0.1452 |
| | | (1.203) |
| $n$ | 1200 | 1200 |
| $\bar{R}^2$ | 0.2251 | 0.2263 |
| $\ell$ | −4814 | −4812 |

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

Workers in the south are expected to earn $1.65 less per hour than those in the northeast holding other variables constant. However, none of the regional indicators is individually significant at 5%. The joint test results are

```
Test on Model 2:

  Null hypothesis: the regression parameters are zero for the variables
    south, midwest, west
  Test statistic: F(3, 1192) = 1.57923, p-value 0.192647
```

The test statistic has an $F(3,\ 992)$ distribution under the null and is equal to 1.57. The $p$-value is greater than 5% and we conclude that the indicators are not jointly significant. We could not conclude that workers with the same education, race and gender in the regions earn different amounts per hour.

### 7.2.3 Testing Equivalence of Two Regions

#### Example 7.4 in *POE5*

The question arises, is the wage equation different for the south than for the rest of the country? There are several ways to do this in **gretl**. One uses the ability to interact variable lists. The other uses smpl commands to estimate models in different subsamples. The `chow` command is able to test the equivalence of subsample regressions based on a indicator variable to determine the subsamples.

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 first method used to estimate the model uses the indicator variable south to create interactions. The script is:

```
1 open "@workdir\data\cps5_small.gdt"
2 series black_female = black * female
3 list demographic = black female black_female
4 list xvars = const educ demographic
5 list inter = south ^ xvars
6 ols wage inter
```

First, *black* and *female* are interacted and a series formed. This is included in the list demographic along with its elements, *black* and *female*. All of the variables are assembled into another list `xvars` which is then interacted with the indicator `south` in line 5. The regression is estimated in line 6. The result appears below:

```
Model 4: OLS, using observations 1-1200
Dependent variable: wage

                      coefficient   std. error   t-ratio    p-value
  --------------------------------------------------------------
  const_south_0         9.99910      2.38723     4.189    3.01e-05  ***
  const_south_1        8.41619     3.43377    2.451   0.0144    **
  educ_south_0          2.52714     0.164196   15.39    6.95e-049 ***
  educ_south_1          2.35572     0.238825    9.864   4.10e-022 ***
  black_south_0         1.12757     3.52466     0.3199  0.7491
  black_south_1        3.49279     2.80905    1.243   0.2140
  female_south_0       4.15199     0.984150   4.219   2.64e-05  ***
  female_south_1       4.34061     1.51665    2.862   0.0043    ***
  black_female_s~_0    4.45398     4.48577    0.9929  0.3210
  black_female_s~_1     3.66549     3.71079     0.9878  0.3235

Mean dependent var   23.64004   S.D. dependent var   15.21655
Sum squared resid    213774.0   S.E. of regression   13.40306
R-squared            0.229978   Adjusted R-squared   0.224154
F(9, 1190)           39.49009   P-value(F)           7.85e-62
```

This matches the results in the first column of Table 7.5 in *POE5*. By interacting each of the variables including the constant with the indicator, we have essentially estimated two separate regressions in one model. Note, the standard errors are computed based on the assumption that the two subsamples have the same overall variance, $\sigma^2$. The next approach does not assume this and the standard errors will be a bit different.

To estimate the equations separately we employ the `smpl` command to restrict the sample to either the south or elsewhere.

```
smpl full
smpl (south==1) --restrict
M_south <- ols wage xvars

smpl full
smpl (south==0) --restrict
M_other <- ols wage xvars
```

We start with the full sample and use the restrict statement with the boolean argument (south==1) to limit the sample to observations where this is true. The model is estimated, the sample restored to full and restricted again to only include observations where south==0. The two models appear below:

OLS estimates
Dependent variable: wage

| | M_south | M_other |
|---|---|---|
| const | $-8.416^{**}$ | $-9.999^{**}$ |
| | (3.871) | (2.227) |
| educ | $2.356^{**}$ | $2.527^{**}$ |

| | | |
|---|---|---|
| | (0.2692) | (0.1532) |
| black | −3.493 | 1.128 |
| | (3.167) | (3.288) |
| female | −4.341** | −4.152** |
| | (1.710) | (0.9182) |
| black_female | 3.665 | −4.454 |
| | (4.183) | (4.185) |
| $n$ | 390 | 810 |
| $\bar{R}^2$ | 0.1635 | 0.2597 |
| $\ell$ | −1610 | −3193 |

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

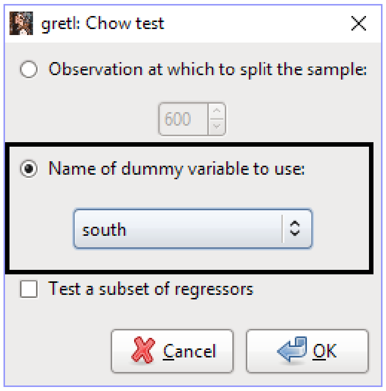
The coefficient estimates match those that were obtained using the indicators. As expected the standard errors differ.

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.

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 smpl full
2 M_pooled <- ols wage xvars
3 chow south --dummy
```

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 `--dummy` option is used to tell **gretl** that `south` is an indicator. When this option is used, `chow` tests the null hypothesis of structural homogeneity with respect to the named indicator. Essentially, **gretl** is creating interaction terms between the indicator and each of the regressors and adding them to the model as done above in Model 4. 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.



○ Observation at which to split the sample: 600

● Name of dummy variable to use: south

□ Test a subset of regressors

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.

```
Augmented regression for Chow test
OLS, using observations 1-1200
Dependent variable: wage

                    coefficient   std. error   t-ratio    p-value
  ----------------------------------------------------------------
  const              9.99910      2.38723     4.189    3.01e-05  ***
  educ                2.52714      0.164196    15.39     6.95e-049 ***
  black               1.12757      3.52466      0.3199   0.7491
  female             4.15199      0.984150    4.219    2.64e-05  ***
  black_female       4.45398      4.48577     0.9929   0.3210
  south               1.58291      4.18206      0.3785   0.7051
  so_educ            0.171420     0.289824    0.5915   0.5543
  so_black           4.62036      4.50710     1.025    0.3055
  so_female          0.188612     1.80798     0.1043   0.9169
  so_black_female     8.11947      5.82169      1.395    0.1634

Mean dependent var   23.64004   S.D. dependent var   15.21655
Sum squared resid    213774.0   S.E. of regression   13.40306
R-squared            0.229978   Adjusted R-squared   0.224154
F(9, 1190)           39.49009   P-value(F)           7.85e-62
Log-likelihood      4812.285   Akaike criterion     9644.570
Schwarz criterion    9695.470   Hannan-Quinn         9663.744

Chow test for structural difference with respect to south
  F(5, 1190) = 0.697969 with p-value 0.6250
```

Notice that the $p$-value associated with the test is 0.625, thus providing insufficient evidence to convince us that wages are structurally different in the south.

The other way to do this uses interactions. Though the `chow` command makes this unnecessary, it is a great exercise that demonstrates how to create more general interactions among variables. Replicating a portion of the script found on page (238):

```
1  list xvars = const educ demographic
2  list inter = south ^ xvars
3  m <- ols wage inter
4  restrict
5      b1-b2=0
6      b3-b4=0
7      b5-b6=0
8      b7-b8=0
9      b9-b10=0
10 end restrict
```

The first line includes each of the variables in the model that are to be interacted with south. Line 2 adds the interactions to the list and the regression is estimated by least squares in line 3. The coefficient restrictions are used to conduct the Chow test. The result indicates exactly what is going on:

```
Restriction set
 1: b[const_south_0] - b[const_south_1] = 0
 2: b[educ_south_0] - b[educ_south_1] = 0
 3: b[black_south_0] - b[black_south_1] = 0
 4: b[female_south_0] - b[female_south_1] = 0
 5: b[black_female_south_0] - b[black_female_south_1] = 0

Test statistic: F(5, 1190) = 0.697969, with p-value = 0.625034

Restricted estimates:

                         coefficient  std. error  t-ratio   p-value
  ------------------------------------------------------------------
  const_south_0           9.48206     1.95797     4.843   1.45e-06  ***
  const_south_1           9.48206     1.95797     4.843   1.45e-06  ***
  educ_south_0             2.47370     0.135104   18.31     3.35e-066 ***
  educ_south_1             2.47370     0.135104   18.31     3.35e-066 ***
  black_south_0           2.06526     2.16163     0.9554   0.3396
  black_south_1           2.06526     2.16163     0.9554   0.3396
  female_south_0          4.22346     0.824927    5.120    3.56e-07  ***
  female_south_1          4.22346     0.824927    5.120    3.56e-07  ***
  black_female_south_0     0.532927    2.80203      0.1902   0.8492
  black_female_south_1     0.532927    2.80203      0.1902   0.8492

  Standard error of the regression = 13.3946
```

The coefficients of the constants, *education*, *black*, *female*, and *black-female* are restricted to be equal to one another and the restriction is tested using an $F$-test. The test statistic is identical to that produced by chow.

### 7.2.4 Log-Linear Models with Indicators

#### Examples 7.5 and 7.6 in *POE5*

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 *POE5*, 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.

```
1 open "@workdir\data\cps5_small.gdt"
2 logs wage
3 ols l_wage const educ female
4
5 scalar wd = exp($coeff(female))-1
6 printf "\nThe estimated male/female wage differential is\
7 = %.3f percent.\n", wd*100
```

The natural logarithm of wage is taken in line 2. Then the model is estimated an the percentage difference computes.

m: OLS, using observations 1–1200
Dependent variable: l_wage

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 1.623 | 0.06917 | 23.46 | 0.0000 |
| educ | 0.1024 | 0.004799 | 21.34 | 0.0000 |
| female | −0.1778 | 0.02794 | −6.364 | 0.0000 |

| | | | |
|---|---|---|---|
| Sum squared resid | 272.2378 | S.E. of regression | 0.476900 |
| $R^2$ | 0.282005 | Adjusted $R^2$ | 0.280806 |
| $F(2, 1197)$ | 235.0716 | P-value($F$) | 7.74e–87 |

The coefficient on education suggests that an additional year of schooling increases the average wage by 10.24%, holding sex constant. The estimated wage differential between men and women of similar education is 17.78% . Using equation (7.4), which is estimated in line 5, we obtain:

```
The estimated male/female wage differential is = -16.288 percent.
```

for a computed difference is $-16.288$, suggesting that females earn about 16.29% less than males who have comparable levels of education. An approximate standard error can be computed via the delta method (discussed at length in section 5.6.1).

```
scalar variance = exp($coeff(female))^2*$vcv[3,3]
scalar se = sqrt(variance)
printf "\nThe estimated standard error is\
= %.3f%% .\n", se*100
```

```
The estimated standard error is = 2.339%.
```

## 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 binary 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.

### Example 7.7 in *POE5*

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.

The script to estimate the model is:

```
1 open "@workdir\data\coke.gdt"
2 summary
3
4 ols coke const pratio disp_coke disp_pepsi --robust
5 series p_hat = $yhat
6 series lt_zero = (p_hat<0)
7 matrix count = sum(lt_zero)
8 printf "\nThere are %.2g predictions that are less than zero.\n", count
```

The data are loaded and summary statistics computed. The regression is estimated by ordinary least squares, with the binary variable coke as the dependent variable. The predictions from OLS are saved as a series in line 5 and in line 6 we count the number of predictions that are less than zero. The main problem with the LPM is that it can predict a probability that is either less than zero or greater than 1, both of which are inconsistent with the theory of probability.

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.

The last line of the script indicates that 16 of the 1140 observation fell below zero:

```
There are 16 predictions that are less than zero.
```

## 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 $\beta_2$. The least squares estimator of $\beta_2$ 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.

### Examples 7.8 in *POE5*

To illustrate, we use the data from project STAR described in *POE5*, Chapter 7.5.

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 "@workdir\data\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      Median       S.D.        Min        Max
totalscore         918.0       912.0      73.14      635.0       1229
small             0.0000      0.0000     0.0000     0.0000     0.0000
tchexper           9.068       9.000      5.724     0.0000      24.00
boy               0.5132       1.000     0.4999     0.0000      1.000
freelunch         0.4738      0.0000     0.4994     0.0000      1.000
white_asian       0.6813       1.000     0.4661     0.0000      1.000
tchwhite          0.7980       1.000     0.4016     0.0000      1.000
tchmasters        0.3651      0.0000     0.4816     0.0000      1.000
schurban          0.3012      0.0000     0.4589     0.0000      1.000
schrural          0.4998      0.0000     0.5001     0.0000      1.000

small = 1 (n = 1738):

                    Mean      Median       S.D.        Min        Max
totalscore         931.9       924.0      76.36      747.0       1253
small              1.000       1.000     0.0000      1.000      1.000
tchexper           8.995       8.000      5.732     0.0000      27.00
boy               0.5150       1.000     0.4999     0.0000      1.000
freelunch         0.4718      0.0000     0.4993     0.0000      1.000
white_asian       0.6847       1.000     0.4648     0.0000      1.000
tchwhite          0.8625       1.000     0.3445     0.0000      1.000
tchmasters        0.3176      0.0000     0.4657     0.0000      1.000
schurban          0.3061      0.0000     0.4610     0.0000      1.000
schrural          0.4626      0.0000     0.4987     0.0000      1.000
```

## Examples 7.9 in *POE5*

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.

In addition 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.

```
smpl aide != 1 --restrict
discrete schid
```

```
3 list fe = dummify(schid)
4 list x1 = const small
5 list x2 = x1 tchexper
```

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. To include school effects, a set of indicator variables is created based on the identification number of the school, schid. To be safe, it is declared to be discrete in line 2 before using the dummify command in line 3 to create the indicators. The indicators are put into a list called fe (fixed effects). 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.

Now each of the models is estimated with the --quiet option and put into a model table.

```
modeltab free

m1 <- ols totalscore x1 --quiet
modeltab add

m2 <- ols totalscore x2 --quiet
modeltab add

m3 <- ols totalscore x1 fe --quiet
omit fe --test-only
modeltab add

m4 <- ols totalscore x2 fe --quiet
t_interval($coeff(small),$stderr(small),$df,.95)
omit fe --test-only
modeltab add
modeltab show
```

For the models that include the school fixed effects, the omit statement is used to test the hypothesis that the school differences are jointly insignificant. A portion of the results appears below:

```
OLS estimates
Dependent variable: totalscore

                        m1            m2            m3            m4

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)

Dschid_123056                                   55.51**      52.90**
                                               (16.16)      (16.14)

Dschid_128068                                   48.27**      51.12**
                                               (16.55)      (16.53)
.....

School effects           NO          NO          YES          YES

             n         3743         3743         3743         3743
     Adj. R**2       0.0083       0.0158       0.2213       0.2245
           lnL  -2.145e+004  -2.144e+004  -2.096e+004  -2.095e+004
```

The coefficient on the *small* indicator variable is not affected by adding or dropping teacher experience from the model. This is indirect evidence that it is not correlated with other regressors. The effects of a small class increase a bit when the school fixed effects are taken into account. The effect of teacher experience on test scores falls quite a bit at the same time. The estimated slopes in columns (3) and (4) match those in *POE5*. The intercepts are different only because a different reference group was used. The substance of the results is unaffected.

The hypothesis tests for fixed effects are significant at 5%. The test results produced for m3 and m4, respectively are:

```
 Test statistic: F(78, 3663) = 14.1177, p-value 1.70964e-154
 Test statistic: F(78, 3662) = 13.9048, p-value 6.65072e-152
```

Also, the 95% confidence interval for the coefficient of small in model four (summoned in line 14) is:

```
The 95% confidence interval centered at 16.066 is (11.7165, 20.4148)
```

It includes each of the other estimates and therefore we would conclude that there is no measurable difference between the size of the effects of small class size on test scores.

### 7.4.1 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.

### Example 7.11 in *POE5*

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 95% confidence interval is obtained using `t_interval`. We find:

```
The 95% confidence interval centered at 0.466 is (0.4170, 0.5160)
```

which includes 0.5, suggesting that assigning children to a small or large class is as fair as a fair coin flip. I think it is safe to omit these other 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.

### Examples 7.12 and 7.13 in *POE5*

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 "@workdir\data\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<br>(1.385) |
| $n$ | 794 | 794 | 794 |
| $\bar{R}^2$ | 0.0036 | 0.1893 | 0.2115 |
| $\ell$ | −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. Also, you will need to limit the sample to the unique observations (in the original, each is duplicated).

```
smpl missing(demp) != 1 --restrict
smpl d = 1 --restrict
ols demp const nj
```

Fortunately, the data set includes the $\Delta FTE$ 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:

$$\widehat{\text{demp}} = -\underset{(1.0355)}{2.28333} + \underset{(1.1543)}{2.75000}\,\text{nj}$$

$$T = 768 \quad \bar{R}^2 = 0.0134 \quad F(1,766) = 11.380 \quad \hat{\sigma} = 8.9560$$

(standard errors in parentheses)

The coefficient on *nj* is not significantly less than zero at the 5% level and we conclude that the increase in minimum wage did not reduce employment.

## 7.6 Script

```
set messages off
# function estimates confidence intervals based on the t-distribution
function void t_interval (scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# Example 7.1
# Indicator Variables in Real Estate Example
open "@workdir\data\utown.gdt"

# summarize and examine
summary --simple
smpl 1 6
print --byobs
smpl full
smpl $nobs-4 $nobs
print --byobs

* estimate dummy variable regression
smpl full
series utown_sqft = utown*sqft
list xvars = const sqft utown age pool fplace utown_sqft
ols price xvars
omit utown utown_sqft --test-only

# generate some marginal effects
scalar premium = $coeff(utown)*1000
scalar sq_u = 10*($coeff(sqft)+$coeff(utown_sqft))
scalar sq_other = 10*$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 = $%.2f\n\
Marginal effect of sqft elsewhere = $%.2f\n\
Depreciation Rate = $%7.2f per year\n\
Pool = $%7.2f\n\
Fireplace = $%7.2f\n",premium,sq_u,sq_other,depr,sp,firep
omit utown_sqft --test-only

# examples creating indicator variables
open "@workdir\data\utown.gdt"
series ld = (sqft>25)
discrete ld

series large = (sqft > 25) ? 1 : 0
series midprice = (215 < price) && (price < 275) ? 1 : 0
```

```
smpl 1 5
print price sqft large midprice --byobs

smpl full
list house = sqft age pool
list loc = utown
list inter = utown ^ house
print inter -o

/*---POE5 Example 7.2---*/
# Applying indicator variables in a wage equation

open "@workdir\data\cps5_small.gdt"
summary --simple

series black_female = black * female
list demographic = black female black_female
m1 <- ols wage const educ demographic
omit demographic --test-only

/*---POE5 Example 7.3---*/
# Add regional indicators to wage equation
list regions = south midwest west
m2 <- ols wage const educ demographic regions
omit regions --test-only

/*---POE5 Example 7.4---*/
# Testing the equivalence of two regressions
list xvars = const educ demographic
list inter = south ^ xvars
m <- ols wage inter
restrict
    b1-b2=0
    b3-b4=0
    b5-b6=0
    b7-b8=0
    b9-b10=0
end restrict

# Estimate separate regressions
smpl (south==1) --restrict_south <- ols wage xvars

smpl full
smpl (south==0) --restrict
M_other <- ols wage xvars

smpl full
M_pooled <- ols wage xvars
chow south --dummy

/*---POE5 Example 7.5---*/
```

```
* Log-linear models
open "@workdir\data\cps5_small.gdt"
logs wage
m <- ols l_wage const educ female

/*---POE5 Example 7.6---*/
scalar wd = exp($coeff(female))-1
printf "\nThe estimated male/female wage differential is\
= %.3f percent.\n", wd*100

scalar variance = exp($coeff(female))^2*$vcv[3,3]
scalar se = sqrt(variance)
printf "\nThe estimated standard error is\
= %.3f%%.\n", se*100

/*---POE5 Example 7.7---*/
# Linear Probability Model

open "@workdir\data\coke.gdt"
summary

ols coke const pratio disp_coke disp_pepsi
series p_hat = $yhat
series lt_zero = (p_hat<0)
matrix count = sum(lt_zero)
printf "\nThere are %.2g predictions that are less than zero.\n", count

/*---POE5 Example 7.8---*/
open "@workdir\data\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
summary v
smpl (aide == 0) --restrict
summary --simple

# create lists
list x1 = const small
list x2 = x1 tchexper
list x3 = x2 boy freelunch white_asian
list x4 = x3 tchwhite tchmasters schurban schrural

summary totalscore x4 --by=small --simple

corr x3

/*---POE5 Example 7.9 and 7.10---*/
# regressions
open "@workdir\data\star.gdt"
discrete schid
```

```
list fe = dummify(schid)
list x1 = const small
list x2 = x1 tchexper
smpl aide != 1 --restrict

modeltab free

m1 <- ols totalscore x1 --quiet
modeltab add

m2 <- ols totalscore x2 --quiet
modeltab add

m3 <- ols totalscore x1 fe
omit fe --test-only
modeltab add

m4 <- ols totalscore x2 fe --quiet
t_interval($coeff(small),$stderr(small),$df,.95)
omit fe --test-only
modeltab add
modeltab show

/*---POE5 Example 7.11---*/
# checking using linear probability models
ols small const boy white_asian tchexper freelunch --robust
t_interval($coeff(const), $stderr(const), $df,.95)

# checking randomness using probit: see Chapter 16
probit small const boy white_asian tchexper freelunch
probit small const boy white_asian tchexper freelunch d

/*---POE5 Example 7.12---*/
# Differences in Differences Estimators

open "@workdir\data\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

# Example 7.13
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, $x_{ij}$ is the $i^{th}$ observation on the $j^{th}$ independent variable, $j = 2, 3, \ldots, k$, $e_i$ 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, $\beta_1$, $\beta_2$, …, $\beta_k$, 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 $\beta$s when the data are heteroskedastic are explored.

## 8.1 Food Expenditure Example

#### Example 8.1 in *POE5*

First, the simple linear regression model of food expenditures is estimated using least squares. The model is

$$\mathit{food\_exp}_i = \beta_1 + \beta_2 \mathit{income}_i + e_i \quad i = 1, 2, \ldots, n \tag{8.2}$$

where $\mathit{food\_exp}_i$ is food expenditure and $\mathit{income}_i$ 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 should use a consistent estimator of their standard errors 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 "@workdir\data\food.gdt"
ols food_exp const income
gnuplot food_exp income --fit=linear --output=display
```

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 there will be more variation around the regression line for some levels of income. The graph is shown in Figure 8.3 and this appears to be the case. There is significantly more variation in the data for high incomes than for low.

### 8.1.1 The `plot` block command

Before continuing with the examples from *POE5*, the `plot` command will be discussed and added to our repertoire of **gretl** tools. The plot block provides an alternative to the **gnuplot** command which may be more convenient when you are producing an elaborate plot (with several options and/or gnuplot commands to be inserted into the plot file).

A plot block starts with the command-word `plot` followed by the required argument that specifies the data to be plotted: this should be the name of a **list**, a **matrix**, or a **single time series**.

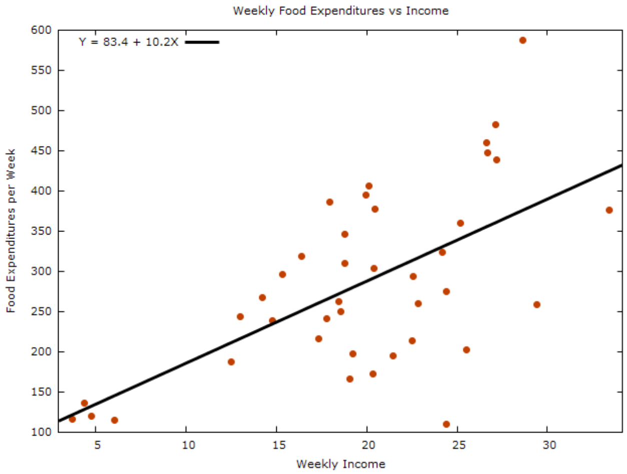


Figure 8.1: Food expenditure regression.

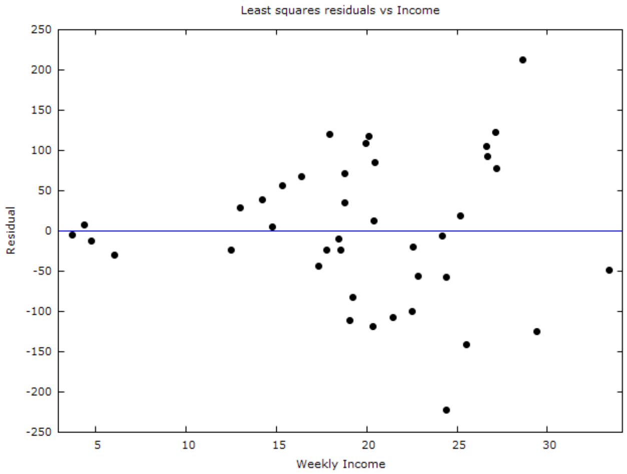


Figure 8.2: Food expenditure residuals

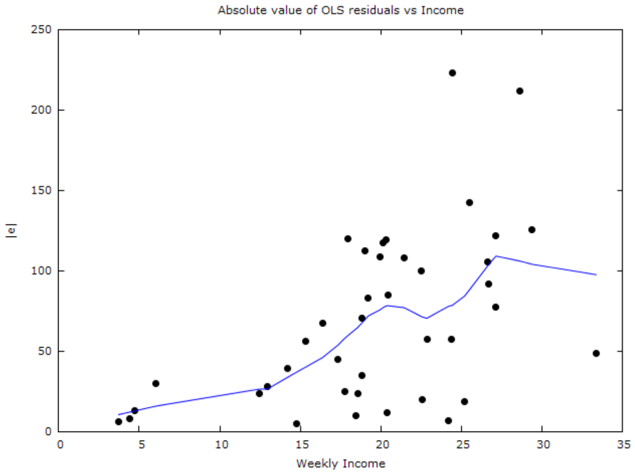


Figure 8.3: Absolute value of least squares residuals against income using with loess fit

If a list or matrix is given, the last element (list) or column (matrix) is assumed to be the x-axis variable and the other(s) the y-axis variable(s), unless the `--time-series` option is given in which case all the specified data go on the y axis.

The starting line may be prefixed with the `savename <-` to save a plot as an icon in the GUI program. The block ends with `end plot`.

The preceding plot is reconfigured using the plot block to present a more uniform appearance for this section. The code is listed below and then explained.

```
1  list plotlist = food_exp income
2  string title = "Weekly Food Expenditures vs Income"
3  string xname = "Weekly Income"
4  string yname = "Food Expenditures per Week"
5  g1 <- plot plotlist
6      options fit=linear
7      literal set linetype 1 lc rgb "dark-orange" pt 7
8      literal set linetype 2 lc rgb "black" lw 3
9      printf "set title \"%s\"", title
10     printf "set xlabel \"%s\"", xname
11     printf "set ylabel \"%s\"", yname
12 end plot --output=display
```

The data to be plotted can be a matrix or a list of series. In this example we plot two series, `food_exp` and `income`. These are placed into the `list` called `plotlist`. Three strings are created in lines 2-4, one for the title, one for the label on the x-axis and one for the label on y.

In **gnuplot** the series (or matrix columns) are given numbers starting at 1. In this example, `food_exp` is series 1. Within the body of the `plot` block the `literal` command is used to pass commands directly to **gnuplot** . Lines 7 and 8 do this. In line 7 the `linetype` for series 1 is set. `lc` stands for line color. Line color is expressed in `rgb` colors (red, green, blue) and is set to the color `dark-orange`. The default line markers is changed to points using `pt`, and the number `7` indicates the type of point to use. 7 corresponds to filled in dots.

The second thing plotted in this graph is the linear fit that was delivered by the **gretl** option in line 6. To change the line color to `black` and to make it wider than the default using `lw` (linewidth).

For help in selecting linewidths, colors, or point types you can launch a **gnuplot** session and type test at the prompt. This will yield the following (Figure 8.4) graph of available choices.

To determine the available colors in **gretl** type the following in the console:

```
eval readfile("@gretldir/data/gnuplot/gpcolors.txt")
```

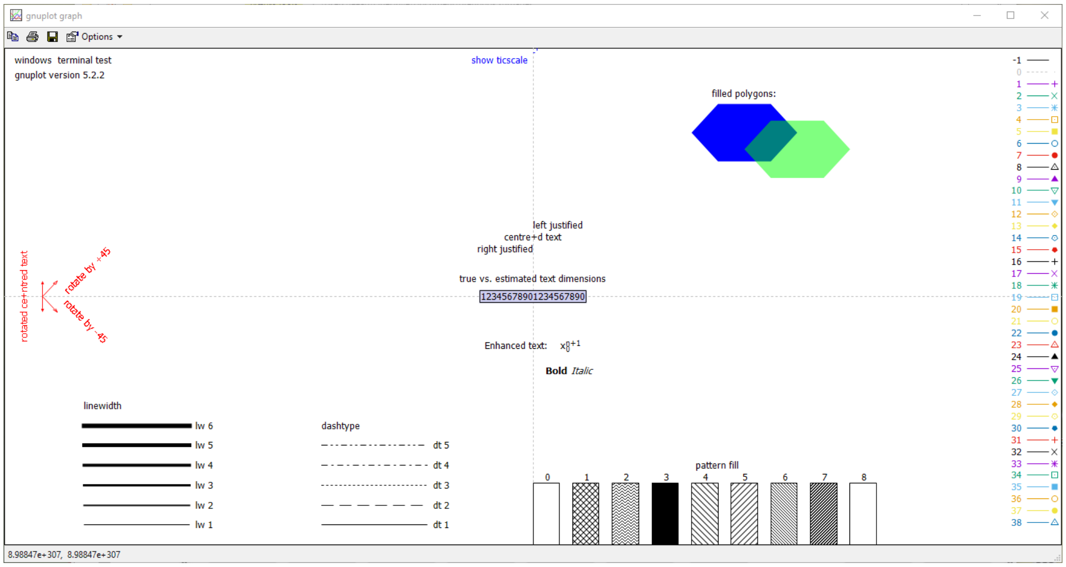


Figure 8.4: Type `test` at the **gnuplot** command prompt to reveal this handy cheat sheet documenting **gnuplot** options.

You can use the name of the color or its hex equivalent (preceded by a #). The contents of this file provide the translation to **gnuplot**, so it is easiest to use those.

The plot will be saved to a session as an icon labeled `g1`. The `plot` command block begins with `plot plotlist`. This is followed by some options (**gretl** options) and some commands that will be taken in for use in **gnuplot**. `literal` that the following command will be passed to **gnuplot** as is (i.e., literally). The `printf` commands are also passed literally to **gnuplot** as gnuplot `printf` commands.

There are two literal commands. The first sets the line type, changes the color of the dots, and changes the default point type markers to filled in dots (`pt 7`). The second literal suppresses the variable key label.

The `printf` commands print the previously defined strings to the title, x-axis and y-axis. The graph appears in Figure 8.5.

### 8.1.2 Robust Covariance Estimation

#### Example 8.2 in *POE5*

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.

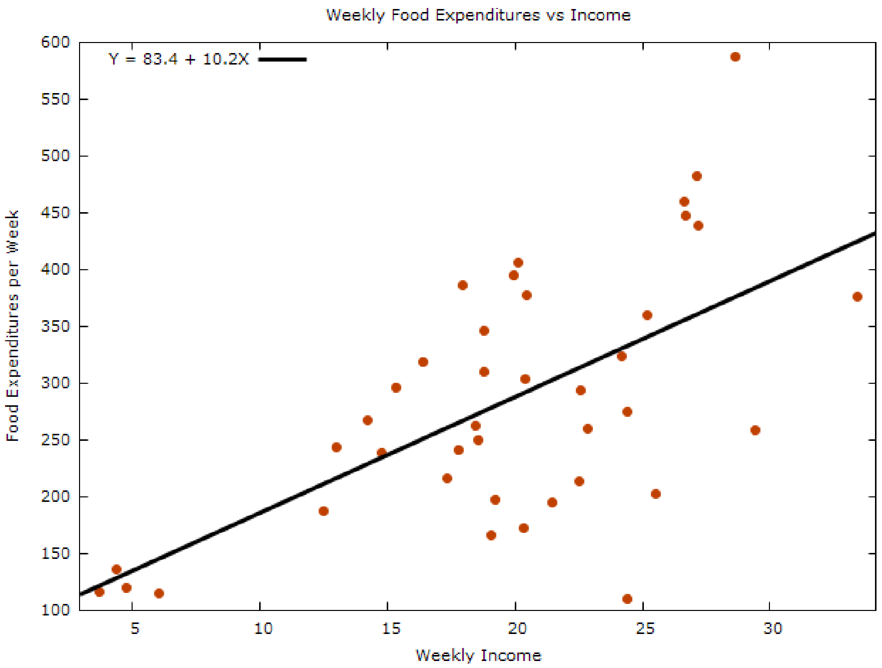


Figure 8.5: Plot of food expenditures against income with least squares fit.

```
ols food_exp const income --robust
t_interval($coeff(income),$stderr(income),$df,.95)
```

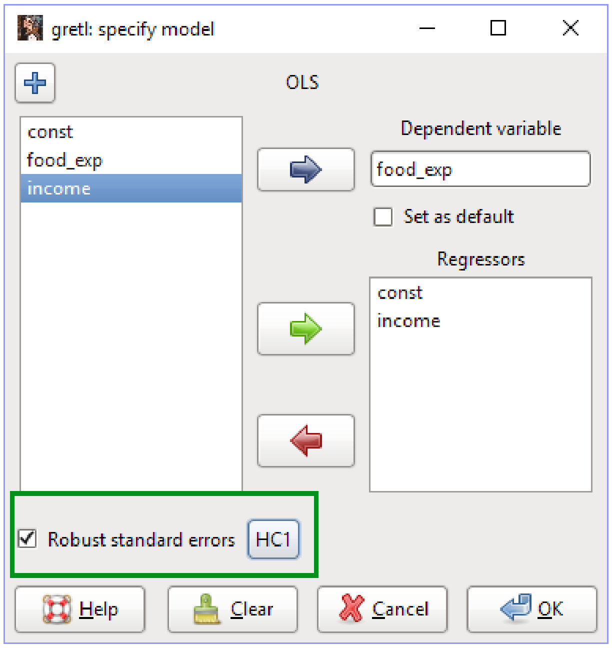
In the script, we have used the `t_interval` function to produce the interval. Remember, the degrees of freedom from the preceding regression are stored in `$df`. The first argument in the function indicates the desired distribution, and the last is the coverage probability of the confidence interval.

The script produces

```
The 95% confidence interval centered at 10.210 is (6.5474, 13.8719)
```

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 8.6 below. Note, the check box to generate 'robust standard errors' is circled. You will also notice that there is a button labeled **HC1** just to the right of the 'Robust standard errors' check box. Clicking this button reveals a dialog from which two options can be selected. One can choose to select from the available heteroskedasticity option or by cluster. The cluster option will be discussed later int this book. Select the first choice to reveal a preferences dialog box shown in Figure 8.7.

To reproduce the results in Hill et al. (2018), select HC1 (**gretl**'s default) from the pull-down list. As you can see, other **gretl** options can be selected here that affect the default behavior of



gretl: specify model

OLS

const
food_exp
income

Dependent variable
food_exp
☐ Set as default

Regressors
const
income

☑ Robust standard errors HC1

Figure 8.6: Check the box for (heteroskedasticity) robust standard errors.

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.

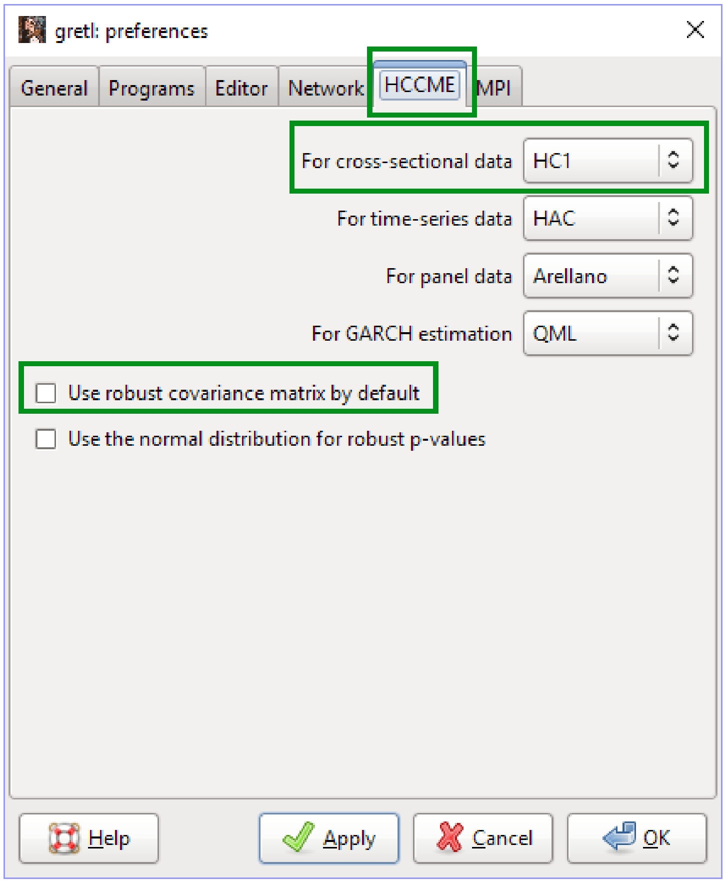
The model results for the food expenditure example appear in the table below. After estimating the model using the dialog, you can use **Analysis>Confidence intervals for coefficients** to generate 95% confidence intervals. Since you used the `robust` option in the dialog, these will be based on the 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
```

## 8.2 Detecting Heteroskedasticity using Residual Plots

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 8.8 is a plot of the least squares residuals against income. It appears that for larger



For cross-sectional data HC1

For time-series data HAC

For panel data Arellano

For GARCH estimation QML

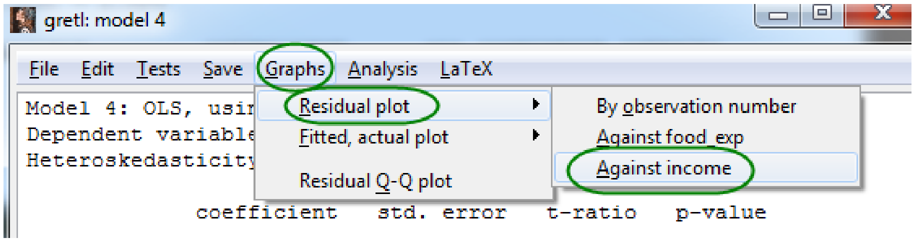
□ Use robust covariance matrix by default

□ Use the normal distribution for robust p-values

Figure 8.7: 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–HC1 in this case.

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, 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.

```
1  ols food_exp const income --robust
2  series res = $uhat
3  setinfo res -d "Least Squares Residuals" -n "Residual"
4
5  list plotmat = res income
6  string title = "Least squares residuals vs Income"
7  string xname = "Weekly Income"
8  string yname = "Residual"
9  g2 <- plot plotmat
10     options fit=linear
11     literal set linetype 1 lc rgb "black" pt 7
12     literal set nokey
13     printf "set title \"%s\"", title
14     printf "set xlabel \"%s\"", xname
15     printf "set ylabel \"%s\"", yname
16 end plot --output=display
```

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 `launch wgnuplot` and `load 'olsres.plt'`.

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"

```

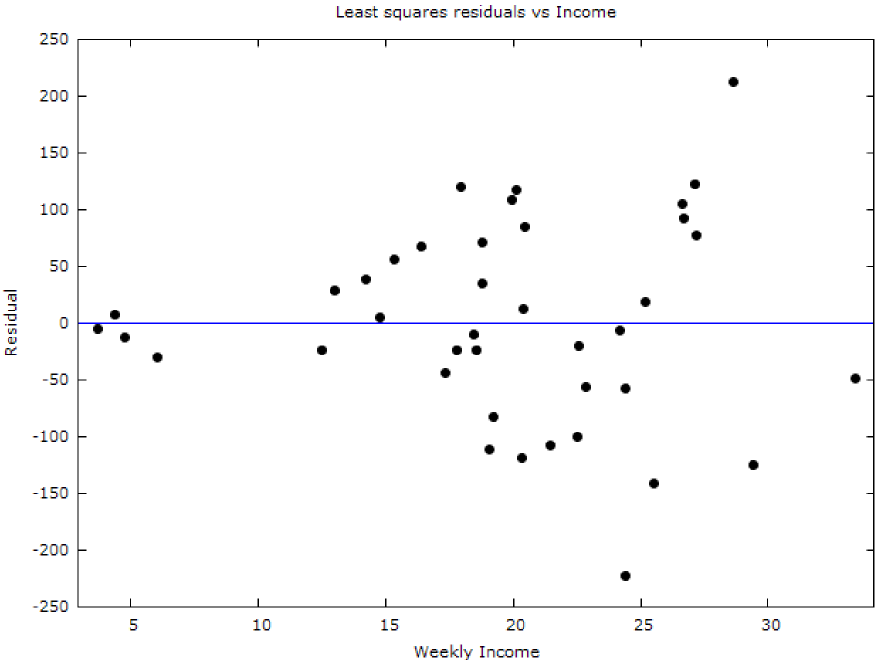


Figure 8.8: Plot of least squares residuals in the food expenditures model against income.

```
list plotmat = abs_e income
string title = "Absolute value of OLS residuals vs Income"
string xname = "Weekly Income"
string yname = "|e|"
g3 <- plot plotmat
    options fit=loess
    literal set linetype 1 lc rgb "black" pt 7
    literal set nokey
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display
```

The graph appears in Figure 8.9. 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 $y_i s$ only. The

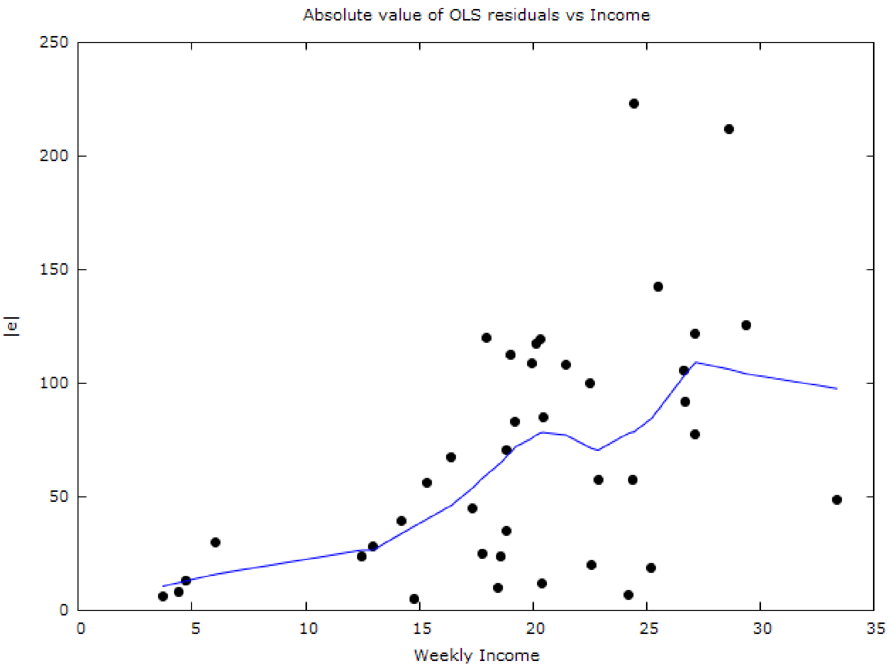


Figure 8.9: Plot of the absolute value of the food expenditures model residuals against income with loess fit.

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 8.9 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 Weighted Least Squares

### Example 8.3 in *POE5*

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 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 less weight and those containing more **signal** a higher weight. Reweighing the data in this way is known is referred to as **weighted least squares** (WLS). This descriptive term is the one used by **gretl** as well.

Suppose that the errors vary proportionally with $x_i$ according to

$$var(e_i) = \sigma^2 x_i \tag{8.3}$$

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

$$\frac{1}{\sqrt{x_i}} var(e_i) = \sigma^2 \tag{8.4}$$

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 shown below does this.

Create the weight (line 4), then call the function `wls` as in line 5. In the second part of the script, the data are transformed manually and the weighted data are used with OLS to produce the same result.

```
open "@workdir\data\food.gdt"

#GLS using built in function
series w = 1/income
wls w food_exp const income
t_interval($coeff(income),$stderror(income),$df,.95)

#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. This is followed by 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, the output of weighted least squares is interpreted 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 $\beta_2$ is (7.645, 13.257).

To gain some insight into the effect on the model's errors, plot the OLS residuals and the GLS residuals shown in Figure 8.10.

```
ols food_exp const income
series ehat = $uhat
wls w food_exp const income
series ehat_gls=$uhat/sqrt(income)

list plotmat = ehat_gls ehat income
string title = "GLS vs OLS residuals"
string xname = "Weekly Income"
string yname = "Residual"
g3 <- plot plotmat
    option single-yaxis
    literal set linetype 1 lc rgb "black" pt 7
    literal set key on
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display
```

Notice that the GLS residuals are divided by the $\sqrt{income}$ to reweigh them. The GLS residuals appear to be homoskedastic relative to OLS.

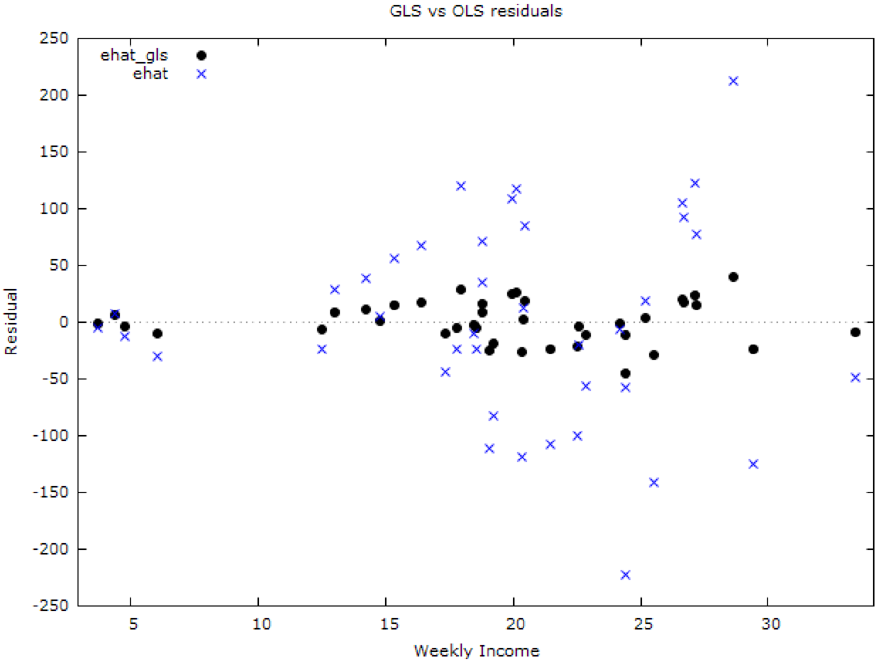


Figure 8.10: OLS and GLS residuals.

### 8.3.1 Heteroskedastic Model

A commonly used model for the error variance is the **multiplicative heteroskedasticity** model. It appears below in equation 8.5.

$$\sigma_i^2 = \exp\left(\alpha_1 + \alpha_2 z_i\right) \tag{8.5}$$

The variable $z_i$ is an independent explanatory variable that determines how the error variance changes with each observation. You can add additional $z$s if you believe that the variance is related to them (e.g., $\sigma_i^2 = \exp(\alpha_1 + \alpha_2 z_{i2} + \alpha_3 z_{i3})$). It's best to keep the number of $z$s relatively small. The idea is to estimate the parameters of (8.5) 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 (8.5) and adding a random error term, $v_i$, yields

$$\ln(\sigma_i^2) = \alpha_1 + \alpha_2 z_i + v_i \tag{8.6}$$

To estimate the $\alpha$s, 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 $z$s to the right-hand side of the model and estimate the $\alpha$s using least squares.

The regression model to estimate is

$$\ln(\hat{e}_i^2) = \alpha_1 + \alpha_2 z_i + v_i \tag{8.7}$$

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.

```
open "@workdir\data\food.gdt"
logs income
list x = const income
list z = const l_income

m1 <- ols food_exp x

# FGLS inconsistent for alpha
series lnsighat = log($uhat*$uhat)
ols lnsighat z
matrix alpha = $coeff
series predsighat = exp($yhat)
series w = 1/predsighat
m2 <- wls w food_exp const income
series ehat_fgls = $uhat/sqrt(predsighat)

#FGLS consistent for alpha
matrix alpha[1]=alpha[1]+1.2704
series wt = 1/exp(lincomb(z, alpha))
m3 <- wls wt food_exp x
```

The first four lines get the data set up for use; the data are loaded, natural log of income is added to the data, and two lists needed for the regression and the heteroskedasticity function are created.

Line 6 estimates the linear regression using least squares and saved to the session as an icon labelled m1.

Next, a new variable is generated (`lnsighat`) that is the natural log of the squared residuals from the preceding regression. Estimate the skedastic function using least squares and put the estimates from this regression into a matrix called, `alpha`. We do this because the least squares estimator of the intercept is biased and 1.2704 must be added to it to remove the bias (line 18). This isn't strictly necessary to get the correct parameter estimates and standard errors in the weighted regression. The weights are easily obtained using the `lincomb` function, which as seen elsewhere multiplies $z\alpha = \alpha_1 + \alpha_2 * \ln(\mathit{income})_i$. Remember, **gretl** automatically takes the square roots of `wt` for you in the `wls` function.

The output is:

Dependent variable: food_exp

| | (1)<br>OLS | (2)<br>WLS | (3)<br>WLS |
|---|---|---|---|
| const | 83.42* | 76.05** | 76.05** |
| | (43.41) | (9.713) | (9.713) |
| income | 10.21** | 10.63** | 10.63** |
| | (2.093) | (0.9715) | (0.9715) |
| $n$ | 40 | 40 | 40 |
| $\ell$ | −235.5 | −73.18 | −47.77 |

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

Comparing columns (2) and (3) one can see that having a biased estimator of $\alpha$ does not affect the estimates or standard errors. It does have a very small impact on the log-likelihood, however.

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.

### 8.3.2 Grouped Data

### Example 8.5 in *POE5*

This example, which uses the midwest subset of the *cps5_small.gdt* dataset, 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 earn a premium relative to those in rural areas (different intercept).

The sample is restricted to persons living in the midwest U.S. and summary statistics are computed for metro and rural areas.

```
open "@workdir\data\cps5_small.gdt"
# Use only metro observations
discrete metro
smpl midwest --restrict
summary wage educ exper --by=metro --simple
```

The `discrete` function is not strictly necessary here since the metro variable already carries this attribute. This is required because the summary statistics use the `--by=metro` option that requires the variable *metro* to be discrete.

The summary statistics are:

```
metro = 0 (n = 84):

                  Mean       Median         S.D.          Min          Max
wage             18.86        17.84        8.520        5.780        53.84
educ             13.99        13.00        2.263        8.000        20.00
exper            24.30        25.00        14.32        1.000        56.00

metro = 1 (n = 213):

                  Mean       Median         S.D.          Min          Max
wage             24.25        21.63        14.00        6.170        80.77
educ             14.25        14.00        2.771        3.000        21.00
exper            23.15        23.00        13.17       0.0000        52.00
```

Average wages in the metro areas are $24.25/hour and only $18.86 in rural areas.

Two regressions are estimated. The first is by OLS using robust standard errors. The second uses FGLS with the multiplicative model where $\ln(\hat{e}^2) = \alpha_1 + \alpha_2 metro$. Since metro is an indicator variable, heteroskedasticity will only take one of two values. Metro areas will have a different variance than rural ones.

```
# OLS w/robust std errors
m1 <- ols wage const educ exper metro --robust

# Multiplicative Heteroskedasticity FGLS
series lnsighat = log($uhat*$uhat)
series z = const metro
scalar alpha = $coeff(metro)
ols lnsighat z
series predsighat = exp($yhat)
series w = 1/predsighat
m2 <- wls w wage const educ exper metro
```

The session icons were added to a model table and the results are found below:

Dependent variable: wage

| | (1) | (2) |
|---|---|---|
| | OLS | WLS |
| const | $-18.45^{**}$ | $-16.97^{**}$ |
| | (4.023) | (3.788) |
| educ | $2.339^{**}$ | $2.258^{**}$ |
| | (0.2606) | (0.2391) |
| exper | $0.1890^{**}$ | $0.1747^{**}$ |
| | (0.04783) | (0.04472) |
| metro | $4.991^{**}$ | $4.996^{**}$ |
| | (1.159) | (1.214) |
| $n$ | 297 | 297 |
| $R^2$ | 0.2749 | 0.2815 |
| $\ell$ | $-1133$ | $-617.5$ |

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

One feature of these results is counterintuitive. Notice that the reported $R^2$ for WLS is larger than that of OLS. This is a consequence of using the generalized version discussed in section 4.2. Otherwise, the WLS estimates are fairly similar to OLS (as expected) and the estimated standard errors are a bit smaller, at least for slopes on education and experience.

## 8.4 Maximum Likelihood Estimation

The two-step estimation of the multiplicative heteroskedasticity model can be improved upon slightly by estimating the model via maximum likelihood. Maximum likelihood estimation of the model requires a set of starting values for the parameters that are easily obtained via the two-step estimator. The log-likelihood is:

$$\ln L = -\frac{n}{2}\ln 2\pi - \frac{1}{2}\sum_{i=1}^{n}\ln\sigma_i^2 - \frac{1}{2}\sum_{i=1}^{n}\frac{u_i^2}{\sigma_i^2} \tag{8.8}$$

where $\sigma_i^2 = \exp\{\alpha_1 + \alpha_2 * \ln(\text{income}_i)\}$ and $u_i$ are the residuals from the regression.

```
 1 # Assemble lists for x and z
 2 list z = const l_income
 3 list x = const income
 4 series y = food_exp
 5 
 6 # Starting values
 7 ols y x
 8 series lnsighat = ln($uhat^2)
 9 ols lnsighat z
10 matrix alpha = $coeff
11 
12 # MLE
13 mle loglik = -0.5 * ln(2*pi) - 0.5*zg - 0.5*e^2*exp(-zg)
14     series zg = lincomb(z, alpha)
15     series e = y - lincomb(x, beta)
16     params beta alpha
17 end mle
```

The first part of the script is basically the same as the one in the preceding section. The only change is that I placed the food_exp into a new series called y. This cosmetic change makes the mle block appear to be more general. It should work with any x, z, and y that has previously been properly populated.

The mle function operates on an observation by observation basis, hence there was no need to use $n$ and the summations from equation (8.8). The first series in line 14 is for the skedasticity function and the second, in line 15, gets the residuals. These are the only inputs we need for loglik defined in line 13 (provided you have defined the series x and z and provided starting values for the parameter vectors alpha and beta). As written, the routine uses numerical derivatives to search for the values that maximize the log-likelihood function. Analytical ones may be specified, which is sometimes useful. Here, the numerical ones work just fine as seen below.

The results are:

```
Using numerical derivatives
Tolerance = 1.81899e-012

Function evaluations: 68
Evaluations of gradient: 39

Model 11: ML, using observations 1-40
loglik = -0.5 * ln(2*pi) - 0.5*zg - 0.5*e^2*exp(-zg)
Standard errors based on Outer Products matrix

             estimate    std. error      z       p-value
  -------------------------------------------------------
  beta[1]    76.0728      8.39834      9.058    1.33e-019 ***
  beta[2]    10.6345      0.975438    10.90     1.12e-027 ***
  alpha[1]    0.468398    1.80525      0.2595   0.7953
  alpha[2]    2.76976     0.611046     4.533    5.82e-06  ***

Log-likelihood      -225.7152   Akaike criterion     459.4304
Schwarz criterion    466.1859   Hannan-Quinn         461.8730
```

You can see that these are very similar to the ones from weighted least squares.

One of the advantages of using this approach is that it yields a t-ratio for the hypothesis:

$$H_0 : \sigma_i^2 = \sigma^2$$
$$H_1 : \sigma_i^2 = \exp\{\alpha_1 + \alpha_2 \ln(\text{income}_i)\}$$

The alternative is specific as to the form of the heteroskedasticity (multiplicative) as well as the cause (ln(*income*). Because the model is estimated by maximum likelihood, the asymptotic distribution of the $t$-ratio is $N(0, 1)$. Gretl produces a $p$-value from this distribution in the output, which in this case is less than 0.05 and hence you can reject the null in favor of this specific alternative at that level of significance.

## 8.5 Detecting Heteroskedasticity using Hypothesis Tests

### 8.5.1 Goldfeld Quandt Test

Using examples from Hill et al. (2018) 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 $\sigma_1^2$ and that for the other subset is $\sigma_2^2$.

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:

$$F = \frac{\hat{\sigma}_1^2/\sigma_1^2}{\hat{\sigma}_2^2/\sigma_2^2} \sim F_{df_1,df_2} \tag{8.9}$$

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, $\sigma_1^2 = \sigma_2^2$,

$$GQ = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2} \sim F_{df_1,df_2} \tag{8.10}$$

This is just the ratio of the estimated variances from the two subset regressions.

### Grouped Data: Example 8.6 in *POE5*

In this example we return to the wage equations estimated using the *cps5_small.gdt* data. The Goldfeld-Quandt test compares variances of the metro and rural areas. Again, the sample is limited to observations from the midwest region of the U.S.

The data are loaded and the sample restricted to the midwest. The `--permanent` option is used, which substitute the restricted dataset for the original. Once the restricted sample is flagged as permanent, the `smpl full` command restores only the midwest subsample.

```
open "@workdir\data\cps5_small.gdt"
smpl midwest --restrict --permanent
```

Next, the metro subsample is estimated and its $\hat{\sigma}$ and degrees of freedom are saved. The midwest subsample is restored using smpl full and the rural subsample estimated. The GQ statistic is computed and the result returned.

```
smpl metro=1 --restrict        # Use only metro sample
ols wage const educ exper
scalar stdm = $sigma           # sighat metro
scalar df_m = $df              # metro df

smpl full                      # Restore the full sample
smpl metro=0 --restrict        # Use only rural observations
```

```
ols wage const educ exper
scalar stdr = $sigma          # sighat rural
scalar df_r = $df             # rural df

scalar gq = stdm^2/stdr^2     # GQ statistic

scalar crit1 = critical(F, df_m, df_r, .025)
scalar crit2 = 1/critical(F, df_r, df_m, .025)

printf "\nThe F(%d, %d) statistic = %.3f.\n\
  The left 0.025 critical value is %.4g\n\
  The right 0.025 critical value is %.3f\n",df_m,df_r,gq,crit2,crit1
```

This results in:

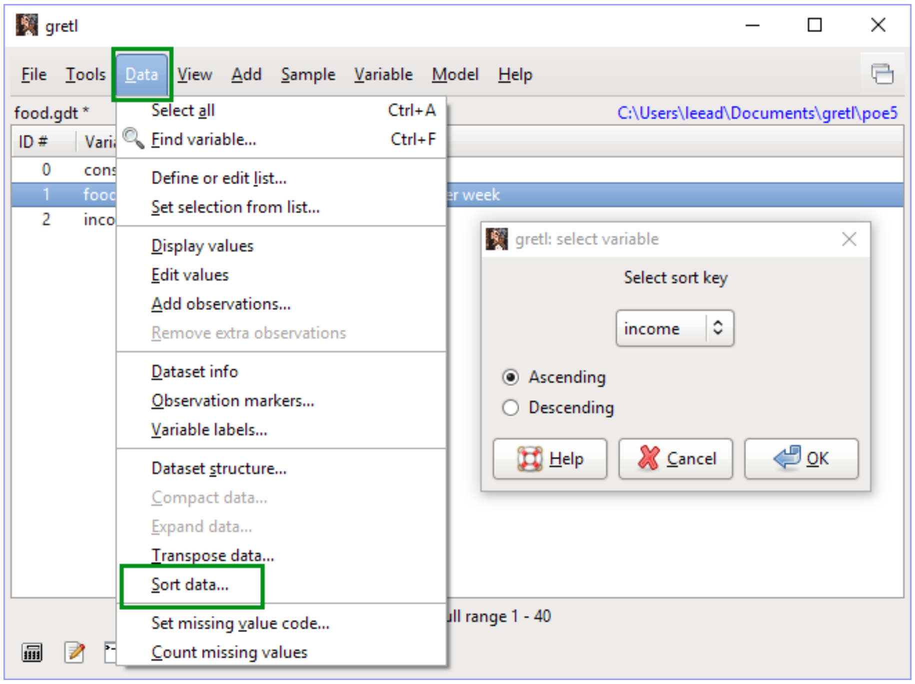
```
The F(210, 81) statistic = 2.603.
   The left 0.025 critical value is 0.7049
   The right 0.025 critical value is 1.461
```

The GQ statistic is in the right-hand rejection region of this test and we conclude that the data are heteroskedastic at the 5% level.

### Food Expenditures: Example 8.7 in *POE5*

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 8.11. 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 "@workdir\data\food.gdt"
dataset sortby income
list x = const income

smpl 21 40              # large variance observations
ols food_exp x
scalar stdL = $sigma    # sighat large variance
scalar df_L = $df       # df large variance subset

smpl 1 20               # small variance observations
ols food_exp x
scalar stdS = $sigma    # sighat small variance
scalar df_S = $df       # df small variance subset
gq = stdL^2/stdS^2      # GQ statistic
```



Select sort key

income

● Ascending

○ Descending

Figure 8.11: Select **Data>Sort data** from the main menu bar to reveal the dialog box shown on the right side of this figure. Choose the desired sort key and indicate whether you want to sort in ascending or descending order.

```
15
16 printf "\nThe F(%d, %d) statistic = %.3f. The right\
17 side p-value is %.4g.\n",df_L,df_S,gq,pvalue(F, df_L, df_S, gq)
```

This yields:

```
The F(18, 18) statistic = 3.615. The right side p-value is 0.004596.
```

The `dataset sortby` command is used 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. 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.

[1] Replace `sortby income` with `dsortby income` to sort the sample by income in descending order.

### 8.5.2 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

$$\begin{aligned} H_0 &: \sigma_i^2 = \sigma^2 \\ H_1 &: \sigma_i^2 = h(\alpha_1 + \alpha_2 z_{i2} + \dots \alpha_s z_{iS}) \end{aligned}$$

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 $z_{is}$, $s = 2, 3, \dots, S$. These variables are exogenous and correlated with the model's variances. The function $h(\cdot)$, 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 $z_{is}$, $s = 2, 3, \dots, S$.
5. Compute $nR^2$ from this regression and compare it to the $\alpha$ 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.

To facilitate this more restrictive formulation of a BP test a short program is given to make computing it quite simple.

```
function void BP_test (series y, list xvars, list zvars)
    ols y xvars --quiet
    series ehat_2 = $uhat^2
    ols ehat_2 zvars --quiet
    scalar pval = pvalue(X,nelem(zvars)-1,$trsq)
    printf "Z-Variables:  %s", varname(zvars)
    printf "\nBreusch-Pagan test: nR2 = %.3f\
    p-value = %.3f \n", $trsq, pval
end function
```

The function is called BP_test and it takes three inputs. The first is a series for the dependent variable of the model. The second is a list of the regression's independent variables, including a constant. The third is a list of variables that cause heteroskedasticity in the tests alternative hypothesis.

The operation of the function should be obvious. The model is estimated and the squared residuals put into a series. Line 4 estimates the auxiliary regression for the BP test using the variables in zvars. The $p$-value is computed and everything is printed to the screen, including the variables in $z$.

Usage is simple.

```
list xvars = const income
list zvars = const income
BP_test(food_exp, xvars, yvars)
```

This produces:

```
Z-Variables: const,income
Breusch-Pagan test: nR2 = 7.384     p-value = 0.007
```

This confirms both of the computations above.

### 8.5.3 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 ($z_{is}$, $s = 2, 3, \ldots, S$) include each non-redundant regressor, its square, and all cross products between regressors. See *POE5* 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 $\alpha$ 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: nR^2 = 7.555079,
with p-value = P(Chi-square(2) > 7.555079) = 0.022879
```

The homoskedasticity null hypothesis is rejected at the 5% level.

Note, our `BP_test` function can be used as well, although there is no need to do so. In fact, if the regressor list is long, it would be tedious to assemble the variable list for `zvars`.

```
list xvars = const income
list zvars = const income sq_income  # all vars, squares, and cross-prods
BP_test(food_exp, xvars, zvars)
```

```
Breusch-Pagan test: nR2 = 7.555     p-value = 0.023
```

It matches. The key is to include each variable, its square (if unique), and cross-products in the list of variables for the heteroskedasticity function. With only a constant and a continuous variable in the model that amounts to a constant, income, and income squared.

### 8.5.4 Variance Stabilizing Transformation

#### Example 8.8 in *POE5*

In this example a simple model of household entertainment expenditures is estimated and tested for heteroskedasticity using the Breusch-Pagan test. In this section, we propose a simple function that will compute the test and report the outcome to the display.

The model to be estimated is:

$$entert_i = \beta_1 + \beta_2 income_i + \beta_3 college_i + \beta_4 advanced_i + e_i$$

The sample is censored to include only those with positive entertainment expenditures. The independent variables are monthly income in \$100, and indicator for highest degree is Bachelor's, and an indicator equal to 1 if the highest degree is masters/professional/PhD.

Frequency plots of the data in levels and natural logs appear below in Figures 8.12 and 8.13. It is clear that entertainment levels is highly skewed and that taking the logarithms produces a more even distribution. Breusch-Pagan tests are conduced with $z_i = 1, income_i$.

```
open "@workdir\data\cex5_small.gdt"
smpl entert>0 --restrict
logs entert
g1 <- freq entert --plot=display --silent
g2 <- freq l_entert --plot=display --silent

list xvars = const income college advanced
list zvars = const income
BP_test(entert, xvars, zvars)
BP_test(l_entert,xvars,zvars)
```

The results of the BP test show:

```
Z-Variables:  const,income
Breusch-Pagan test: nR2 = 31.337     p-value = 0.000

Z-Variables:  const,income
Breusch-Pagan test: nR2 = 0.355     p-value = 0.551
```

The null hypothesis of no heteroskedasticity due to income in the levels model is rejected and not rejected in the log-linear model.

## 8.6 Heteroskedasticity in the Linear Probability Model

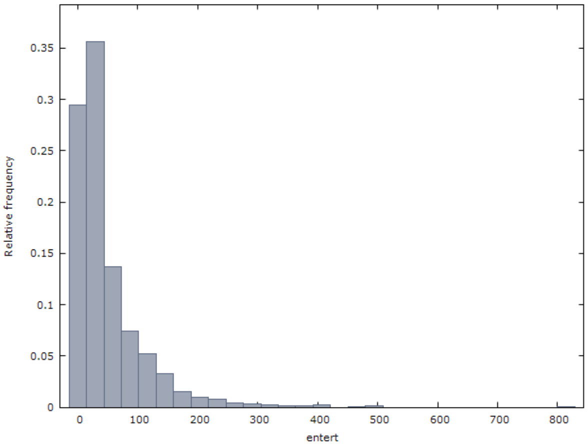


Figure 8.12: Levels of household entertainment expenditures.

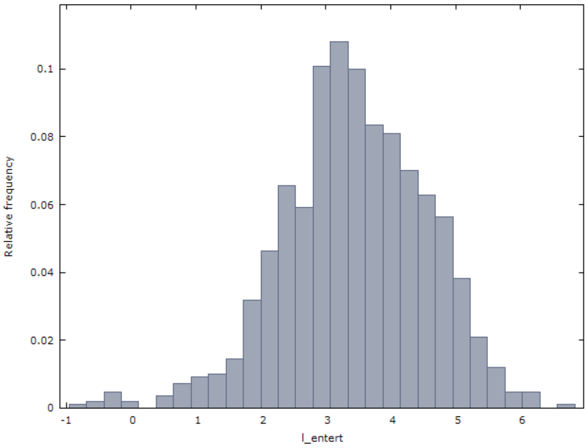


Figure 8.13: Natural Log of household entertainment expenditures.

### Example 8.9 in *POE5*

The linear probability model was introduced in Chapter 7. It was shown that the indicator variable, $y_i$ is heteroskedastic. That is,

$$var(y_i) = \pi_i(1 - \pi_i) \tag{8.11}$$

where $\pi_i$ is the probability that the dependent variable is equal to 1 (the choice is made). The estimated variance is

$$\widehat{var(y_i)} = \hat{\pi}_i(1 - \hat{\pi}_i) \tag{8.12}$$

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 "@workdir\data\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:

```
                    Mean      Median        S.D.         Min         Max
coke              0.4474      0.0000      0.4974      0.0000       1.000
pr_pepsi           1.203       1.190      0.3007      0.6800       1.790
pr_coke            1.190       1.190      0.2999      0.6800       1.790
disp_pepsi        0.3640      0.0000      0.4814      0.0000       1.000
disp_coke         0.3789      0.0000      0.4853      0.0000       1.000
pratio             1.027       1.000      0.2866      0.4972       2.325
```

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.

$$p_t = \begin{cases} b(0.01) + p(1-b) = 0.01 & \text{when } b = 1 \\ b(0.01) + p(1-b) = p & \text{when } b = 0 \end{cases} \tag{8.13}$$

There is another, less transparent, way to generate the truncated probabilities: use the ternary conditional assignment operator first discussed in section 2.8.3. This operates like an if statement and can be used to save a line of script. This syntax would create the series as

```
series pt = ( (p<.01) || (p>.99) ) ?  0.01 : p
```

The bound condition in parentheses (`p` $<$ `.01`)||(`p` $>$ `.99`) is checked: that is what the question mark represents. If the condition 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) | (2) | (3) | (4) |
|---|---|---|---|---|
|  | OLS | OLS | WLS | 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 |
| $\ell$ | −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.

## 8.7 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 heteroskedasticity 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 several variations of the robust sets of standard errors. Based on these, 95% confidence intervals are computed.

Start by estimating the food expenditure model using least squares and add the estimates, which are saved as icons to the session, to a model table. Reestimate the model using the `--robust` option and store the results as icons. Open the session icon view, drag the models to the model table and open it for viewing.[2]

```
open "@workdir\data\food.gdt"
list xvars = const income
Incorrect <- ols food_exp xvars --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
set hc_version 1
HC1 <- ols food_exp xvars --robust --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
set hc_version 2
HC2 <- ols food_exp xvars --robust --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
set hc_version 3
HC3 <- ols food_exp xvars --robust --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
```

The model table,

OLS estimates
Dependent variable: food_exp

| | (Incorrect) | (HC1) | (HC2) | (HC3) |
|---|---|---|---|---|
| const | 83.42* | 83.42** | 83.42** | 83.42** |
| | (43.41) | (27.46) | (27.69) | (28.65) |
| income | 10.21** | 10.21** | 10.21** | 10.21** |

[2] Or, you could use the `modeltab` commands in the script.

| | | | | |
|---|---|---|---|---|
| | (2.093) | (1.809) | (1.823) | (1.886) |
| $n$ | 40 | 40 | 40 | 40 |
| $R^2$ | 0.3850 | 0.3850 | 0.3850 | 0.3850 |
| $\ell$ | −235.5 | −235.5 | −235.5 | −235.5 |

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 across the columns, but that the estimated standard errors are different. 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. Using this option forces subsequent Wald tests based on least squares estimates to use the HCCME for computation. This ensures that results from omit or restrict will be statistically valid under heteroscedasticity when the preceding regression is estimated with the `--robust` flag.

The confidence intervals are computed using the `t_interval` program supplied with this manual. The results:

```
The 95% confidence interval centered at 10.210 is (5.9721, 14.4472)
The 95% confidence interval centered at 10.210 is (6.5474, 13.8719)
The 95% confidence interval centered at 10.210 is (6.5183, 13.9010)
The 95% confidence interval centered at 10.210 is (6.3913, 14.0280)
```

which refer to models ‘Incorrect’, ‘HC1’, ‘HC2’ and ‘HC3,’ respectively.

## 8.8 Monte Carlo simulation of OLS, GLS and FGLS

There are five different designs in this simulation. All are based on a linear model

$$y = 5 + x_2 + 0 * x_3 + e$$

The heteroskedasticity function is

$$h(x_2) = 3\exp(1 + \alpha x_2)/\bar{h}$$

The heteroskedasticity is controlled via $\alpha$, which can be 0, 0.3, and 0.5. Sample sizes are either 100 or 5000. In the simulation below, $\bar{h}$ has been computed and is referred to as “eta”.

For instance, the design that produces column 1 in Table 8E.1 of *POE5* is

```
# * table 8e column 1
# n = 100
# het = 0
# eta = 2.7182818
```

Sample size is set to 100, het ($\alpha$) is zero–no heteroskedasticity, and eta is 2.7182818. These are set in lines 32, 34, and 36 in the script. As an exercise, see if you can replicate (at least get close to the results in *POE5*). Hint: I did!

```
1  # Appendix 8E in POE5
2  # Setting the parameters for the simulation
3  
4  # * table 8e column 1
5  # n = 100
6  # het = 0
7  # eta = 2.7182818
8  
9  # * 8e column 2
10 # n = 100
11 # het = .3
12 # eta = 6.938608
13 
14 # * 8e column 3
15 # n = 100
16 # het = .5
17 # eta = 13.8982
18 
19 # * table 8e column 4
20 # n = 5000
21 # het = .5
22 # eta = 14.25737
23 
24 # * table 8e column 5
25 # n = 5000
26 # het = .5
27 # eta = 6.025252
28 
29 # Set the sample size and save it in n
30 set hc_version 3
31 # Set the values of the parameters
32 nulldata 5000
33 scalar n = $nobs
34 scalar het = 0.5
35 scalar sigma = 1
36 scalar eta = 6.025252
37 
38 # set a seed if you want to get same results each time you run this
39 set seed 1234567
```

```

# generate n observations on x2 and x3
# series x2 = uniform(1,5)
# series x3 = uniform(1,5)

# start the loop, indicating the desired number of samples.
loop 1000 --progressive --quiet
    series x2 = uniform(1,5)             # comment out if holding x2 const
    series x3 = uniform(1,5)             # comment out if holding x3 const
    # generate variances that depend on x2
    series  sig = 3*(exp(1+het*x2 + 0*x3)/eta)
    # generate normal errors
    series u = sig*normal()
    # generate sample of y
    series y = 5+x2+0*x3 + u
    # run the regression with usual error
    ols y const x2 x3
    # save the estimated coefficients
    scalar b1 = $coeff(const)
    scalar se1 = $stderr(const)

    scalar b2 = $coeff(x2)
    scalar se2 = $stderr(x2)

    scalar b3 = $coeff(x3)
    scalar se3 = $stderr(x3)

    # run OLS regression with HC std errors
    ols y const x2 x3 --robust
    scalar robse1 = $stderr(const)
    scalar robse2 = $stderr(x2)
    scalar robse3 = $stderr(x3)

    # BP test
    series ehat2=$uhat^2
    ols ehat2 const x2 x3
    scalar tr2 = $trsq
    scalar reject = (pvalue(X,2,tr2) < 0.05)

    # FGLS
    series ln_ehat2=ln(ehat2)
    ols ln_ehat2 const x2 x3
    series h0 = 1/exp($coeff(x2)*x2 + $coeff(x3)*x3)
    wls h0 y const x2 x3

    scalar b1_fgls = $coeff(const)
    scalar se1_fgls = $stderr(const)

    scalar b2_fgls = $coeff(x2)
    scalar se2_fgls = $stderr(x2)

    scalar b3_fgls = $coeff(x3)
    scalar se3_fgls = $stderr(x3)

# gls for b2 only
    series h = 1/x2
    wls h y const x2 x3
    scalar b2_gls = $coeff(x2)
    scalar se2_gls = $stderr(x2)

# gls robust for b2 only
    wls h y const x2 x3 --robust
    scalar robse2_gls = $stderr(x2)

    print b1 se1 b2 se2 b3 se3 robse1 robse2 robse3 \
      reject b1_fgls se1_fgls b2_fgls se2_fgls b3_fgls\
      se3_fgls b2_gls se2_gls robse2_gls
endloop
```

## 8.9 Script

```
set echo off
# function estimates confidence intervals based on the t-distribution
function void t_interval (scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# Breusch-Pagan test
function void BP_test (series y, list xvars, list zvars)
    ols y xvars --quiet
    series ehat_2 = $uhat^2
    ols ehat_2 zvars --quiet
    scalar pval = pvalue(X,nelem(zvars)-1,$trsq)
    printf "Z-Variables: %s", varname(zvars)
    printf "\nBreusch-Pagan test: nR2 = %.3f\
    p-value = %.3f \n", $trsq, pval
end function

open "@workdir\data\food.gdt"
m1 <- ols food_exp const income
gnuplot food_exp income --fit=linear --output=display

list plotmat = food_exp income
string title = "Weekly Food Expenditures vs Income"
string xname = "Weekly Income"
string yname = "Food Expenditures per Week"
g1 <- plot plotmat
    options fit=linear
    literal set linetype 1 lc rgb "black" pt 7
    literal set nokey
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

# ols with HCCME standard errors
ols food_exp const income --robust
# confidence intervals (Robust)
t_interval($coeff(income),$stderr(income),$df,0.95)

# residual plot
ols food_exp const income --robust
series res = $uhat
setinfo res -d "Least Squares Residuals" -n "Residual"

list plotmat = res income
string title = "Least squares residuals vs Income"
string xname = "Weekly Income"
string yname = "Residual"
g2 <- plot plotmat
    options fit=linear
    literal set linetype 1 lc rgb "black" pt 7
    literal set nokey
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

# launch gnuplot (Windows only)
launch wgnuplot
# To view graph, type: load ’olsres.plt’ 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"

list plotmat = abs_e income
string title = "Absolute value of OLS residuals vs Income"
string xname = "Weekly Income"
string yname = "|e|"
g3 <- plot plotmat
    options fit=loess
    literal set linetype 1 lc rgb "black" pt 7
    literal set nokey
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

# Example 8.3 WLS
#GLS using built in function
open "@workdir\data\food.gdt"
logs income
list x = const income
list z = const l_income

ols food_exp const income
series ehat = $uhat
series w = 1/income

wls w food_exp const income
series ehat_gls=$uhat/sqrt(income)
t_interval($coeff(income),$stderr(income),$df,0.95)

list plotmat = ehat_gls ehat income
string title = "GLS vs OLS residuals"
string xname = "Weekly Income"
string yname = "Residual"
g3 <- plot plotmat
    option single-yaxis
    literal set linetype 1 lc rgb "black" pt 7
    literal set key on
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

#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

# Example 8.4
# heteroskedastic model
# OLS
m1 <- ols food_exp x

# FGLS inconsistent for alpha
series lnsighat = log($uhat*$uhat)
ols lnsighat z
matrix alpha = $coeff
series predsighat = exp($yhat)
series w = 1/predsighat
```

```
m2 <- wls w food_exp const income
series ehat_fgls = $uhat/sqrt(predsighat)

# Fix alpha
matrix alpha[1]=alpha[1]+1.2704
series wt = 1/exp(lincomb(z, alpha))
m3 <- wls wt food_exp x

# Plot gls and fgls residuals
list plotmat = ehat_gls ehat_fgls income
string title = "GLS and FGLS residuals"
string xname = "Weekly Income"
string yname = "Residual"
g4 <- plot plotmat
    option single-yaxis
    literal set linetype 1 lc rgb "black" pt 7
    literal set key on
    literal set size square
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

# Example 8.5
# Heteroskedastic Partition
#Wage Example
open "@workdir\data\cps5_small.gdt"
ols wage const educ exper metro
# Use only metro observations
discrete metro
smpl midwest --restrict
summary wage educ exper --by=metro --simple

m1 <- ols wage const educ exper metro --robust

series lnsighat = log($uhat*$uhat)
list z = const metro
scalar alpha = $coeff(metro)
ols lnsighat z
series predsighat = exp($yhat)
series w = 1/predsighat
m2 <- wls w wage const educ exper metro

# Example 8.6
# Goldfeld Quandt
# grouped data--Goldfeld-Quandt
open "@workdir\data\cps5_small.gdt"
smpl midwest --restrict --permanent
smpl metro=1 --restrict       # Use only metro sample
ols wage const educ exper
scalar stdm = $sigma          # sighat metro
scalar df_m = $df              # metro df

smpl full                      # Restore the full sample
smpl metro=0 --restrict        # Use only rural observations
ols wage const educ exper
scalar stdr = $sigma           # sighat rural
scalar df_r = $df              # rural df

scalar gq = stdm^2/stdr^2      # GQ statistic

scalar crit1 = critical(F, df_m, df_r, .025)
scalar crit2 = 1/critical(F, df_r, df_m, .025)

printf "\nThe F(%d, %d) statistic = %.3f.\n\
  The left 0.025 critical value is %.4g\n\
  The right 0.025 critical value is %.3f\n",df_m,df_r,gq,crit2,crit1
# Example 8.7
# Goldfeld-Quandt for food expenditure
open "@workdir\data\food.gdt"
dataset sortby income
list x = const income

smpl 21 40            # large variance observations
ols food_exp x
scalar stdL = $sigma  # sighat large variance
scalar df_L = $df     # df large variance subset

smpl 1 20             # small variance observations
ols food_exp x
scalar stdS = $sigma  # sighat small variance
scalar df_S = $df     # df small variance subset
gq = stdL^2/stdS^2    # GQ statistic

printf "\nThe F(%d, %d) statistic = %.3f. The right\
side p-value is %.4g.\n",df_L,df_S,gq,pvalue(F, df_L, df_S, gq)

# BP test for food_exp model
list xvars = const income
list zvars = const income
BP_test(food_exp, xvars, zvars)

# White’s 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

list xvars = const income
list zvars = const income sq_income  # vars, squares, and X-products
```

```
BP_test(food_exp, xvars, zvars)

ols food_exp const income --quiet
modtest --white --quiet

# Example 8.8
# Variance Stabilizing Transformation
open "@workdir\data\cex5_small.gdt"
smpl entert>0 --restrict
logs entert
g1 <- freq entert --plot=display --silent
g2 <- freq l_entert --plot=display --silent

list xvars = const income college advanced
list zvars = const income
BP_test(entert, xvars, zvars)
BP_test(l_entert,xvars,zvars)

# Example 8.9
open "@workdir\data\coke.gdt"
summary --simple
list xvars = const pratio disp_coke disp_pepsi
# OLS
ols coke xvars
modeltab add
# OLS w/robust
ols coke xvars --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 xvars
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 xvars
modeltab add
modeltab show

ols coke xvars --quiet
modtest --white --quiet

```

```
# Example 8.10
# Alternative HCCME
open "@workdir\data\food.gdt"
list xvars = const income
m1 <- ols food_exp xvars --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
set hc_version 1
m2 <- ols food_exp xvars --robust --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
set hc_version 2
m3 <- ols food_exp xvars --robust --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
set hc_version 3
m4 <- ols food_exp xvars --robust --quiet
t_interval($coeff(income),$stderr(income),$df,0.95)
```

# Chapter 9

# Regression with Time-Series Data: Stationary Variables

In this chapter 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 must be stationary in order for the usual econometric procedures to have the desired statistical properties. This requires that the means, variances and covariances of the time-series data not 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.

The first diagnostic tool used when considering a new time series is to construct a simple of the data against time. A time-series plot may reveal potential problems with the data and suggest ways to proceed statistically. As seen earlier, graphs and plots are simple to generate in **gretl** and a few new tricks will be explored below.

Finally, since this chapter deals with time series, 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 in panels, both $n$ and $T$ will be 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 your data do not have a proper date to identify the time period in which the observations were recorded, 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, you should know how to add this information yourself and this is shown below. You need to identify that the data are time series, specify their frequency of observation, and 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.

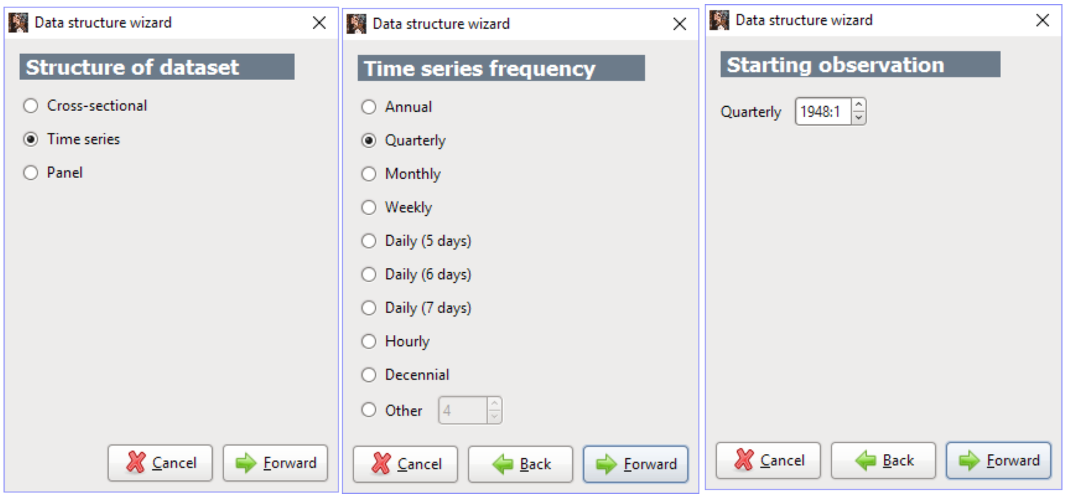
Before getting to the specific examples from the text, something should be said about how **gretl** handles dates and times.

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 or 1997q1. 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 as comprehensive 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 and there are functions included that will assist in converting whatever format you may have to something that **gretl** understands as a date.

There are two methods of getting your dataset to be recognized 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. These work well if there are no missing time periods in your dataset.

The first dialog defines the dataset 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. Check to see whether the data start and stop when expected. If so, then the data structure is almost certainly correct. If the end date is something other than expected, then go back and try again. There may be 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 this right, otherwise you may end up having to redo you analysis. Figure 9.1 shows the first three dialog boxes for defining a time-series structure. The last box (Figure 9.2) confirms



Structure of dataset

○ Cross-sectional
● Time series
○ Panel

Time series frequency

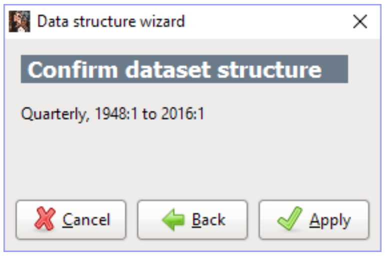
○ Annual
● Quarterly
○ Monthly
○ Weekly
○ Daily (5 days)
○ Daily (6 days)
○ Daily (7 days)
○ Hourly
○ Decennial
○ Other 4

Starting observation

Quarterly 1948:1

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.

that the series starts in 1948:1 and ends in 2016:1.

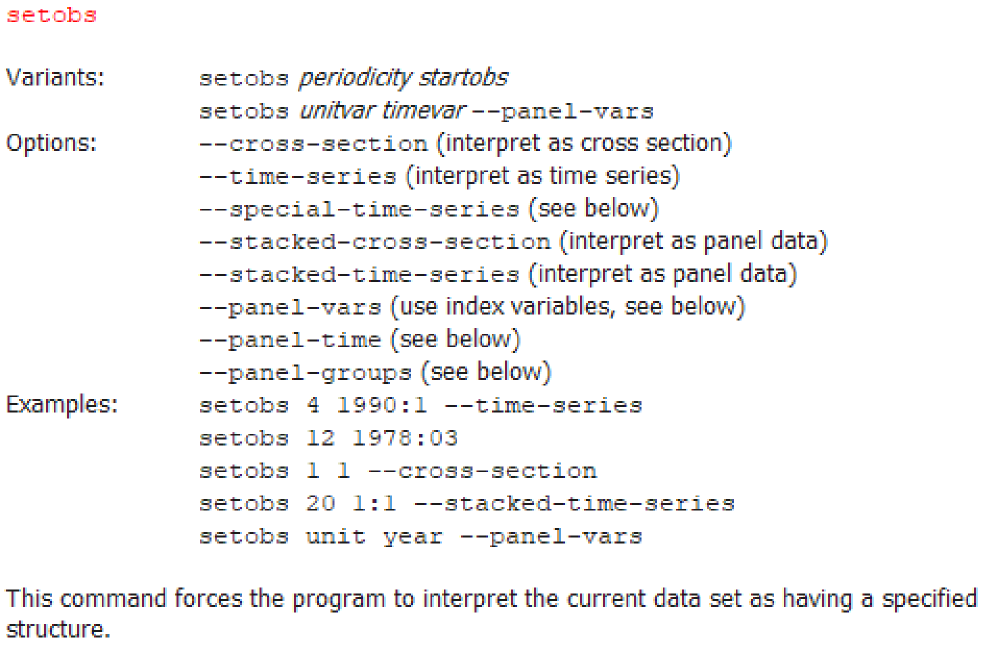


Confirm dataset structure

Quarterly, 1948:1 to 2016:1

Figure 9.2: Check the confirmation box to be sure the expected time periods are given.

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 --panel-vars
Options:      --cross-section (interpret as cross section)
              --time-series (interpret as time series)
              --special-time-series (see below)
              --stacked-cross-section (interpret as panel data)
              --stacked-time-series (interpret as panel data)
              --panel-vars (use index variables, see below)
              --panel-time (see below)
              --panel-groups (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
```

This command forces the program to interpret the current data set as having a specified structure.

Define the desired periodicity and the date at which 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.

| **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

## 9.2 Time-Series Plots

Gnuplot handles all the plotting in **gretl**. Gretl includes some functions that communicate with **gnuplot**, which makes generating simple graphs very easy to do. In section 8.1.1 the `plot` command was discussed that provides a bridge from **gretl** to **gnuplot** that can help you to enhance the basic output by giving you direct access to **gnuplot** commands. It is worth taking a look at that if your time series plotting needs are not met with the basic graphs.

On the other hand, if you have something really fancy to plot, you could use **gnuplot** directly to get the desired result. However, the `literal` commands provided by the `plot` block can likely accomplish what you want. Still, this requires some knowledge of the scripting language in **gnuplot**. This is most important when generating publication quality graphics. For diagnostic purposes the basic graphs are very good and the **gretl** graph editing commands available through the GUI will handle most needs quite well. All-in-all, **gretl**'s graphical interface that works with

**gnuplot** is easy to use and powerful.

Gretl's time-series plot is really just an $XY$ scatter plot that uses time as the x-axis variable. By default it uses the `--lines` option to connect the data points. It's relatively primitive, but can be edited to improve its appearance. 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 4.22) 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. Additional editing of these graphs require a trip through **gnuplot**. You can, however, you can save them in several formats. Examples of this are found below.

### Example 9.1 in *POE5*

In this example time-series graphs are plotted for the U.S. unemployment rate and GDP growth from 1948.1 to 2016.1. The data are found in the *usmacro.gdt* data file.

```
open "@workdir\data\usmacro.gdt"
# change variable attributes
setinfo g -d "% 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"
setinfo inf -d "U.S. Inflation Rate\
(%change CPI, seasonally adjusted) " -n "Inflation Rate"

gnuplot g --with-lines --time-series --output=display
gnuplot u --with-lines --time-series --output=display
```

Here, the `setinfo` command is used to add meaningful labels for the y-axis variable (`-n`). The graphs are generated using the simple `gnuplot` command with the desired options.

The two plots, after some editing using **gretl** plot controls, are shown in Figures 9.3 and 9.4. The graphs can be combined using the GUI by choosing **View>Multiple graphs>Time-series**. The result appears in Figure 9.5.

The plot command can be used to replicate this.

```
g1_simple <- plot u
    options time-series with-lines
end plot --output=display
```

This is no more complicated than using `gnuplot`. The advantage is that its output can be assigned to an icon and sent to the current session. Not only does this make it available for further editing in **gretl**, you can also open the **gnuplot** script by right-clicking on the icon and choosing **Edit plot commands**.

You can add titles and labels to the plot block as shown here, where unemployment and GDP growth are plotted in separate graphs:

```
string title = "U.S. Quarterly unemployment rate"
string xname = "Year"
string yname = "Unemployment Rate"
g1 <- plot u
    options time-series with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

string title = "U.S. GDP growth rate"
string xname = "Year"
string yname = "Quarterly GDP growth rate"
g2 <- plot g
    options time-series with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display
```

The strings '`title`' and '`xname`' help to label the graphs informatively. The **gretl** options handle plotting a single series against time (`time-series`) and lines (`with-lines`). Notice that these are **gretl** options, but appear without the usual flag `--`.

The **gretl** command to generate multiple series in multiple graphs is

```
g3 <- scatters g u
```

An advantage of using the `scatters` command is that its output can be sent to the current session as an icon using the assignment operator. In this example it is assigned to `g3`. The output from scatters can be seen in Figure 9.5 below.

## 9.3 Serial Correlation in a Time-Series

Correlations measure the strength of linear association between two variables. When there is no linear association, the covariance between the variables is zero and consequently so is the correlation. In time-series data, observations located near one another may be correlated. Correlation of this kind is called **autocorrelation** (sometimes, serial correlation).[1]

time-series samples are obviously not random draws from a population of individuals. They are specifically ordered in time and cannot be reshuffled without losing information about how the variables evolve or change over time. A useful tool in determining how to parameterize these relationships is the **correlogram**. The correlogram is a graph of a time series sample autocorrelation functions against the time lags.

A common assumption made in he classical multiple linear regression model (5.1) is that the observations not be correlated with one another. While this is certainly believable if the sample is drawn randomly, it's less likely if one has drawn observations sequentially in time. time-series observations, which are usually drawn at regular intervals, often embody a structure where time is an important component. If this structure cannot be adequately modeled 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.

### Example 9.2 in *POE5*

For visual evidence of autocorrelation series can be plotted against lagged values. If there is serial correlation, you should see some sort of positive or negative relationship between the series. Below (Figure 9.6) is a plot for the U.S. unemployment rate. Clearly, there is a positive relationship between $u$ and its lagged value.

Better evidence can be obtained by looking at the correlogram. A **correlogram** is simply a plot of a series' sample autocorrelations. The $s^{th}$ order sample autocorrelation for a series $y$ is the

[1] Auto=self, serial=adjacent

correlation between observations that are $s$ periods apart (equation 9.1). The formula is

$$r_s = \frac{\sum_{t=s+1}^{T}(y_t - \bar{y})(y_{t-s} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2} \tag{9.1}$$

In **gretl** the command that computes and graphs autocorrelations and partial autocorrelations is `corrgm`. This command prints the values of the autocorrelation function (ACF) for a series, which may be specified by name or number.

The partial autocorrelations (PACF, calculated using the Durbin-Levinson algorithm) are also shown: these are net of the effects of intervening lags. In addition the Ljung-Box Q statistic is printed. This may be used to test the null hypothesis that the series is "white noise"; it is asymptotically distributed as $\chi^2$ with degrees of freedom equal to the number of lags used.

By default, a plot of the correlogram is produced: a **gnuplot** graph in interactive mode or an ASCII graphic in batch mode. This can be adjusted via the `--plot` option.

The `corrgm` command plots a number of these against lags. The syntax to plot 24 autocorrelations of the series `g` is

```
corrgm g 24
```

which yields the plot in Figure 9.7. The correlogram is the plot at the top and the partial autocorrelations 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` command generates a matrix return. There is an option to use Bartlett standard errors for computing the confidence bands, `--bartlett`.

A script to generate the first 12 default intervals is

```
matrix ac = corrgm(u, 12)
matrix lb = ac[,1]-1.96/sqrt($nobs)
matrix ub = ac[,1]+1.96/sqrt($nobs)
matrix all = lb~ac[,1]~ub
cnameset(all, "Lower  AC Upper ")
printf "\nAutocorrelations and 95%% confidence intervals\n %9.4f\n", all
```

The intervals generated are:

```
Autocorrelations and 95% confidence intervals
    Lower        AC     Upper
   0.3885    0.5072    0.6258
   0.2500    0.3686    0.4872
   0.0303    0.1489    0.2676
  -0.0338    0.0848    0.2034
  -0.1430   -0.0244    0.0942
  -0.0658    0.0529    0.1715
  -0.0208    0.0979    0.2165
   0.0077    0.1264    0.2450
   0.1012    0.2198    0.3384
   0.1132    0.2318    0.3505
   0.0671    0.1857    0.3043
  -0.0405    0.0781    0.1967
```

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 cnameset 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 *POE5* for more guidance.

### Example 9.3 in *POE5*

In this example the correlogram for the GDP growth series is generated for 45 lags.

```
corrgm g 45 --plot=display
```

The correlogram is shown in Figure 9.8. The first four rows of the output show:

```
Autocorrelation function for g
***, **, * indicate significance at the 1%, 5%, 10% levels
using standard error 1/T^0.5

  LAG      ACF           PACF          Q-stat. [p-value]

    1    0.5072  ***   0.5072 ***     70.9939  [0.000]
    2    0.3686  ***   0.1499 **     108.6314  [0.000]
    3    0.1489   **    -0.1185 *       114.7996  [0.000]
    4    0.0848          0.0055         116.8060  [0.000]
```

The first 3 autocorrelations are significantly different from zero at 5%.

## 9.4 Forecasting

Forecasting the values of economic variables is an important activity for firms, individuals, and governments. Forecasting can also provide feedback on the quality of the model or its estimators. In this brief section the fcast command is used to forecast out-of-sample using AR(2) and ARDL(2,1) models.

### Example 9.5 and 9.6 in *POE5*

First, the AR(2) model of unemployment is considered.

$$u_t = \delta + \theta_1 u_{t-1} + \theta_2 u_{t-2} + e_t$$

The model is estimated using OLS using the available sample in *usmacro.gdt*, which ends in 2016:1. Out-of-sample forecasts are generated for the three subsequent periods, 2016:2-2016:4. To make this possible, 3 empty observations must be added to the sample before using the `fcast` command to generate dynamic forecasts of future unemployment. The script is:

```
open "@workdir\data\usmacro.gdt"
m1 <- ols u const u(-1 to -2)
dataset addobs 3
fcast 2016:2 2016:4 --dynamic
```

The observations are added using the `dataset addobs` command. The `fcast` beginning and ending periods must be given and the `--dynamic` option issued. The results are:

```
 For 95% confidence intervals, t(268, 0.025) = 1.969

                     u    prediction    std. error        95% interval

2016:2                      4.88089     0.294702      4.30067 -   5.46112
2016:3                      4.91629     0.559250      3.81521 -   6.01737
2016:4                      4.98602     0.799577      3.41177 -   6.56028
```

This matches the output in Table 9.3 of*POE5*.

### Example 9.7 in *POE5*

In this example one distributed lag term is added to the model.

$$u_t = \delta + \theta_1 u_{t-1} + \theta_2 u_{t-2} + \delta_1 g_{t-1} + e_t$$

The out-of sample forecast for $u_{t+1}$ depends on $g_t$ which is available from the sample. Out-of-sample forecasts for $u_{t+2}$ and $u_{t+3}$ are conditional on given values of $g_{t+1}$ and $g_{t+2}$. These values must be added to the dataset before dynamic forecasts can be generated using `fcast`.

There are at least three ways to do this. 1) The observations could be added from another dataset using `append`. For two observations this is not worth the trouble. 2) Highlight the series `g` and right-click. Choose **Edit values** to open the series for editing. Scroll to the bottom and add the desired values of $g$ to observations as pictured in 9.9. Set $g_{2016:2} = 0.869$ and $g_{2016:3} = 1.069$. And the easiest way is to use the indexing command as shown in lines 1 and 2 below. Notice that the indexing recognizes the actual dates you want to fill.

That done, use `fcast` just as done in the AR(2) example and as shown below:

```
1 series g[2016:2]=.869
2 series g[2016:3]=1.069
3 ols u const u(-1 to -2) g(-1)
4 fcast 2016:2 2016:4 --dynamic
```

```
For 95% confidence intervals, t(267, 0.025) = 1.969

                 u    prediction    std. error        95% interval

2016:2                   4.94987     0.291923      4.37511 -   5.52463
2016:3                   5.05754     0.534339      4.00549 -   6.10959
2016:4                   5.18395     0.743022      3.72102 -   6.64688
```

Once again, these match the values in *POE5*.

## 9.5 Model Selection

In ARDL models time is an important part of the model's specification. Economic theory suggests that policy changes take time to reach their full effect, but theory is silent about how long it will take. Omitting relevant lags creates bias and overspecifying lags creates inefficiency. Hence, lags in any ARDL model must be chosen wisely. Model selection rules, like those discussed in 6.4, are often used for this purpose.

For this task, we reuse the `modelsel` function from section 6.4.3, with a minor modification that will suppress most of the printing. Knowing $p$ and $q$ tells us everything we need to know about the regressors in the model; it is unnecessary to list them separately.

The modified function is shown below:

```
function matrix modelsel_np (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, $rsq, rbar2, aic, bic}
    return A
end function
```

This differs from `modelsel` by two lines of code (the `printf` statements were removed). The function is renamed `modelsel_np` for model selection no print.

The function is executed within a loop that increments over $q = 0, 1, \cdots, 8$ and $p = 1, 2, \cdots, 8$. The model selection statistics are collected into a matrix, the columns given names, and printed to the screen.

```
open "@workdir\data\usmacro.gdt"
smpl 1950:1 2016:1
matrix A = {}
loop p = 1..8  --quiet
    loop q = 0..8 --quiet
         if q==0
            list xvars = u(-1 to -p) const
         else
            list xvars = u(-1 to -p) g(-1 to -q) const
         endif
        matrix a = p~q~modelsel_np(u, xvars)
        matrix A = A | a
    endloop
endloop
cnameset(A,"p q k n R2 Adj_R2 AIC SC ")
matrix B = msortby(A,8)
printf "Model Selection in ARDL\n %8.4g\n",B[1:6,]
```

A selection of results appear below:

```
Model Selection in ARDL
       p       q       k       n      R2  Adj_R2     AIC      SC
       2       0       3     265  0.9687  0.9684  -2.454  -2.414
       2       1       4     265  0.9691  0.9688  -2.462  -2.408
       2       3       6     265  0.9704  0.9698  -2.488  -2.407
       2       4       7     265  0.9709  0.9702  -2.497  -2.403
       3       0       4     265  0.9689  0.9686  -2.456  -2.402
       3       1       5     265  0.9694  0.9689  -2.462  -2.395
       2       5       8     265  0.9712  0.9704  -2.500  -2.392
```

Even with suppression of some printing in the "no print" version of modelsel the matrix A would produce a lot of output. So the matrix sort function (msortby) is used to sort by column 8 (the SC criterion) and the first six rows are printed. The AR(2) model minimizes SC and the ARDL(2,1) is a close runner-up.

## 9.6 Granger Causality test

### Example 9.8 in *POE5*

In this example we test to determine whether GDP growth Granger causes unemployment. In the context of an ARDL(2,1) model

$$u_t = \delta + \theta_1 u_{t-1} + \theta_2 u_{t-2} + e_t$$

This amounts to a test of the null hypothesis $\delta_1 = 0$ against the alternative $\delta_1 \neq 0$. The $t$-ratio from the regression table and the corresponding $p$-value are sufficient to test this.

Alternatively, one could use the omit command as done below.

```
open "@workdir\data\usmacro.gdt"
ols u(0 to -2) g(-1) const
omit g_1
```

This is verified below in Figure 9.10 You can see that the $p$-value from the $t$-ratio is the same as that for the $F$-statistic. We reject the hypothesis that g Granger causes u at the 5% level.

When there are more variables involved in the test, one could use lists to clean things up. Put the lagged values of u in one list and the lagged values of g in another. Then, the omit statement can be applied to the entire list. For instance, in an ARDL(2,4) model we would have:

```
list ulags = u(-1 to -2)
list glags = g(-1 to -4)

smpl 1949:1 2016:1
ols u ulags glags const --quiet
omit glags --test-only
```

This produces the following result:

```
Null hypothesis: the regression parameters are zero for the variables
  g_1, g_2, g_3, g_4
Test statistic: F(4, 262) = 5.98095, p-value 0.000127776
```

Again, g Granger causes u.

## 9.7 Serial Correlation in Residuals

### Example 9.10 in *POE5*

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. In the first example, the residuals from the ARDL(2,1) model are examined using their correlogram. The entire sample is used to estimate the model.

```
smpl full
m1 <- ols u u(-1 to -2) g(-1) const
series residual = $uhat
g1 <- corrgm residual 24 --plot=display
```

The estimated model is:

$$\widehat{\mathrm{u}} = \underset{(0.07228)}{0.3616} + \underset{(0.05555)}{1.533}\ \mathrm{u\_1} - \underset{(0.05559)}{0.5818}\ \mathrm{u\_2} - \underset{(0.01949)}{0.04824}\,\mathrm{g\_1}$$

$$T = 271 \quad \bar{R}^2 = 0.9681 \quad F(3, 267) = 2734.5 \quad \hat{\sigma} = 0.29192$$

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

The 24 autocorrelations are shown below in Figure 9.11 Three of the autocorrelations (7, 8, 17) lie outside the 95% confidence bounds.

### Example 9.11 in *POE5*

The previous example is repeated using the residuals from an ARDL(1,1) model. Estimation of the ARDL(1,1) yields:

$$\widehat{\mathrm{u}} = \underset{(0.08416)}{0.4849} + \underset{(0.01280)}{0.9628}\ \mathrm{u\_1} - \underset{(0.01871)}{0.1672}\ \mathrm{g\_1}$$

$$T = 272 \quad \bar{R}^2 = 0.9555 \quad F(2,269) = 2910.2 \quad \hat{\sigma} = 0.34538$$

(standard errors in parentheses)

The first 24 autocorrelations are shown below in Figure 9.12. The first three autocorrelations lie outside the 95% confidence bounds, which is often taken as strong evidence of autocorrelation among the residuals.

## 9.8 Tests for Autocorrelation

Another way to determine whether or not your residuals are autocorrelated is to use an *LM* (Lagrange multiplier) test. The null hypothesis of this test is no autocorrelation. The alternative is that the errors are either autoregressive of order $k$ or are a moving average of $k$ random errors MA($k$).

$$\begin{array}{lrcl} \text{AR(2)} & e_t & = & \rho_1 e_{t-1} + \rho_2 e_{t-2} + v_t \\ \text{MA(2)} & e_t & = & \phi_1 v_{t-1} + \phi_2 v_{t-2} + v_t \end{array}$$

where $v_t$ is white noise.

The test is based on an auxiliary regression where lagged least squares residuals are added to the original regression equation. The parameter $k$ determines the order of the AR or MA process under the alternative and to conduct the test $k$ lags of residuals should be added to the auxiliary regression. If the coefficient on the lagged residual is significant (or when $k > 1$, if the lagged residuals are jointly significant) then you conclude that the model is autocorrelated.

For example, suppose you want to test the residuals of the model $y_t = \beta_1 + \beta_2 x_t + e_t$ for autocorrelation. The null hypothesis is $H_0$: no autocorrelation and the alternative is $H_1$: MA(2) or AR(2). Estimate the regression model using least squares and save the residuals, $\hat{e}_t$. Add two lags of the residuals to the model and run the following regression.

$$\hat{e}_t = \beta_1 + \beta_2 x_t + \delta_1 \hat{e}_{t-1} + \delta_2 \hat{e}_{t-2} + v_t$$

Compute $TR^2$ which is distributed $\chi^2(2)$ if $H_0$ is true.

### Example 9.12 in *POE5*

The residuals of the ARDL(1,1) and ARDL(2,1) model of unemployment are tested using the *LM* test. Fortunately, **gretl** includes a command that computes several model tests for autocorrelation, including the *LM* test discussed above.

The `modtest` syntax is:

```
modtest [order] --autocorr
```

The command takes an input (order) which refers to the number $k$ in either AR($k$) or MA($k$) process.

In this example orders $k = 1, 2, 3, 4$ are each tested. This is easily done in a loop. The script to accomplish this is:

```
ols u u(-1) g(-1) const
    loop i=1..4
        modtest $i --autocorr --quiet
    endloop

ols u u(-1 to -2) g(-1) const
    loop i=1..4
        modtest $i --autocorr --quiet
    endloop
```

The first loop is for the ARDL(1,1) model and the second for the ARDL(2,1) model.

These loops produce a good bit of output, but the statistics found in table 9.6 of *POE5* are reproduced. An example of `modtest 1 --autocorr` for the ARDL(2,1) model is:

```
Breusch-Godfrey test for first-order autocorrelation

Test statistic: LMF = 2.466115,
with p-value = P(F(1,266) > 2.46611) = 0.118

Alternative statistic: TR^2 = 2.489391,
with p-value = P(Chi-square(1) > 2.48939) = 0.115

Ljung-Box Q' = 1.1404,
with p-value = P(Chi-square(1) > 1.1404) = 0.286
```

The `modtest` command sets the lagged values of residuals that would otherwise be missing to zero.

For instance, in the AR(2)/MA(2) example, $\hat{e}_{t-1} = 0$ and $\hat{e}_{t-2} = 0$ in the auxiliary regressions. Keep this in mind if you try to replicate the modtest computations.

The statistic named *LMF* actually performs an *F*-test of the no autocorrelation hypothesis based upon the auxiliary regression where $\hat{e}_{t-1} = 0$ and $\hat{e}_{t-2} = 0$. With only one autocorrelation parameter this is equivalent to the square of the *t*-ratio. The next test is the *LM* test, i.e., $TR^2 = 2.489391$ 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 *POE5* exactly.

If you prefer to use the dialogs, then estimate the model using least squares in the usual way (**Model>Ordinary least squares**). In the models window 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.

This example shows the relative strength of the *LM* test. One can use it to test for any order of autocorrelation due to either autoregressive or moving average errors. Other tests, like the Durbin-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.

## 9.9 Case Studies

In this section we analyze several models through examples. The examples include Okun's Law, Phillips curve, and estimation of a consumption function.

### 9.9.1 Okun's Law

#### Example 9.13 in *POE5*

Okun's Law provides another opportunity to search for an adequate specification of the time-series model. In this model, the change in the unemployment rate depends on deviation of actual from normal growth. If the economy grows faster than normal, the unemployment rate will drop.

$$u_t - u_{t-1} = -\gamma(g_t - g_N)$$

where $g_N$ is the long-run normal level of economic growth.

Let $u_t - u_{t-1} = du_t$, $\beta_0 = -\gamma$, and $\alpha = \gamma g_N$; add an error term and the regression model is

$$du_t = \alpha + \beta_0 g_t + e_t$$

Recognizing that some time will pass before growth deviations have their full impact we get a distributed lag model:

$$du_t = \alpha + \beta_0 g_t + \beta_1 g_{t-1} + \cdots + \beta_q g_{t-q} + e_t$$

The data are found in *okun5_aus.gdt*. They are loaded into **gretl** and the difference of the unemployment rate is added to the dataset. The `setinfo` command is used to change attributes so as to improve labelling. A scatter plot is made, with lines, and output to the display. It is also saved to the current session as an icon.

```
open "@workdir\data\okun5_aus.gdt"
diff u
setinfo g -n "GDP growth rate"
setinfo d_u -d "Change in Australian Civilian Unemployment \
  Rate  (Seasonally adjusted)" -n \
  "Change in Unemployment Rate"
g4 <- scatters g d_u --with-lines --output=display
```

The figures produced by this script are found in Figure 9.13.

Two finite distributed lag models are estimated. The first has a lag length $q = 5$ and the second $q = 4$. The results are collected into a model table:

OLS estimates
Dependent variable: d_u

| | (1) | (2) |
|---|---|---|
| const | 0.3930** | 0.4100** |
| | (0.04493) | (0.04155) |
| g | −0.1287** | −0.1310** |
| | (0.02556) | (0.02440) |
| g_1 | −0.1721** | −0.1715** |
| | (0.02488) | (0.02395) |
| g_2 | −0.09320** | −0.09400** |
| | (0.02411) | (0.02402) |
| g_3 | −0.07260** | −0.07002** |
| | (0.02411) | (0.02391) |
| g_4 | −0.06363** | −0.06109** |
| | (0.02407) | (0.02384) |
| g_5 | 0.02317 | |
| | (0.02398) | |
| $n$ | 148 | 149 |
| $\bar{R}^2$ | 0.4816 | 0.4813 |
| $\ell$ | 13.82 | 13.83 |

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

The lag weight on $g_5$ is not statistically different from zero and we choose the model with $q = 4$. In addition, the `modelsel_np` function is used to compute model selection rules for $q \leq 6$. The results confirm our choice.

```
Model Selection in ARDL
       p       q       K       N      R2  Adj_R2     AIC      SC
       4       0       6     145  0.5062  0.4884   -2.93  -2.806
       3       0       5     145  0.4835  0.4687  -2.899  -2.796
```

Finally, a loop is used to recursively compute impact and delay multipliers based on the estimated model.

### 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

$$y_t = \alpha + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_q x_{t-q} + e_t \tag{9.2}$$

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**

$$\frac{\partial E(y_t)}{\partial x_{t-s}} = \beta_s \tag{9.3}$$

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 an **interim multiplier**. The interim multiplier simply adds the immediate effect (impact multiplier), $\beta_0$, 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 $\sum_{s=0}^{q} \beta_s$.

In terms of the estimated model of Okun's law we assemble the multipliers using:

```
1  open "@workdir\data\okun5_aus.gdt"
2  diff u
3  ols d_u g(0 to -4) const
4  matrix b = $coeff
5  matrix mult = zeros(5,2)
6  loop i=1..5
7      matrix mult[i,1]=b[i+1]
8      matrix mult[i,2]=b[i+1]
9      if i>1
10     matrix mult[i,2]=mult[i-1,2]+b[i+1]
11     endif
12 endloop
13 cnameset(mult,"Delay Interim")
14 rnameset(mult,"0 1 2 3 4")
15 printf "Multipliers for Okun's Law, q=4\n%10.4f\n", mult
16
17 printf "\nNormal Growth rate = %.4f%% per quarter\n", -b[1]/mult[5,2]
18 printf "\nThe Total Multiplier = %.4f\n", mult[5,2]
```

This script if relatively straightforward. The data are reloaded and the differences added to the dataset. In line 3 the regression is estimated and its coefficients are saved as a vector (`matrix`) b in line 4. The vector b contains all of the $s$-period delay multipliers (i.e., the lag weights) in our linear finite distributed lag model.

In line 5 a $5 \times 2$ matrix of zeros is created to hold the multipliers we compute iteratively. The loop commences in line 6. Lines 7 and 8 take care of the impact multiplier and 10 computes the remaining ones. The normal growth rate is calculated in line 17 and the total multiplier is computed in 18.

The results are:

```
Multipliers for Okun's Law, q=4
       Delay   Interim
0    -0.1310   -0.1310
1    -0.1715   -0.3025
2    -0.0940   -0.3965
3    -0.0700   -0.4665
4    -0.0611   -0.5276

Normal Growth rate = 0.7770% per quarter
The Total Multiplier = -0.5276
```

The ARDL model adds lagged values of the dependent variable to the AR model,

$$y_t = \delta + \theta_1 y_{t-1} + \cdots + \theta_p y_{t-p} + \delta_0 x_t + \delta_1 x_{t-1} + \cdots + \delta_q x_{t-q} + v_t \tag{9.4}$$

and this makes the multiplier analysis a little harder. The model must first 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.

For the ARDL(1,1) that contains a linear trend we have

$$\Delta y_t = y_t - y_{t-1} = \delta + \theta_1(x_t - x_{t-1}) + v_t = \delta + \theta_1 \Delta x_t + v_t$$

Written with the lag operator, L

$$(1 - \theta_1 L)\, \Delta y_t = \delta + (\delta_0 + \delta_1 L)\, x_t + v_t$$

$$\begin{aligned}
\Delta y_t &= (1 - \theta_1 L)^{-1}\delta + (1 - \theta_1 L)^{-1}\,(\delta_0 + \delta_1 L)\, x_t + (1 - \theta_1 L)^{-1} v_t \\
\Delta y_t &= \alpha + \beta_0 x_t + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \beta_3 x_{t-3} + \cdots + e_t \\
&= \alpha + \left(\beta_0 + \beta_1 L + \beta_2 L^2 + \beta_3 L^3 + \cdots\right) x_t + e_t
\end{aligned}$$

This is just an infinite distributed lag model. The coefficients for the multipliers involve the $\beta$ coefficients, which must be solved for in terms of the estimated parameters of the ARDL. The solutions for the ARDL(1,1) are

$$\begin{aligned}
\beta_0 =& \delta_0 && (9.5) \\
\beta_1 =& \delta_1 + \beta_0 \theta_1 && (9.6) \\
\beta_j =& \beta_{j-1}\theta_1 \quad \text{for } j \geq 2 && (9.7)
\end{aligned}$$

Although the computation of the multipliers is fairly transparent, it involves a lot of code. Gretl contains two functions that can simplify this and make it much more general. The revised script[2] is:

```
open "@workdir\data\okun5_aus.gdt"
diff u

list exo = const
p = 0
q = 4
list exo = const
horizon = 4

ols d_u exo g(0 to -q)
k = nelem(exo)
matrix b = $coeff[k+1:k+p]
matrix a = $coeff[k+p+1:k+p+q+1]

mult = filter(1|zeros(horizon, 1), a, null)
mult = mult ~ cum(mult)

```

[2] Adapted from one provided to the author by Jack Lucchetti.

```
cnameset(mult,"Delay Interim")
rnameset(mult,"0 1 2 3 4")
printf "Multipliers for Okun's Law, q=4\n%10.4f\n", mult
printf "\nNormal Growth rate = %.4f%% per quarter\n", -b[1]/mult[5,2]
printf "\nThe Total Multiplier = %.4f\n", mult[5,2]
```

The first improvement is that this one handles additional exogenous regressors easily. These are place into a list called `exo` and the number of exogenous regressors is captured in line as $k$. The second improvement is the adaptation to a ARDL model that permits $p$ lagged dependent variables as regressors. Our example contains no lagged endogenous variables and $p$=0. The scalar $q$ captures the number of lags for the DL portion of the model, in this case 4. The coefficients are parsed into two sets, $a$ and $b$. The vector $b$ contains the coefficients of the AR part of the model, and $a$ contains the lag weights on the DL portion.

The `filter` command computes an ARMA-like filter as in:

$$y_t = a_0 x_t + a_1 x_{t-1} + \cdots + a_q x_{t-q} + b_1 y_{t-1} + \cdots + b_p y_{t-p}$$

The `filter` command as used here takes three arguments. The first is a vector $(1, 0, 0, 0, 0, 0)$ which represents the values of $x$ for the multiplier. This means that $x_t = 1$ and the other lags are zero. The next argument contains the DL lag weights starting at lag 0. Since there are no AR terms, $b$ is empty and set to `null`. This returns a vector containing the delay multipliers. The `cum` (cumulative) function takes the cumulative sum which produces the interim multipliers. This produces the same result as our original script.

### 9.9.2 Phillips Curve

**Example 9.14**

The Phillips curve and corresponding model to be estimated are in equations 9.8 and 9.9, respectively:

$$\begin{aligned} inf_t &= inf_t^e - \gamma(u_t - u_{t-1}) && (9.8) \\ inf_t &= \alpha + \beta_0 du_t + e_t && (9.9) \end{aligned}$$

where $inf_t^e = \alpha$ is expected inflation and $du_t = (u_t - u_{t-1})$ is the change in the unemployment rate. The data are in *phillips5_aus.gdt*.

The data are quarterly and begin in 1987:1. A time-series plot of the inflation rate is shown below in Figure 9.14. The graphs show some evidence of serial correlation in the inflation rate.

The model is estimated by least squares and the residuals are plotted against time. These appear in Figure 9.15. 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:

```
open "@workdir\data\phillips5_aus.gdt"

# Graph of series against lags
string title = "Australian Inflation Rate: 1987:1 - 2016:1"
string xname = "Year"
string yname = "Inflation Rate"
list plotvars = inf
g4 <- plot plotvars      # Plotting the series, save to session as g4
    options --time-series --with-lines # gretl options
    printf "set title \"%s\"", title   # title and axis labels
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

ols inf const du                   # Phillips curve estimation
series residual = $uhat
corrgm residual 16 --plot=display  # Graph of correlogram
```

Unfortuantely, **gretl** will not accept the accessor, `$uhat`, as an input into either `gnuplot`, `plot`, or `corrgm`. That means the residuals must be saved as a series, `residual`, first. All three functions work as expected when used on the series.

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.

If you are using the GUI rather than a **hansl** 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 9.16 below. Click on the **lags** button and the dialog shown on the right will pop-up. Add the desired number of lags to the variable of choice. Click **OK** and the lags will be added to the regressor list as shown on the left.

### 9.9.3 Least Squares and HAC Standard Errors

As is the case with heteroskedastic errors, HAC covariance estimation provides a statistically valid way to use least squares when your data are also autocorrelated. Standard errors will be robust with respect 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.

### Bandwidth and Kernel

HAC is not quite as automatic as the heteroskedasticity consistent (HCCME) estimator in Chapter 8 because it contains an extra parameter. To be robust with respect to autocorrelation the number of time periods for which there is significant autocorrelation among residuals must be specified. Autocorrelated errors over the chosen time window are averaged in the computation of the HAC standard errors

The language of time-series analysis can 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 the **bandwidth**. The kernel provides a weighting scheme over which the average is taken; the bandwidth determines the number of periods to use to compute the weighted average. In **gretl** you may choose the 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 $nw1 = 0.75 \times N^{1/3}$. 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 both bias (good) and 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 generic recommendation is to choose a bandwidth that is large enough to contain the largest autocorrelations. Goldilock's choice will ultimately depend on the frequency of observation and the length of time it takes for the system under study 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 set command is used to manipulate options that make this possible without much effort. First, bandwidth choice is switched to the nw2 option using the set hac_lag switch. In this example, nw2 is estimated to be equal to 4, which is what is used in *POE5*. The set force_hc switch is set to off (the default, which for time series produces HAC). Also, the Bartlett kernel is chosen by setting the hac_kernel switch to bartlett. After this model is estimated and saved to a model table, the force_hc switch is turned on to force computation of HC errors. The entire set of code is:

```
ols inf const du           # Phillips curve estimation
series residual = $uhat
corrgm residual 16 --plot=display

ols inf const du           # OLS with inconsistent std errors
modeltab add
set hac_lag nw2            # automatic bandwidth setting
set force_hc off            # off: --robust produces HAC
set hac_kernel bartlett     # choose the kernel

ols inf const du --robust   # OLS with HAC
modeltab add
set force_hc on             # on: --robust produces HC1

ols inf const du --robust   # OLS with HC1
modeltab add
modeltab show
```

The model table appears below.

OLS estimates
Dependent variable: inf

| | (Usual OLS) | (HAC) | (HC1) |
|---|---|---|---|
| const | 0.7317** | 0.7317** | 0.7317** |
| | (0.05606) | (0.09075) | (0.05688) |
| du | −0.3987* | −0.3987 | −0.3987 |
| | (0.2061) | (0.2854) | (0.2632) |
| $n$ | 117 | 117 | 117 |
| $R^2$ | 0.0315 | 0.0315 | 0.0315 |
| $\ell$ | −106.1 | −106.1 | −106.1 |

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

The HAC standard errors are the larger than the HC1 and the usual OLS standard errors. Notice that the slope is not significant with HAC standard errors.

### Example 9.15

In this example, the Phillips curve with AR(1) errors is estimated using three techniques: 1) OLS with HAC standard errors, 2) Nonlinear Least Squares (which was introduced in section 6.8), and 3) Feasible GLS.

First, OLS:

```
set force_hc off
set hac_kernel bartlett
set hac_lag 4
m1 <- ols inf const du --robust
```

### Nonlinear Least Squares

Perhaps the best way to estimate a linear model that is autocorrelated is using **nonlinear least squares**. The nonlinear least squares estimator (NLS) only requires that the time series be stable (not necessarily stationary). 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, $\rho$, 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.

As mentioned in section 6.8 nonlinear least squares requires more computational power than linear estimation, though this is not much of a deterrent these days. Nonlinear least squares (and other nonlinear estimators) use numerical methods rather than analytical ones to find the minimum of the sum-of-squared-errors objective function. The routines are iterative. The user supplies a good guess for the values of the parameter and the algorithm evaluates the sum-of-squares function at this guess. The slope of the sum-of-squares function at the guess points in a direction that leads closer to a smaller sum of squared errors and computes a *step* in the parameter space that moves the next iteration 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 can be found.

In the context of the area response equation the AR(1) model is

$$inf_t = \beta_1(1-\rho) + \beta_2(\Delta u_t - \rho\,\Delta u_{t-1}) + \rho\, inf_{t-1} + v_t \tag{9.10}$$

The errors, $v_t$, are random and the goal is to find $\beta_1$, $\beta_2$, and $\rho$ that minimize $\sum v_t^2$.

### A More General Model

Equation 9.10 can be expanded and rewritten in the following way:

$$inf_t = \delta + \delta_0\Delta u_t + \delta_1\Delta u_{t-1} + \theta_1\, inf_{t-1} + v_t \tag{9.11}$$

Both equations contain the same variables, but Equation (9.10) contains only 3 parameters while (9.11) has 4. This means that (9.10) is *nested* within (9.11) and a formal hypothesis test can be performed to determine whether the implied restriction holds. The restriction is $\delta_1 = -\theta_1\delta_0$.[3]

[3] $\delta = \beta_1(1-\rho), \delta_0 = \beta_2, \delta_1 = -\rho\beta_2, \theta_1 = \rho$

To test this hypothesis using **gretl** write a function for the nonlinear hypothesis and use the `restrict` statement to estimate and test the restriction. The script is:

```
1  m3 <- ols inf const du(0 to -1) inf(-1) --robust
2
3  function matrix restr (const matrix b)
4      matrix v = b[3] + b[4]*b[2]
5      return v
6  end function
7
8  restrict
9      rfunc = restr
10     end restrict
```

The linear regression in equation 9.11 is estimated in line 1 using HAC standard errors.

$$\widehat{\text{inf}} = \underset{(0.07381)}{0.3483} - \underset{(0.2445)}{0.3728}\,\text{du} + \underset{(0.2470)}{0.01714}\,\text{du\_1} + \underset{(0.1033)}{0.4992}\,\text{inf\_1}$$

$$T = 116 \quad \bar{R}^2 = 0.2638 \quad F(3,112) = 13.502 \quad \hat{\sigma} = 0.51712$$

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

The estimates of $\rho$ and $\beta_2$ are very close to the ones from NLS found below. The lagged unemployment rate has a $t$-ratio of 0.069. It is not significant and it may be worth considering removing it from the model using the `omit du_1` statement.

Lines 3-6 host a function that returns a matrix called `restr`. It has only one argument, `const matrix b`. `const matrix` signals that the argument is a **constraint matrix**. See section 6.1.3. Finally, the last three lines test the hypothesis. The restricted model in not estimated in this instance.

For the example, the test statistic and $p$-value are:

```
Test statistic: Robust chi^2(1) = 0.424682, with p-value = 0.51461
```

The NLS model (null) cannot be rejected at 5% significance.

To estimate the model by NLS requires starting values for the parameters. Ordinary least squares is a good place to start since OLS is consistent for the slope(s) and intercept. The autocorrelation parameter, $\rho$, is started at zero. The script is this follows:

```
open "@workdir\data\phillips5_aus.gdt"
ols inf const du --quiet

```

```
scalar beta1 = $coeff(const)
scalar beta2 = $coeff(du)
scalar rho = 0

nls inf = beta1*(1-rho) + rho*inf(-1) + beta2*(du-rho*d_u(-1))
    params rho beta1 beta2
end nls
```

The `nls` block is initiated with `nls` followed by the equation representing the systematic portion of your model. The block is closed by the statement `end nls`. When possible, it is a good idea to supply analytical derivatives for nonlinear optimization. I did not, opting to let **gretl** compute numerical derivatives. When using numerical derivatives, the `params` statement is required 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 `genr` or `series` command.

Magically, this yields the same result as *POE5*!

```
              m2: NLS, using observations 1987:2–2016:1 (T = 116)
        inf = beta1*(1-rho) + rho*inf(-1) + beta2*(du-rho*du(-1))

                      Estimate    Std. Error   t-ratio   p-value

          rho         0.500064    0.0809356     6.179    0.0000
          beta1       0.702846    0.0963154     7.297    0.0000
          beta2      −0.383025    0.210459     −1.820    0.0714

   Mean dependent var     0.729741   S.D. dependent var   0.602674
   Sum squared resid      30.19489   S.E. of regression   0.516925
   R^2                    0.277114   Adjusted R^2         0.264319
   Log-likelihood        −86.53366   Akaike criterion     179.0673
   Schwarz criterion      187.3281   Hannan–Quinn         182.4207

              GNR: R^2 = 8.88178e-016, max |t| = 2.88248e-007
                 Convergence seems to be reasonably complete
```

To compare the constrained parameters estimated by NLS and the unconstrained one in the generalization estimated by OLS, we compute $\delta = \beta_1(1-\rho)$ and $\delta_1 = \rho\beta_2$.

```
scalar delta = $coeff(beta1)*(1-$coeff(rho))
scalar delta1 = -$coeff(rho)*$coeff(beta2)
```

```
13 printf "\nThe estimated delta is %.3f and the estimated delta1\
14 is %.3f.\n",delta,delta1
```

In lines 11 and 12 $\delta$ and $\delta_1$ are approximated from the NLS estimated AR(1) regression. the result is

```
The estimated delta is 0.351 and the estimated delta1 is 0.192.
```

## FGLS

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:

```
m4 <- ar1 inf const du
m5 <- ar1 inf const du --pwe
m6 <- ar1 inf const du --hilu
```

The `ar` command is more general and computes parameter estimates using the generalized Cochrane-Orcutt iterative procedure. This routine allows additional lags of the errors to be modeled by specifically listing which lags to include in the error function. The syntax is:

```
ar

Arguments:  lags ; depvar indepvars
Option:     --vcv (print covariance matrix)
Example:    ar 1 3 4 ; y 0 x1 x2 x3
```

For the example we include:

```
m7 <- ar 1; inf const du
```

Notice that the lag number(s) follows the `ar` command followed by a semi-colon and the regression model to be estimated. In the script found at the end of this chapter these are combined into a model table, the results of which are shown below.

Dependent variable: inf

|  | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| Estimator | `ols --robust` | `ar1` | `ar 1` | `ar1 --pwe` | `ar1 --hilu` |
| const | 0.7317** | 0.7028** | 0.7028** | 0.7343** | 0.7028** |
|  | (0.09075) | (0.09568) | (0.09568) | (0.09581) | (0.09568) |
| du | −0.3987 | −0.3830* | −0.3830* | −0.3950* | −0.3830* |
|  | (0.2854) | (0.2087) | (0.2087) | (0.2116) | (0.2087) |
| $n$ | 117 | 116 | 116 | 117 | 116 |
| $\bar{R}^2$ | 0.0231 | 0.2709 | 0.2709 | 0.2714 | 0.2709 |
| $\ell$ | −106.1 |  |  |  |  |

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

The first column is estimated using ols with the `--robust` option, which produces HAC standard errors. The next two columns are virtually identical; they differ only by the command that was used to produce them. Column (2) uses the specialized `ar1` command and (3) uses the more general `ar` command with a single lag of 1 listed as an argument. The final two columns are estimated using `ar1` with the Prais-Winsten option and the Hildreth-Lu option, respectively. Note that only OLS and `ar1 --pwe` uses the entire sample of 117 observations.

### Maximum Likelihood

There is one more alternative to consider: the `arima` command., 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 uses the “native” **gretl** ARMA functionality, with estimation by exact maximum likelihood using the Kalman filter.[4] You can estimate the parameters via conditional maximum likelihood as well.

The model has one auto regressive term, no moving average term, and no difference taken. The `arima` syntax is similar to the `ar` command, 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 simple syntax is:

```
m8 <- arima 1 0 0; inf const du
```

The results are:

m8: ARMAX, using observations 1987:1–2016:1 ($T = 117$)
Dependent variable: inf
Standard errors based on Hessian

| | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | 0.734462 | 0.0977049 | 7.517 | 0.0000 |
| $\phi_1$ | 0.514054 | 0.0815822 | 6.301 | 0.0000 |
| du | −0.394770 | 0.209924 | −1.881 | 0.0600 |

| | | | |
|---|---|---|---|
| Mean dependent var | 0.740598 | S.D. dependent var | 0.611454 |
| Mean of innovations | −0.005387 | S.D. of innovations | 0.517551 |
| Log-likelihood | −89.10748 | Akaike criterion | 186.2150 |
| Schwarz criterion | 197.2637 | Hannan–Quinn | 190.7006 |

| | | | Real | Imaginary | Modulus | Frequency |
|---|---|---|---|---|---|---|
| AR | | | | | | |
| | Root | 1 | 1.9453 | 0.0000 | 1.9453 | 0.0000 |

You can see that these are very close to those obtained using NLS or FGLS. The parameter $\phi_1$ corresponds to $\rho$ in the NLS and FGLS estimators. It is estimated to be .51. 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.

### 9.9.4 A Consumption Function

[4] Cottrell and Lucchetti (2018)

### Example 9.16 in *POE5*

Suppose current consumption is a function of permanent income:

$$c_t = \omega + \beta y_t^P$$

The ARDL model adds lagged values of the dependent variable to the AR model,

$$y_t = \delta + \theta_1 y_{t-1} + \cdots + \theta_p y_{t-p} + \delta_0 x_t + \delta_1 x_{t-1} + \cdots + \delta_q x_{t-q} + v_t \tag{9.12}$$

Permanent income, $y^P$, is not observed. It is assumed that it consists of a trend and a geometrically weighted average of observed current and past incomes. This is transformed into an infinite distributed lag model using the properties of the lag operator, $L$. That is, $L^i x_t = x_{t-i}$.

$$y_t^p = \gamma_0 + \gamma_1 t + \gamma_2 \left(1 + \lambda_1 L + \lambda_2 L^2 + \lambda_3 L^3 + \cdots\right) y_t \tag{9.13}$$

Taking the difference of $c_t$ produces

$$dc_t = c_t - c_{t-1} = \beta\gamma_1 + \beta\gamma_2 \left(1 + \lambda L + \lambda_2 L^2 + \lambda_3 L^3 + \cdots\right) dy_t$$

Setting $\alpha = \beta\gamma_1$, $\beta_0 = \beta\gamma_2$ and adding an error term produces the regression model:

$$dc_t = c_t - c_{t-1} = \alpha + \beta_0 \left(1 + \lambda L + \lambda_2 L^2 + \lambda_3 L^3 + \cdots\right) dy_t + e_t$$

The ARDL(1,0) representation of this infinite distributed lag model (i.e., a special case of equation (9.9.1)) becomes:

$$dc_t = \delta + \lambda dc_{t-1} + \beta_0 dy_t + v_t \tag{9.14}$$

This is estimated in **gretl** :

```
open "@workdir\data\cons_inc.gdt"
diff cons y
list xvars = const d_cons(-1) d_y
ols d_cons xvars
```

which produces:

$$\widehat{\text{d\_cons}} = \underset{(74.20)}{478.6} + \underset{(0.0599)}{0.337}\ \text{d\_cons\_1} + \underset{(0.0215)}{0.0991}\,\text{d\_y}$$

$$T = 227 \quad \bar{R}^2 = 0.214 \quad F(2, 224) = 31.847 \quad \hat{\sigma} = 734.69$$

(standard errors in parentheses)

```
ols d_cons xvars

```

```
matrix b = $coeff
printf "\nTotal Multiplier = %.3f\n", b[3]/(1-b[2])
loop i=1..4
    modtest $i --autocorr --quiet
endloop
```

The total multiplier for an infinite distributed lag model is $\beta_0/(1-\lambda)$. This is computed using the least squares coefficients accessed through `$coeff` and placed into a matrix called `b`.

Delay multipliers are simple to compute as well. The impact is simply the coefficient of $dy$, 0.0991. The one period delay is $\lambda\beta_0$, the second delay is $\lambda^2\beta_0$ and so on.

The $LM$ test from section 9.8 was conducted using a loop. In each case the null was of no autocorrelation was not rejected by the $LM$ test.

Another test was conducted as suggested by McClain and Wooldridge (1995). There are four steps to this test.

1. Estimate the regression and save the residuals, $\hat{u}_t$.
2. Using the estimated coefficient on the lagged dependent variable, $\hat{\lambda}$, and starting with $\hat{e}_t = 0$, compute $\hat{e}_t = \hat{\lambda}\hat{e}_{t-1} + \hat{u}_t$
3. Run a regression of $\hat{u}_t$ on the regressors in the model augmented by $\hat{e}_t$
4. If the estimator is consistent and assuming $u_t$ is homoskedastic, $(T-1)R^2 \sim \chi^2(1)$ in large samples.

For the Phillips curve this is accomplished using:

```
ols d_cons xvars
series uhat = $uhat
series ehat = 0
series ehat = $coeff(d_cons_1)*ehat(-1) + uhat
ols uhat xvars ehat(-1) --quiet
scalar stat = $trsq
pvalue X 1 stat
```

and produces:

```
Generated scalar stat = 0.056747

Chi-square(1): area to the right of 0.056747 = 0.811713
(to the left: 0.188287)
```

The consistency of the model's estimator is not rejected at 5%.

### Example 9.17 and 9.18 in *POE5*

In these examples multipliers are derived and estimated for an ARDL(2,1) model. As done in section 9.9.1 above, the ARDL is converted into an infinite distributed lag model before multipliers are derived. This is done in *POE5* example 9.17. At the end of this example, we show how to avoid this step in **gretl** using the `filter` function.

The ARDL(2,1) is:

$$du_t = \delta + \theta_1 du_{t-1} + \theta_2 du_{t-2} + \delta_0 g_t + \delta_1 g_{t-1} + v_t \tag{9.15}$$

They show that the coefficients of the IDL are:

$$\begin{aligned}\beta_0 &= \delta_0\\ \beta_1 &= \theta_1\beta_0\\ \beta_j &= \theta_1\beta_{j-1} + \theta_2\beta_{j-2} \qquad j \geq 2\end{aligned}$$

For Okun model the **hansl** script estimates the model and computes the multipliers recursively. To make the script easier to decipher, scalars are created for the parameters ($\delta_0$, $\delta_1$, $\theta_1$, $\theta_2$) in lines 5-8. We compute only eleven of the multipliers and create a matrix to hold them as they are computed in the loop. The loop has three equations to compute that depend on lag length. This is done using `if`, `elif`, and `else` conditional commands.

```
1  open "@workdir\data\okun5_aus.gdt"
2  diff u
3  ols d_u(0 to -2) g(0 to -1) const
4
5  scalar d0 = $coeff(g)
6  scalar d1 = $coeff(g_1)
7  scalar theta1=$coeff(d_u_1)
8  scalar theta2=$coeff(d_u_2)
9
10 scalar h = 11
11 matrix mult = zeros(h,2)
12
13 loop i=1..h
14     mult[i,1] = i-1
15     if i=1
16         mult[i,2]=d0
17     elif i=2
18         mult[i,2]=d1 + theta1*d0
19     else
20         mult[i,2]=mult[i-1,2]*theta1 + theta2*mult[i-2,2]
21     endif
22 endloop
```

The multipliers are stored in memory to a matrix called mult. Once mult is populated the various statistics that are computed using them are easy to form. First, we add column names to the matrix using cnameset and plot the delay multipliers using **gnuplot**. The graph appears in Figure 9.17.

```
cnameset(mult, " Lag    Delay_Mult ")
gnuplot 2 1 --matrix=mult --output=display --with-lines --suppress-fitted
```

Next, the delay multipliers depicted in the graph are printed.

```
The impact and delay multipliers are
          Lag  Delay_Mult
           0     -0.0904
           1      -0.154
           2     -0.0593
           3     -0.0475
           4     -0.0248
           5     -0.0164
           6    -0.00946
           7    -0.00589
           8    -0.00352
           9    -0.00215
          10     -0.0013
```

Finally, the total multiplier based on the first 10 delay multipliers, the normal growth rate, and the asymptotically derived total multiplier are computed.

```
printf "\nTotal multiplier using the sum of estimates is %.3g\n",\
       sum(mult[,2])
scalar alpha = $coeff(const)/(1-theta1-theta2)
scalar TotalMult = (d0+d1)/(1-theta1-theta2)
printf "\nNormal Growth = %.3f%%\n", -alpha/TotalMult
printf "\nAsymptotic Total Multiplier = %.3f%%\n", TotalMult
```

This produces:

```
Total multiplier using the sum of estimates is -0.414
Normal Growth = 0.776%
Asymptotic Total Multiplier = -0.416%
```

The first 10 delay multipliers captures most of the changes accounted for based on the asymptotic computation of the total multiplier. The estimated sustained growth rate needed to maintain full employment is 0.776% per quarter.

### Using filter

As seen in Example 9.13, the filter function can be used to generate multipliers. In this case, there is no need to generate the infinite DL model, which saves a lot of work. This example shows how easily it is to conduct a multiplier analysis using any ARDL model.

```
list exo = const
p = 2
q = 1
list exo = const
horizon = 10

ols d_u exo d_u(-1 to -p) g(0 to -q)
k = nelem(exo)
matrix b = $coeff[k+1:k+p]
matrix a = $coeff[k+p+1:k+p+q+1]

mult = filter(1|zeros(horizon, 1), a, b)
mult = mult ~ cum(mult)

cnameset(mult,"Delay Interim")
printf "Multipliers for Okun's Law, p=%g, q=%g\n%10.4f\n", p, q, mult
```

Simply choose the desired p, q, exogenous variables, and horizon. The rest is automated and produces:

```
Multipliers for Okun's Law, p=2, q=1
     Delay   Interim
   -0.0904   -0.0904
   -0.1535   -0.2439
   -0.0593   -0.3032
   -0.0475   -0.3506
   -0.0248   -0.3754
   -0.0164   -0.3918
   -0.0095   -0.4013
   -0.0059   -0.4072
   -0.0035   -0.4107
   -0.0021   -0.4128
   -0.0013   -0.4141
```

Very slick. Thanks Jack!

### Example 9.19

In this example Okun's ARDL(2,1) from the preceding example is tested for consistency using the McClain and Wooldridge test. Assuming that the errors of the IDL follow an AR(2) process

$$e_t = \psi_1 e_{t-1} + \psi_2 e_{t-2} + v_t.$$

In terms of the coefficients of the equation (9.15), $H_0$: $\psi_1 = \theta_1$ and $\psi_2 = \theta_2$ against the two-sided alternative. In this instance, $\hat{u}_t$ are from the estimated ARDL(2,1) model, $\hat{e}_t = \hat{\theta}_1 \hat{e}_{t-1} + \hat{\theta}_2 \hat{e}_{t-2} + \hat{u}_t$. Regress $\hat{u}_t$ onto the regressors of the ARDL(2,1) augmented with $\hat{e}_{t-1}$ and $\hat{e}_{t-1}$. $TR^2 \sim \chi^2(2)$ if $H_0$ is true. The script is:

```
list xvars = d_u(-1 to -2) g(0 to -1) const
ols d_u xvars
series uhat = $uhat
matrix b = $coeff

series ehat = 0
#  McLain & Wooldridge test
series ehat = $coeff(d_u_1)*ehat(-1) + $coeff(d_u_2)*ehat(-2)+ uhat

ols uhat xvars ehat(-1 to -2) --quiet
scalar stat = $trsq
pvalue X 2 stat
```

which yields:

```
Chi-square(2): area to the right of 3.13367 = 0.208704
(to the left: 0.791296)
```

The null hypothesis cannot be rejected at 5%. The parameters of the ARDL are estimated consistently.

### Example 9.20 in *POE5*

Finally, the Durbin-Watson test is used to test for evidence of AR(1) errors in the Phillips curve example.

```
open "@workdir\data\phillips5_aus.gdt"
list x = du const
ols inf x
scalar dw_p = $dwpval
print dw_p
```

Gretl computes the actual small sample $p$-value for this test using Imhoff integration. It stores the result in temporary memory and can be accessed using the accessor, `$dwpval`. The regression output is:

m1: OLS, using observations 1987:1–2016:1 ($T = 117$)
Dependent variable: inf

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 0.731739 | 0.0560595 | 13.05 | 0.0000 |
| du | −0.398670 | 0.206055 | −1.935 | 0.0555 |

| | | | |
|---|---|---|---|
| Mean dependent var | 0.740598 | S.D. dependent var | 0.611454 |
| Sum squared resid | 42.00244 | S.E. of regression | 0.604350 |
| $R^2$ | 0.031525 | Adjusted $R^2$ | 0.023103 |
| $F(1, 115)$ | 3.743348 | P-value($F$) | 0.055474 |
| Log-likelihood | −106.0857 | Akaike criterion | 216.1714 |
| Schwarz criterion | 221.6958 | Hannan–Quinn | 218.4142 |
| $\hat{\rho}$ | 0.499727 | Durbin–Watson | 0.964608 |

The DW statistic is 0.9646. It's $p$-value is:

```
dw_p =  4.8337456e-010
```

The $p$-value is tiny and DW is definitely significant at 5% according to this test.

## 9.10 Script

```
set verbose off
# Example 9.1
# Plotting time series
open "@workdir\data\usmacro.gdt"
# change variable attributes
setinfo g -d "% 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"
setinfo inf -d "U.S. Inflation Rate\
(%change CPI, seasonally adjusted) " -n "Inflation Rate"

# plot series and save output to files
gnuplot g --with-lines --time-series --output=display
gnuplot u --with-lines --time-series --output=display
```

```

g1_simple <- plot u
    options time-series with-lines
end plot --output=display

string title = "U.S. Quarterly unemployment rate"
string xname = "Year"
string yname = "Unemployment Rate"
g1 <- plot u
    options time-series with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

string title = "U.S. GDP growth rate"
string xname = "Year"
string yname = "Quarterly GDP growth rate"
g2 <- plot g
    options time-series with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

# graphing multiple time-series
g3 <- scatters g u --with-lines --output=display

# Graph of series against lags
string title = "Unemployment against lagged unemployment"
string xname = "Lagged Unemployment"
string yname = "Unemployment Rate"
list plotvars = u(0 to -1)
g4 <- plot plotvars
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

# Example 9.2
# Sample autocorrelations for u
corrgm u 24 --plot=display

matrix ac = corrgm(g, 12)            # ACFs and PACFs to a matrix
matrix lb = ac[,1]-1.96/sqrt($nobs) # col 1 = acf, col 2 = pacf
matrix ub = ac[,1]+1.96/sqrt($nobs)
matrix all = lb~ac[,1]~ub
cnameset(all, "Lower  AC Upper ")
printf "\nAutocorrelations and 95%% confidence intervals\n\
%9.4f\n", all
```

```

# Example 9.3
# Sample autocorrelations for g
corrgm g 45 --plot=display

# Example 9.6
# Forecasting Unemployment with AR(2)
m1 <- ols u const u(-1 to -2)
dataset addobs 3
fcast 2016:2 2016:4 --dynamic

# Example 9.7
# Forecasting Unemployment with ARDL(2,1)
# correlogram and confidence interval
ols u const u(-1 to -2) g(-1)
# Add g_2016:2=0.869 and g_2016:3=1.069 to dataset manually
series g[2016:2]=.869
fcast 2016:2 2016:4 --dynamic

# Example 9.8 Choosing lag lengths, SC criterion
# 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, $rsq, rbar2, aic, bic}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "k = %d, n = %d, R2 = %.4f, Adjusted R2 = %.4f, AIC = %.4f,\
and SC = %.4f\n", k, N, $rsq, rbar2, aic, bic
    return A
end function

# Same as modelsel except the print statements are suppressed
function matrix modelsel_np (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, $rsq, rbar2, aic, bic}
    return A
end function

# using modelsel_np
open "@workdir\data\usmacro.gdt"
```

```
smpl 1950:1 2016:1
matrix A = {}
loop p = 1..8  --quiet
    loop q = 0..8 --quiet
         if q==0
            list xvars = u(-1 to -p) const
         else
            list xvars = u(-1 to -p) g(-1 to -q) const
         endif
        matrix a = p~q~modelsel_np(u, xvars)
        matrix A = A | a
    endloop
endloop
cnameset(A,"p q k n R2 Adj_R2 AIC SC ")
matrix B = msortby(A,8)
printf "\nModel Selection in ARDL\n%8.4g\n",B[1:6,]

# Example 9.9
# Does growth Granger cause unemployment?
smpl full
ols u(0 to -2) g(-1) const
omit g_1

list ulags = u(-1 to -2)
list glags = g(-1 to -4)

smpl 1949:1 2016:1
ols u ulags glags const --quiet
omit glags --test-only

# Another utility to create lags
lags 4 ; g
list g_lags = g_1 g_2 g_3 g_4

# Example 9.10
# Residual Correlogram
smpl full
ols u u(-1 to -2) g(-1) const
series residual = $uhat
g1 <- corrgm residual 24 --plot=display

# Example 9.11
# Residual Correlogram
smpl full
ols u u(-1) g(-1) const
series residual = $uhat
g2 <- corrgm residual 24 --plot=display


# Example 9.12
# LM Tests for k-1,2,3,4
```

```
open "@workdir\data\usmacro.gdt"
ols u u(-1) g(-1) const
    loop i=1..4
        modtest $i --autocorr --quiet
    endloop

ols u u(-1 to -2) g(-1) const
    loop i=1..4
        modtest $i --autocorr --quiet
    endloop

# Example 9.13
# Okun’s Law
open "@workdir\data\okun5_aus.gdt"
diff u
setinfo g -n "GDP growth rate"
setinfo d_u -d "Change in Australian Civilian Unemployment \
  Rate  (Seasonally adjusted)" -n \
  "Change in Unemployment Rate"
g4 <- scatters g d_u --with-lines --output=display

modeltab free
m5 <- ols d_u const g(0 to -5)
modeltab add
m4 <- ols d_u const g(0 to -4)
modeltab add
modeltab show

open "@workdir\data\okun5_aus.gdt"
diff u
    smpl 1980:1 2016:1
    matrix A = {}
    scalar q=0
    loop p = 1..6  --quiet
        list vars = g(0 to -p) const
        matrix a = p˜q˜modelsel_np(d_u, vars)
        matrix A = A | a
    endloop
cnameset(A,"p q K N R2 Adj_R2 AIC SC ")
print A
matrix B = msortby(A,8)
printf "\nModel Selection in ARDL\n%8.4g\n",B[1:2,]

# multiplier analysis
open "@workdir\data\okun5_aus.gdt"
diff u
ols d_u g(0 to -4) const
matrix b = $coeff
print b
matrix mult = zeros(5,2)
loop i=1..5
```

```
    matrix mult[i,1]=b[i+1]
    matrix mult[i,2]=b[i+1]
    if i>1
    matrix mult[i,2]=mult[i-1,2]+b[i+1]
    endif
endloop
cnameset(mult,"Delay Interim")
rnameset(mult,"0 1 2 3 4")
printf "Multipliers for Okun’s Law, q=4\n%10.4f\n", mult

printf "\nNormal Growth rate = %.4f%% per quarter\n", -b[1]/mult[5,2]
printf "\nThe Total Multiplier = %.4f\n", mult[5,2]

# Simplification using filter and cum
open "@workdir\data\okun5_aus.gdt"
diff u

list exo = const
p = 0
q = 4
list exo = const
horizon = 4

ols d_u exo g(0 to -q)
k = nelem(exo)
matrix b = $coeff[k+1:k+p]
matrix a = $coeff[k+p+1:k+p+q+1]

mult = filter(1|zeros(horizon, 1), a, null)
mult = mult ~ cum(mult)

cnameset(mult,"Delay Interim")
rnameset(mult,"0 1 2 3 4")
printf "Multipliers for Okun’s Law, q=4\n%10.4f\n", mult
printf "\nNormal Growth rate = %.4f%% per quarter\n", -a[1]/mult[5,2]
printf "\nThe Total Multiplier = %.4f\n", mult[5,2]

# Example 9.14
# AR(1) errors
modeltab free
open "@workdir\data\phillips5_aus.gdt"

# Graph of series against lags
string title = "Australian Inflation Rate: 1987:1 - 2016:1"
string xname = "Year"
string yname = "Inflation Rate"
list plotvars = inf
g4 <- plot plotvars
    options --time-series --with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

ols inf const du           # Phillips curve estimation
series residual = $uhat
corrgm residual 16 --plot=display

ols inf const du           # OLS with inconsistent std errors
modeltab add
set hac_lag nw2            # automatic bandwidth setting
set force_hc off           # off: --robust produces HAC
set hac_kernel bartlett    # choose the kernel

ols inf const du --robust  # OLS with HAC
modeltab add
set force_hc on            # on: --robust produces HC1

ols inf const du --robust  # OLS with HC1
modeltab add
modeltab show

# Example 9.15
# Phillips Curve with AR(1) errors
open "@workdir\data\phillips5_aus.gdt"
setinfo inf -d "Australian Inflation Rate" -n "Inflation Rate"
setinfo du -d "Change in Australian Civilian \
  Unemployment Rate  (Seasonally adjusted)" -n \
  "D.Unemployment Rate"
scatters inf du --with-lines --output=display

# OLS with HAC standard errors
set force_hc off
set hac_kernel bartlett
set hac_lag 4
m1 <- ols inf const du --robust
modeltab free
modeltab add
# NLS
scalar beta1 = $coeff(const)
scalar beta2 = $coeff(du)
scalar rho = 0

m2 <- nls inf = beta1*(1-rho) + rho*inf(-1) + beta2*(du-rho*du(-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

```

```
# More General model
m3 <- ols inf const du(0 to -1) inf(-1) --robust
modeltab add
function matrix restr (const matrix b)
    matrix v = b[3] + b[4]*b[2]
    return v
end function
restrict
    rfunc = restr
    end restrict

# These two are equivalent. The second preferred since it more general.
m4 <- ar1 inf const du
m7 <- ar 1; inf const du
modeltab add

m5 <- ar1 inf const du --pwe
modeltab add
m6 <- ar1 inf const du --hilu
modeltab add
modeltab show

m8 <- arima 1 0 0; inf const du

# Example 9.16 Consumption function

open "@workdir\data\cons_inc.gdt"
diff cons y
list xvars = const d_cons(-1) d_y
m1 <-  ols d_cons xvars

matrix b = $coeff
printf "\nTotal Multiplier = %.3f\n", b[3]/(1-b[2])
loop i=1..4
    modtest $i --autocorr --quiet
endloop

#  McLain & Wooldridge test
ols d_cons xvars --quiet
series uhat = $uhat
series ehat = 0
series ehat = $coeff(d_cons_1)*ehat(-1) + uhat
ols uhat xvars ehat(-1) --quiet
scalar stat = $trsq
pvalue X 1 stat

# Example 9.18
# Multipliers for Okun's Law
# The hard way
open "@workdir\data\okun5_aus.gdt"
diff u
```

```
ols d_u(0 to -2) g(0 to -1) const
scalar d0 = $coeff(g)
scalar d1 = $coeff(g_1)
scalar theta1=$coeff(d_u_1)
scalar theta2=$coeff(d_u_2)

scalar h = 11
matrix mult = zeros(h,2)

loop i=1..h
    mult[i,1] = i-1
    if i==1
        mult[i,2]=d0
    elif i==2
        mult[i,2]=d1 + theta1*d0
    else
        mult[i,2]=mult[i-1,2]*theta1 + theta2*mult[i-2,2]
    endif
endloop

cnameset(mult, " Lag    Delay_Mult ")
gnuplot 2 1 --matrix=mult --output=display --with-lines --suppress-fitted

printf "\nThe impact and delay multipliers are \n %12.3g\n", mult

printf "\nTotal multiplier using the sum of estimates is %.3g\n", sum(mult[,2])
scalar alpha = $coeff(const)/(1-theta1-theta2)
scalar TotalMult = (d0+d1)/(1-theta1-theta2)
printf "\nNormal Growth = %.3f%%\n", -alpha/TotalMult
printf "\nAsymptotic Total Multiplier = %.3f%%\n", TotalMult

# Using filter to make things easy
list exo = const
p = 2
q = 1
list exo = const
horizon = 10

ols d_u exo d_u(-1 to -p) g(0 to -q)
k = nelem(exo)
matrix b = $coeff[k+1:k+p]
matrix a = $coeff[k+p+1:k+p+q+1]

mult = filter(1|zeros(horizon, 1), a, b)
mult = mult ~ cum(mult)

cnameset(mult,"Delay Interim")
printf "Multipliers for Okun's Law, p=%g, q=%g\n%10.4f\n", p, q, mult

# Example 9.19
# Testing consistency of OLS
```

```
list xvars = d_u(-1 to -2) g(0 to -1) const
ols d_u xvars
series uhat = $uhat
matrix b = $coeff

series ehat = 0
#  McLain & Wooldridge test
series ehat = $coeff(d_u_1)*ehat(-1) + $coeff(d_u_2)*ehat(-2)+ uhat

ols uhat xvars ehat(-1 to -2) --quiet
scalar stat = $trsq
pvalue X 2 stat

# Example 9.20 DW test
open "@workdir\data\phillips5_aus.gdt"
list x = du const
m1 <- ols inf x
scalar dw_p = $dwpval
print dw_p

# ----------------------------------------------------

# exponential smoothing
open "@workdir\data\okun5_aus.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
```

```

scalar tmid = round(($nobs+1)/2)
scalar a = .38
series exsm = movavg(g, a, tmid)
```

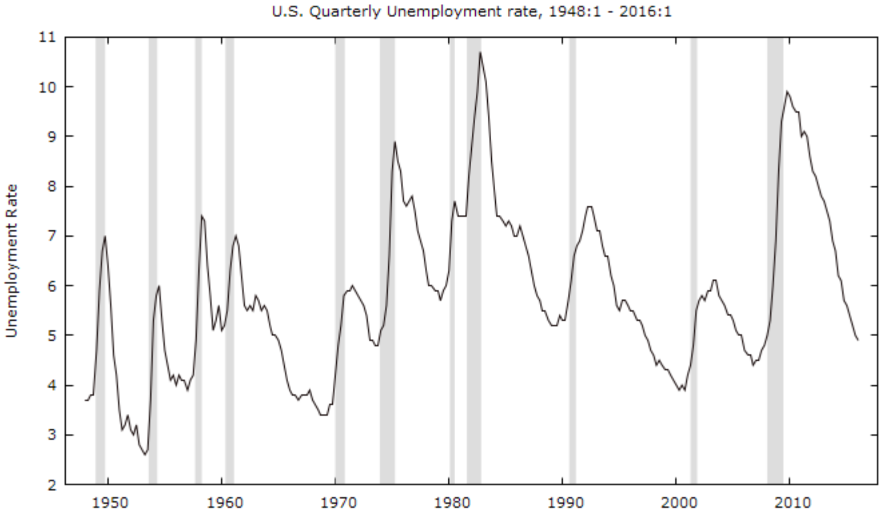


Figure 9.3: Time-Series graph of Unemployment. The shaded bars are periods of recession as determined by the NBER.

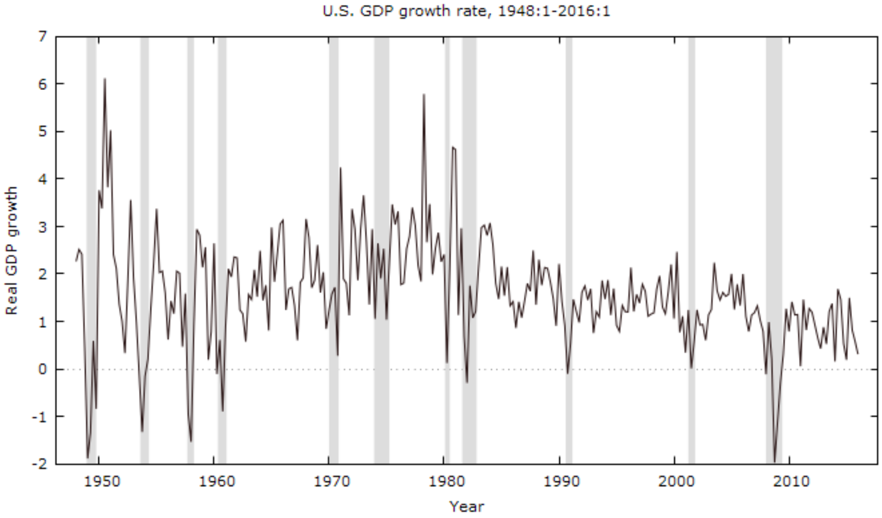


Figure 9.4: Time-Series graph of Unemployment.

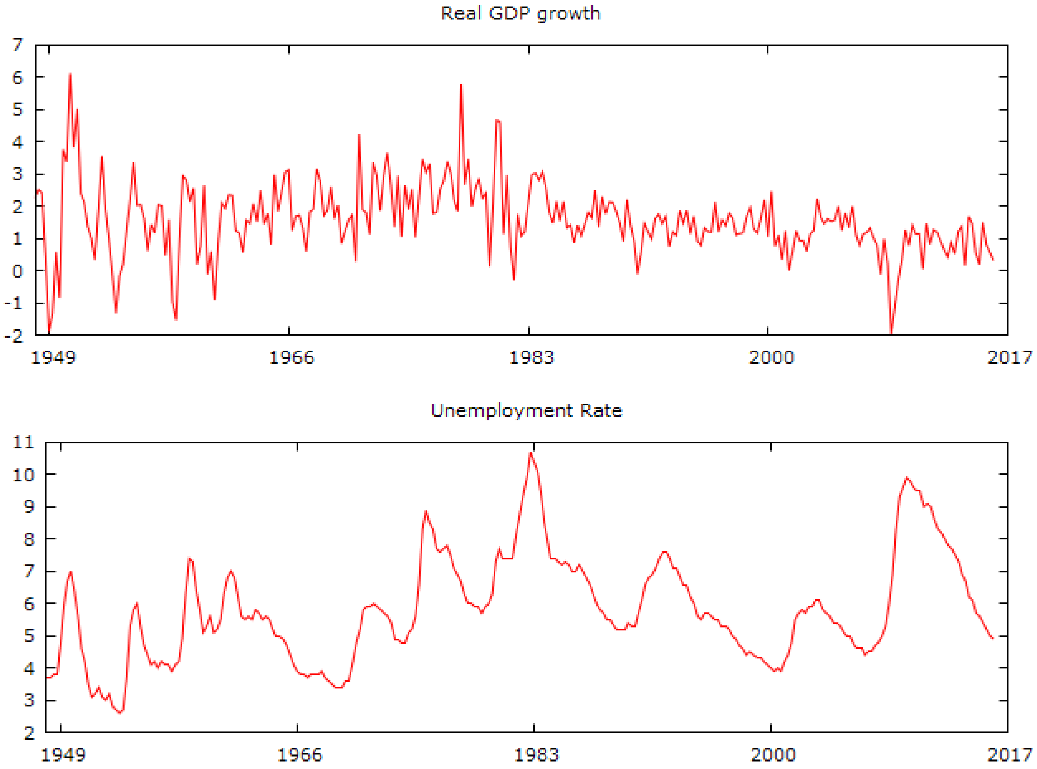


Figure 9.5: Multiple time-series graphs of U.S. macro data produced using **View>Multiple graphs>Time-series**. This uses the `scatters` command.

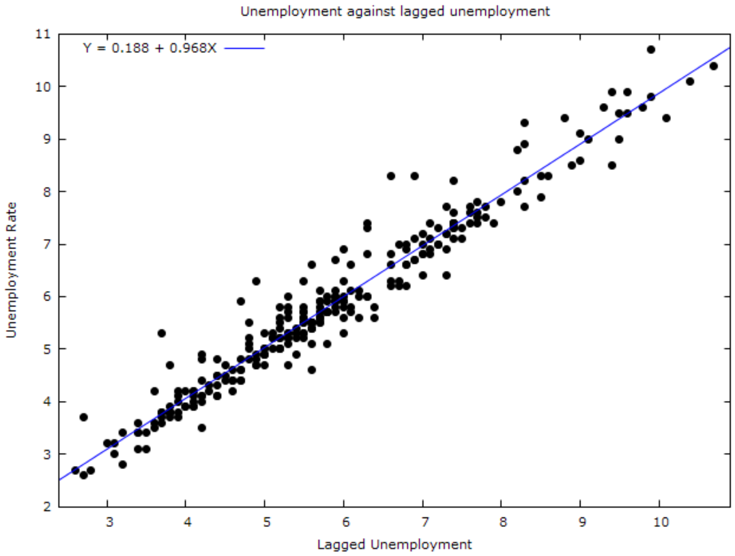


Figure 9.6: Plot of the U.S. quarterly unemployment rate against its 1 period lagged value.

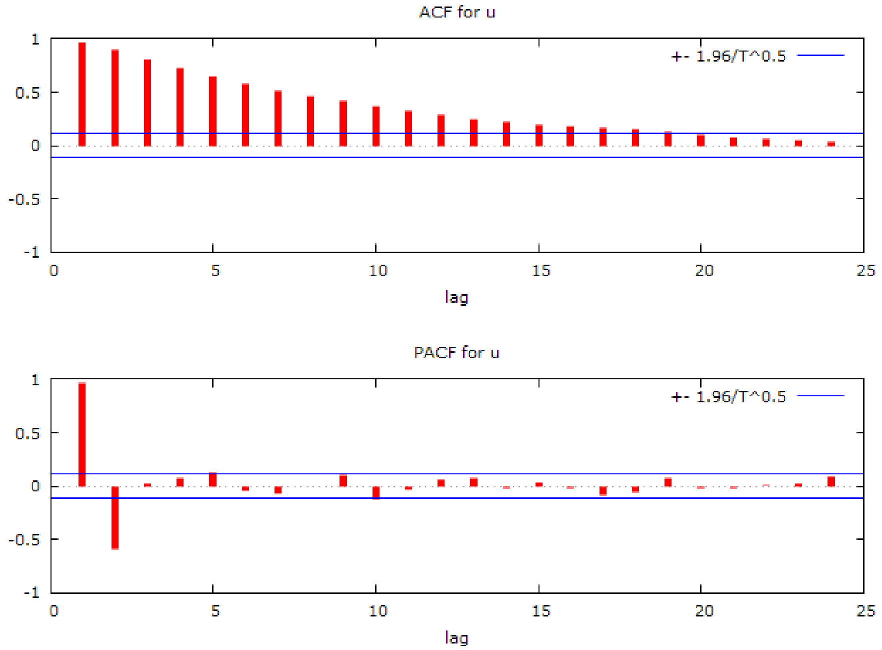


Figure 9.7: The 24 period correlogram for the U.S. unemployment rate.

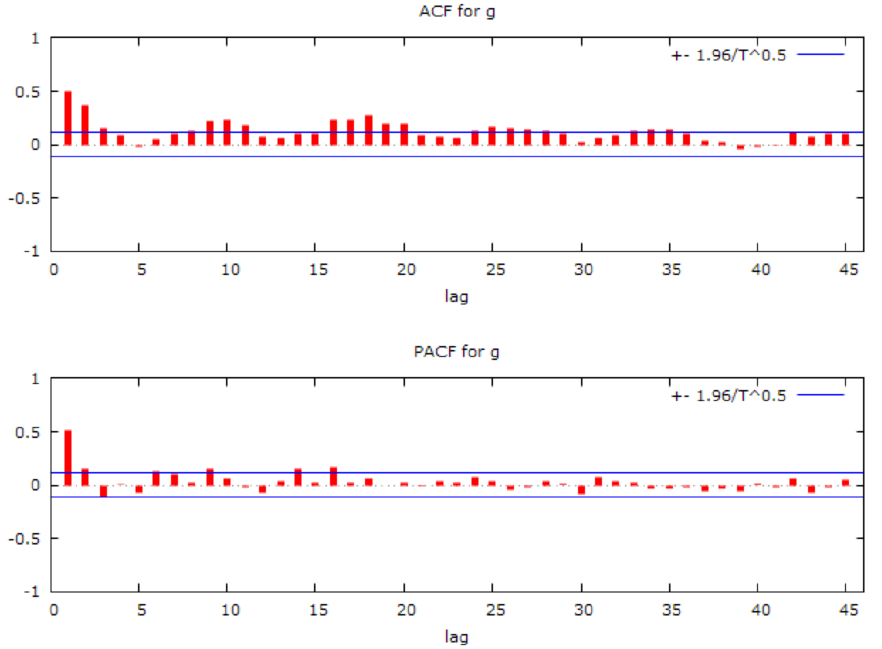
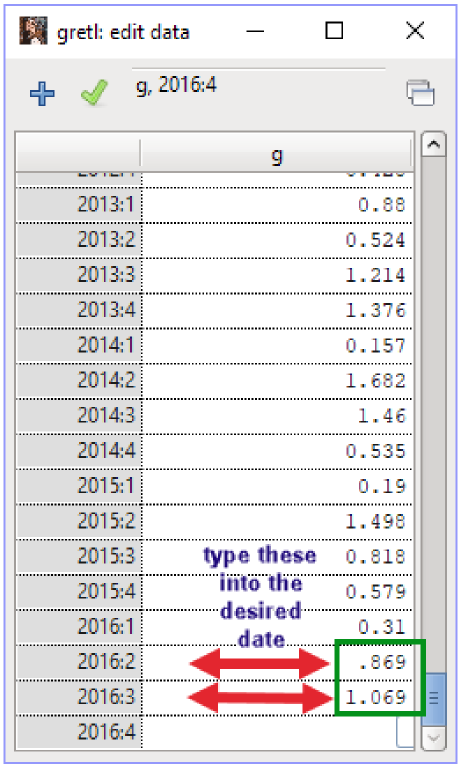
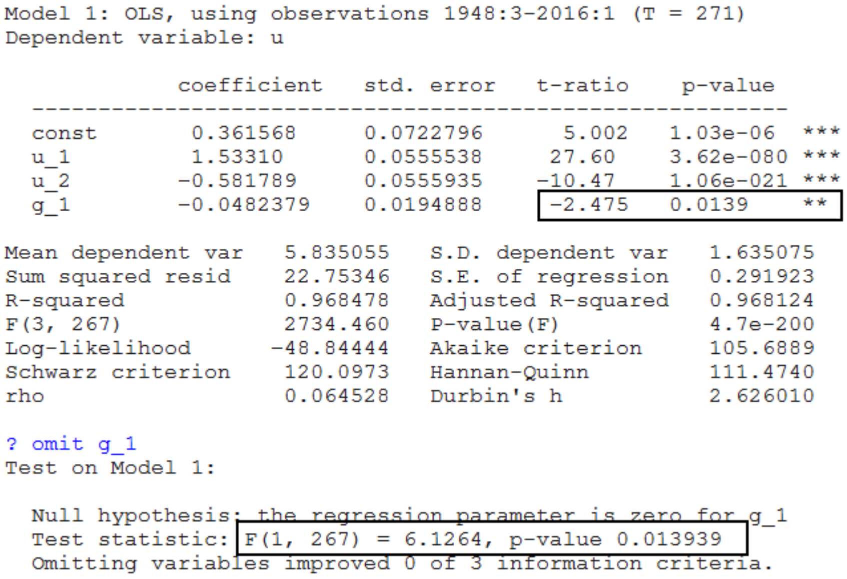


Figure 9.8: The 45 period correlogram for the U.S. quarterly GDP growth rate.



| | g |
|---|---|
| 2013:1 | 0.88 |
| 2013:2 | 0.524 |
| 2013:3 | 1.214 |
| 2013:4 | 1.376 |
| 2014:1 | 0.157 |
| 2014:2 | 1.682 |
| 2014:3 | 1.46 |
| 2014:4 | 0.535 |
| 2015:1 | 0.19 |
| 2015:2 | 1.498 |
| 2015:3 | 0.818 |
| 2015:4 | 0.579 |
| 2016:1 | 0.31 |
| 2016:2 | .869 |
| 2016:3 | 1.069 |
| 2016:4 | |

Figure 9.9: Edit data box



```
Model 1: OLS, using observations 1948:3-2016:1 (T = 271)
Dependent variable: u

              coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const         0.361568    0.0722796     5.002    1.03e-06  ***
  u_1           1.53310     0.0555538    27.60     3.62e-080 ***
  u_2          -0.581789    0.0555935   -10.47     1.06e-021 ***
  g_1          -0.0482379   0.0194888    -2.475    0.0139    **

Mean dependent var   5.835055   S.D. dependent var   1.635075
Sum squared resid    22.75346   S.E. of regression   0.291923
R-squared            0.968478   Adjusted R-squared   0.968124
F(3, 267)            2734.460   P-value(F)           4.7e-200
Log-likelihood      -48.84444   Akaike criterion     105.6889
Schwarz criterion    120.0973   Hannan-Quinn         111.4740
rho                  0.064528   Durbin's h           2.626010

? omit g_1
Test on Model 1:

  Null hypothesis: the regression parameter is zero for g_1
  Test statistic: F(1, 267) = 6.1264, p-value 0.013939
  Omitting variables improved 0 of 3 information criteria.
```

Figure 9.10: Granger Causality test result.

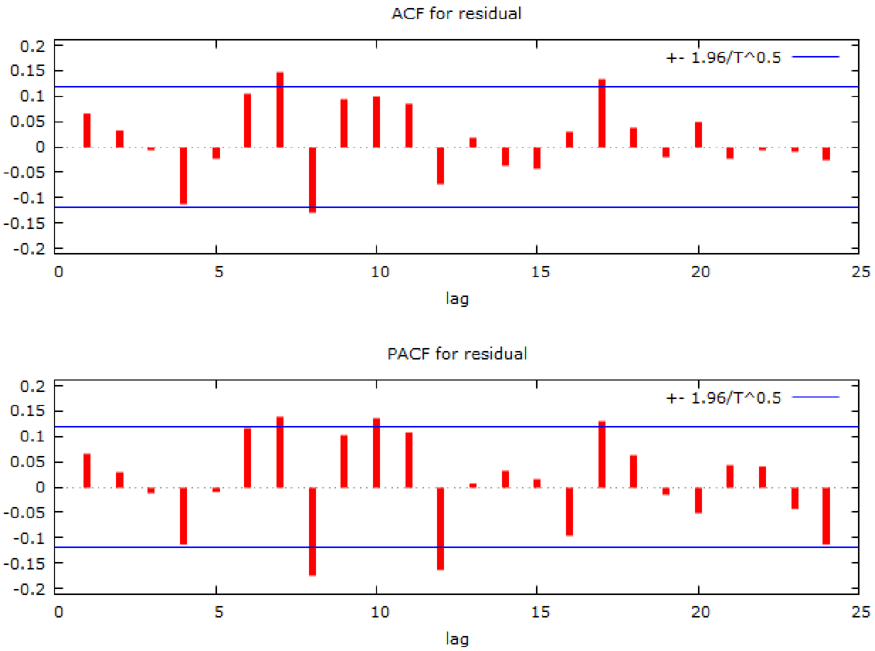


Figure 9.11: The correlogram of least squares residuals from estimation of an ARDL(2,1) of the unemployment rate using the *usmacro.gdt* dataset.

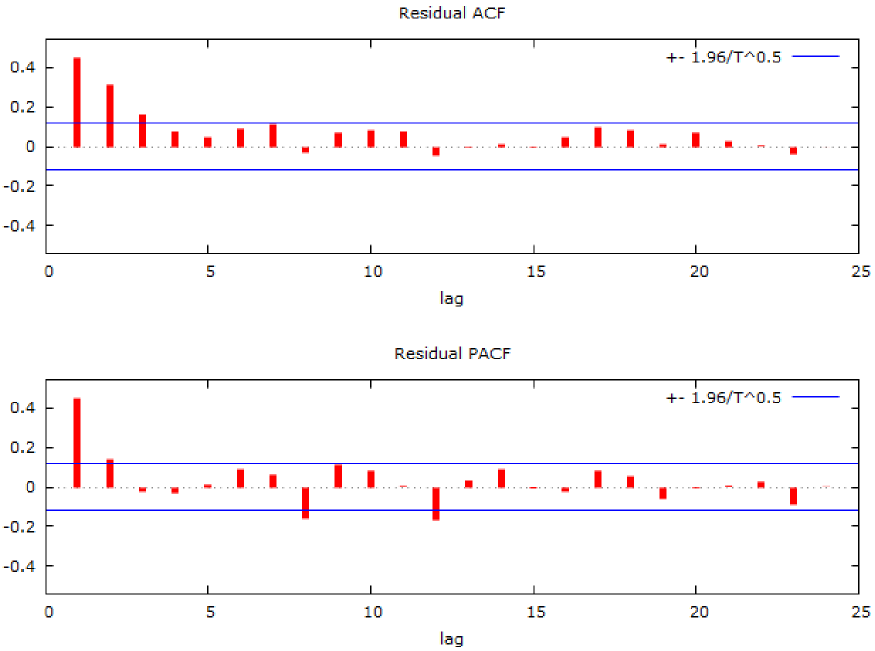


Figure 9.12: The correlogram of least squares residuals from estimation of an ARDL(1,1) of the unemployment rate using the *usmacro.gdt* dataset.

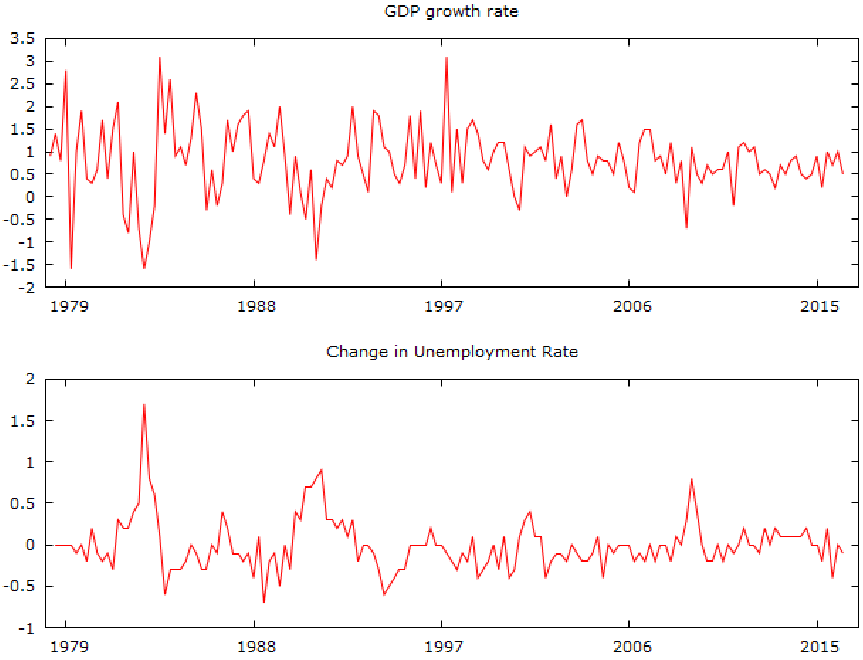
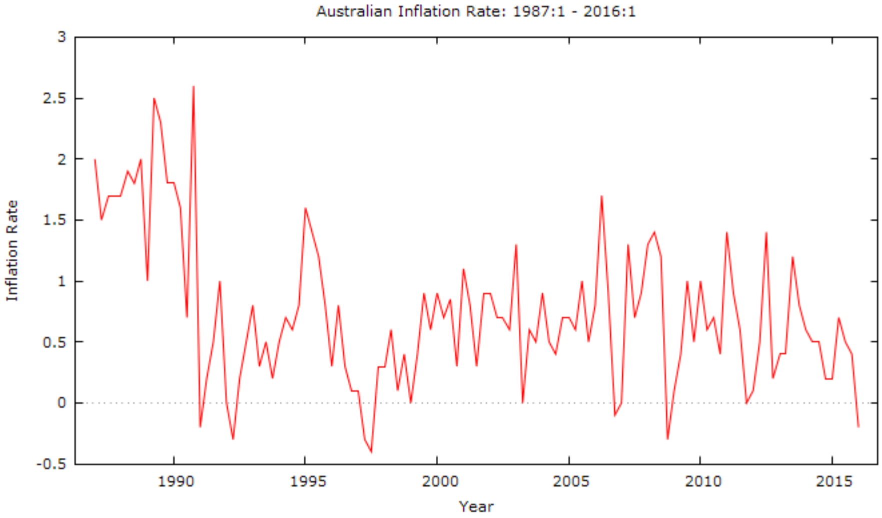


Figure 9.13: Changes in Australian Unemployment and Growth



inin

Figure 9.14: This plot shows the relationship between inflation and the change in unemployment in Australia, 1987:1 - 2016:1.

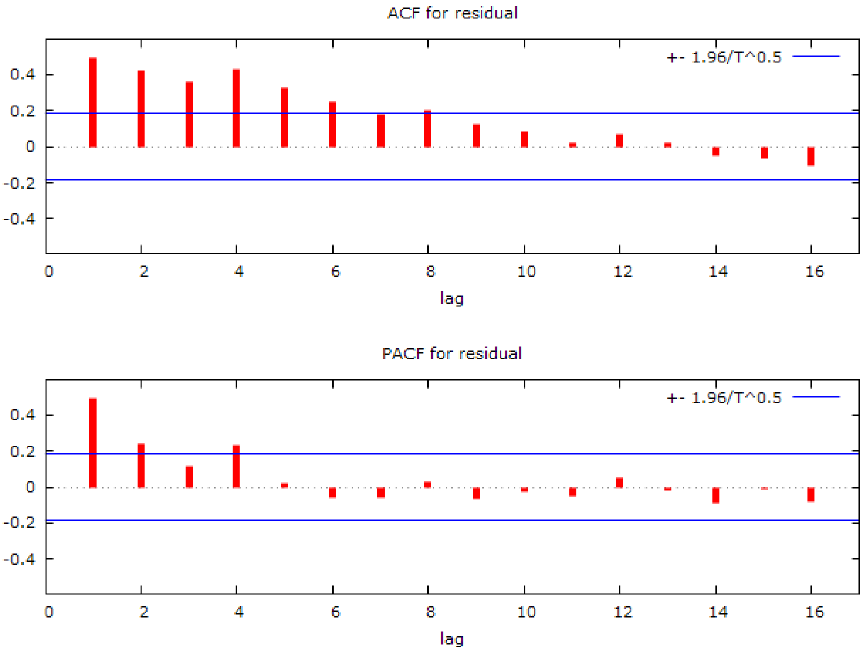
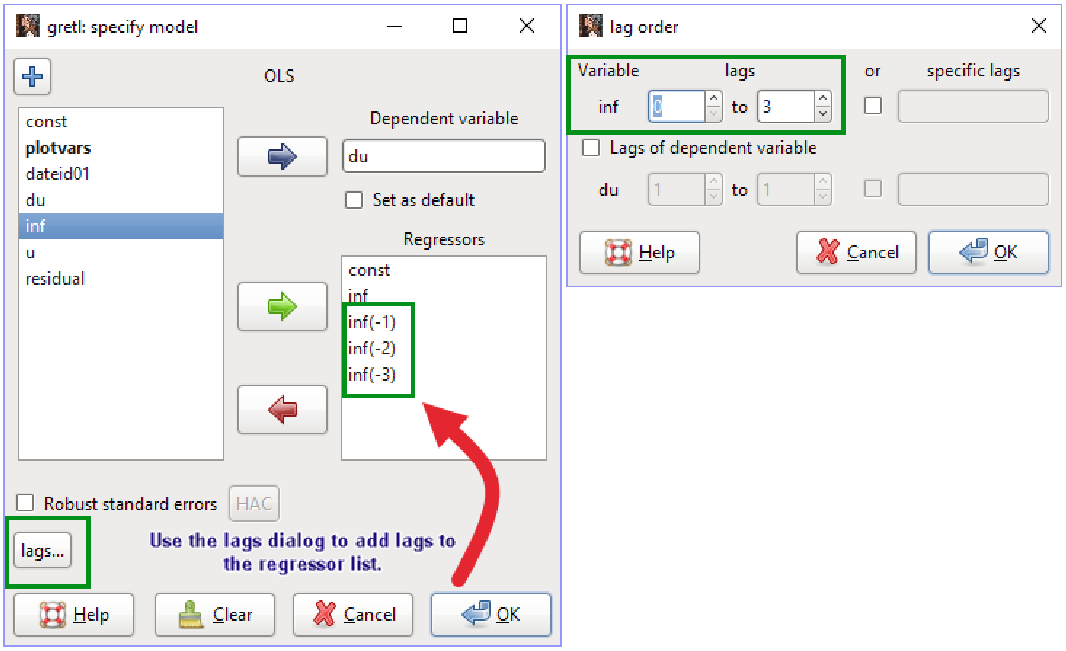


Figure 9.15: This plot shows that the residuals from the simple Phillips curve model are serially correlated. Australia, 1987:1 - 2016:1.



gretl: specify model

OLS

const
plotvars
dateid01
du
inf
u
residual

Dependent variable: du

□ Set as default

Regressors:
const
inf
inf(-1)
inf(-2)
inf(-3)

□ Robust standard errors

Use the lags dialog to add lags to the regressor list.

lag order

| Variable | lags | | | or | specific lags |
|---|---|---|---|---|---|
| inf | 0 | to | 3 | □ | |

□ Lags of dependent variable

| | | | | | |
|---|---|---|---|---|---|
| du | 1 | to | 1 | □ | |

Figure 9.16: 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.

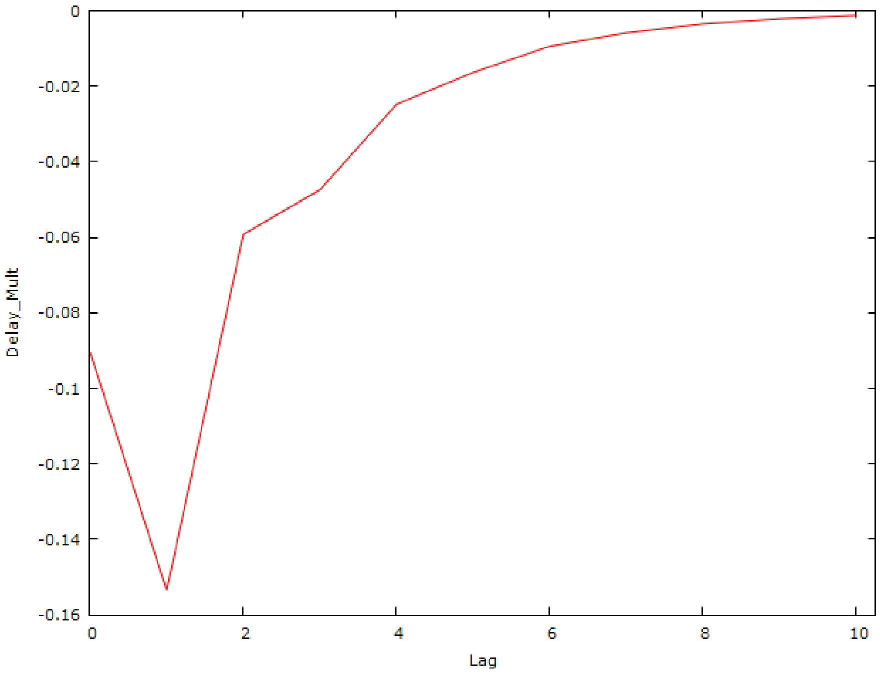


Figure 9.17: Lag Weights from Okun's Law

# Chapter 10

# Random Regressors and Moment Based Estimation

In this chapter **gretl**'s instrumental variables estimator is used to obtain consistent estimates of a model's parameters when its independent variables are correlated with the model's errors. Several tests of important assumptions are also explored. We end with a simulation that demonstrates important properties of OLS and IV estimation when the model contains endogenous regressors.

## 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 said to be **endogenous**.[1] If a model includes an endogenous regressor, least squares is 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 $e$. The parameters of equation (10.1) can be estimated consistently by using the **instrumental variables** or **two-stage least squares estimator**, rather than the 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 easily using what is commonly known 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 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

#### Example 10.1 in *POE5*

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 omitted ability are positively correlated, then it is being consistently **overestimated** by least squares.

### 10.2.2 Two-Stage Least Squares

Two-Stage Least Squares (TSLS) or Instrumental Variables (IV) estimation requires variables (a.k.a., instruments) that are correlated with the independent variables, but not correlated with the errors of your model.

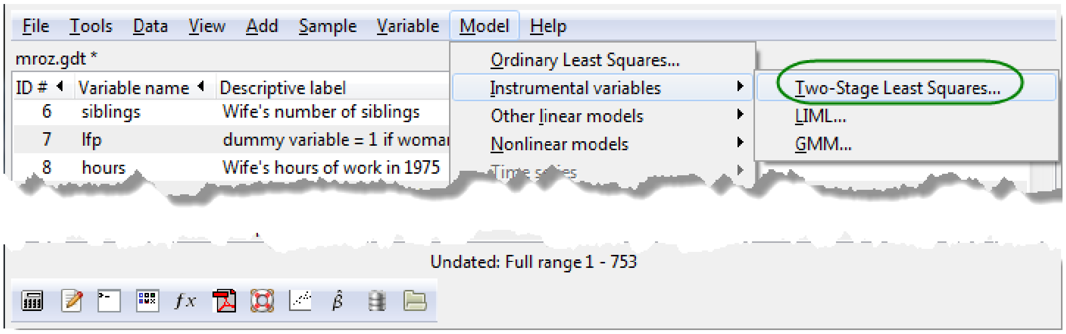
#### Example 10.2 in *POE5*

In the following simple wage regression model, we need one or more variables that are correlated with education, but not with the model's errors.

$$\ln(wage)_i = \beta_1 + \beta_2 educ_i + e_i \tag{10.3}$$

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 tested later. In the meantime, estimating the wage equation using the instrumental variable estimator is carried out in the following example.

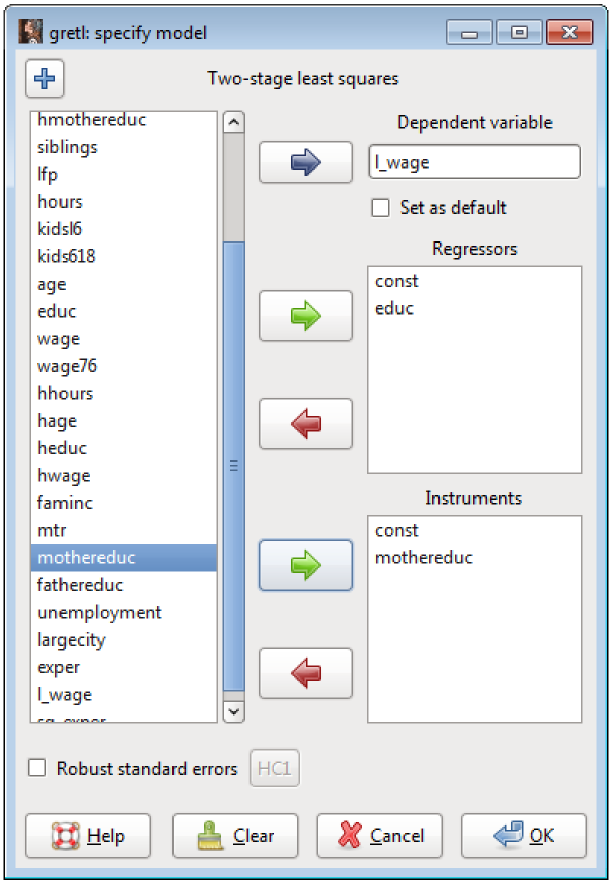
First, load the *mroz.gdt* data into **gretl**. Then, to open the basic **gretl** dialog box that computes the IV estimator choose **Model**>**Instrumental Variables**>**Two-Stage Least Squares** from the pull-down menu as shown below in Figure 10.1. This opens the dialog box shown in Figure 10.2.



| ID # | Variable name | Descriptive label |
|---|---|---|
| 6 | siblings | Wife's number of siblings |
| 7 | lfp | dummy variable = 1 if womai |
| 8 | hours | Wife's hours of work in 1975 |

Figure 10.1: Two-stage least squares estimator from the pull-down menus

In this example `l_wage` is the dependent variable, the desired instrument(s) are entered into the *Instruments* box and the independent variables, including the one(s) measured with error, into the *Independent Variables* box. Exogenous right-hand side variables should be referenced in both lists. Press the **OK** button and the results are found in Table 10.1. Notice that **gretl** ignores the



Dependent variable: l_wage

□ Set as default

Regressors: const, educ

Instruments: const, mothereduc

□ Robust standard errors HC1

Figure 10.2: Two-stage least squares dialog box

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 `tsls` syntax is very simple.

`tsls`

Arguments: *depvar indepvars* ; *instruments*
Options: `--no-tests` (don't do diagnostic tests)
`--vcv` (print covariance matrix)
`--no-df-corr` (no degrees-of-freedom correction)
`--robust` (robust standard errors)
`--cluster=`*clustvar* (clustered 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`

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.

The command syntax is: `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
list z = const 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`).

The output from OLS and IV estimation are compared below:

Dependent variable: l_wage

| | (1)<br>OLS | (2)<br>IV |
|---|---|---|
| const | −0.1852 | 0.7022 |
| | (0.1852) | (0.4851) |

| | | |
|---|---|---|
| educ | 0.1086** | 0.03855 |
| | (0.01440) | (0.03823) |
| $n$ | 428 | 428 |
| $R^2$ | 0.1179 | 0.1179 |
| $\ell$ | −441.3 | −3137 |

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

You can see that the coefficient for the return to another year of schooling has dropped from 0.1086 to 0.0385. The IV standard error has also increased in value as well, and the return to schooling coefficient is not significantly positive.

Several other statistics are computed. Following the example in *POE5*, the ratio of the slope standard errors is computed. The correlation between mother's education and daughters is found and the reciprocal of this number is taken. The latter measures the loss of efficiency due to IV estimation when the regressor is not actually endogenous.

```
m2 <- ols l_wage x
scalar se_educ_ols = $stderr(educ)
m3 <- tsls l_wage x; z
scalar se_educ_iv = $stderr(educ)

scalar a=corr(educ, mothereduc)
scalar ratio = se_educ_iv/se_educ_ols
scalar approx = 1/a
printf "\nThe correlation between mothers education and daughter is %.3f\n", a
printf "\nThe ratio of IV/OLS standard error for b2 is %.3f\n", ratio
printf "\nReciprocal of the correlation is %.3f\n", approx
```

The outcome is:

```
The correlation between mothers education and daughter is 0.387
The ratio of IV/OLS standard error for b2 is 2.655
Reciprocal of the correlation is 2.584
```

In this case the ratio of standard errors (2.655) is very close in magnitude to $1/r_{xz} = 2.584$.

### Example 10.3 in *POE5*

Two-stage least squares can be computed in two steps, but in practice it is not recommended. The estimates of the slopes and intercept will be the same as you get using the `tsls` IV estimator. However, the standard errors will not be computed correctly. To demonstrate, the model is estimated in two discrete steps and the results are compared. The **gretl** code for two step estimation is

```
smpl wage>0 --restrict
m4 <- ols educ const mothereduc
series educ_hat = $yhat
m5 <- ols l_wage const educ_hat
```

Notice that the sample has to be restricted to wages greater than zero using the `--restrict` option. Failing to do this causes the first stage regression to be estimated using all 753 observations instead of the 428 used in `tsls`. The IV estimator is implicitly limiting the first stage estimation to the non-missing values of `l_wage`.

Dependent variable: l_wage

|  | (1) | (2) |
|---|---|---|
|  | IV | OLS |
| const | 0.7022 | 0.7022 |
|  | (0.4851) | (0.5021) |
| educ | 0.03855 |  |
|  | (0.03823) |  |
| educ_hat |  | 0.03855 |
|  |  | (0.03957) |
| $n$ | 428 | 428 |
| $R^2$ | 0.1179 | 0.0022 |
| $\ell$ | −3137 | −467.6 |

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

The coefficient estimates on `educ` and `educ_hat` are the same, but the standard errors are not. If `educ` is endogenous, then the least squares standard errors are estimated inconsistently.

### Example 10.4 in *POE5*

In this example there are extra instruments–the model is **overidentified**. The worker's experience and experience squared are added to the instrument list. The reduced form equation is estimated and it's predictions are stored to a series. In the second step, OLS is used to estimate the model using the predictions as regressors. The estimated standard errors are incorrect, but the estimated slope and intercept are fine. Finally, the IV estimator is computed which fixes the problem with standard errors.

The reduced form equation is:

$$\widehat{\text{educ}} = \underset{(0.3211)}{9.480} + \underset{(0.03582)}{0.1564}\ \text{mothereduc} + \underset{(0.03363)}{0.1881}\ \text{fathereduc}$$

$$n = 428 \quad \bar{R}^2 = 0.2043 \quad F(2,425) = 55.830 \quad \hat{\sigma} = 2.0386$$

(standard errors in parentheses)

Both slopes are significantly different from zero at 5%.

Next the model is estimated using OLS. two-step, and IV estimators. The results are:

Dependent variable: l_wage

| | (1)<br>OLS | (2)<br>IV-two steps | (3)<br>IV |
|---|---|---|---|
| const | $-0.1852$ | 0.5510 | 0.5510 |
| | (0.1852) | (0.4258)$^{+}$ | (0.4086) |
| educ | 0.1086$^{**}$ | | 0.05049 |
| | (0.01440) | | (0.03217) |
| educ_hat | | 0.05049 | |
| | | (0.03352)$^{+}$ | |
| $n$ | 428 | 428 | 428 |
| $R^2$ | 0.1179 | 0.0053 | 0.1179 |
| $\ell$ | $-441.3$ | $-467$ | |

Standard errors in parentheses
+ Two-step IV standard errors not valid
* indicates significance at the 10 percent level
** indicates significance at the 5 percent level

### Example 10.5 in *POE5*

In this example a worker's years of experience and experience squared are added to the model. Parameters estimated using the IV estimator. The model is:

$$\ln(wage) = \beta_1 + \beta_2 exper + \beta_3 exper^2 + \beta_4 educ + e \tag{10.4}$$

Experience is considered exogenous and education endogenous. Two instruments considered are mother's education and father's education. The model is estimated:

```
list xlist = const exper sq_exper educ
list instruments = const mothereduc fathereduc exper sq_exper
m10 <- tsls l_wage xlist ; instruments

ols educ instruments                # First stage regression
omit mothereduc fathereduc --test-only
```

The IV estimation results

m10: TSLS, using observations 1–428
Dependent variable: l_wage
Instrumented: educ
Instruments: const mothereduc fathereduc exper sq_exper

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 0.04810 | 0.4003 | 0.1202 | 0.9044 |
| exper | 0.04417 | 0.01343 | 3.288 | 0.0011 |
| sq_exper | −0.0008990 | 0.0004017 | −2.238 | 0.0257 |
| educ | 0.06140 | 0.03144 | 1.953 | 0.0515 |

| | | | |
|---|---|---|---|
| Mean dependent var | 1.190173 | S.D. dependent var | 0.723198 |
| Sum squared resid | 193.0200 | S.E. of regression | 0.674712 |
| $R^2$ | 0.145660 | Adjusted $R^2$ | 0.139615 |
| $F(3, 424)$ | 8.140709 | P-value($F$) | 0.000028 |

Another year of schooling is predicted to increase average wage by 0.0614 × 100=6.14%.

The reduced form results are also computed and given below:

fs10: OLS, using observations 1–428
Dependent variable: educ

|  | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 9.10264 | 0.426561 | 21.34 | 0.0000 |
| mothereduc | 0.157597 | 0.0358941 | 4.391 | 0.0000 |
| fathereduc | 0.189548 | 0.0337565 | 5.615 | 0.0000 |
| exper | 0.0452254 | 0.0402507 | 1.124 | 0.2618 |
| sq_exper | −0.00100909 | 0.00120334 | −0.8386 | 0.4022 |

|  |  |  |  |
|---|---|---|---|
| Mean dependent var | 12.65888 | S.D. dependent var | 2.285376 |
| Sum squared resid | 1758.575 | S.E. of regression | 2.038967 |
| $R^2$ | 0.211471 | Adjusted $R^2$ | 0.204014 |
| $F(4, 423)$ | 28.36041 | P-value($F$) | 6.87e–21 |

The $t$-ratios on the external instruments are quite large. As seen in the next section, this bodes well for estimation of the model via the IV estimator.

## 10.3 Specification Tests

There are a number of specification tests one subjects an endogenous regressor model. Instrument strength, the exoogeneity of regressors, and the suitability of the instruments can in most cases be checked using a battery of statistical tests. These procedures are discussed in this section.

### 10.3.1 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:

$$x_k = \gamma_1 + \gamma_2 x_2 + \cdots + \gamma_{k-1} x_{k-1} + \theta_1 z_1 + \cdots + \theta_\ell z_\ell + \nu_k \tag{10.5}$$

In this notation, the $z_1, \ldots, z_\ell$ are the **external** instruments. The others, $x_2, \ldots, x_{k-1}$, 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, $\theta_1, \ldots, \theta_\ell$ are jointly zero is less than a suitable critical value, $c$, then you conclude that the instruments are weak. If $F > c$, the instruments are strong (enough).

The problem with weak instruments is that they affect the distribution of the F-statistic, and hence the critical value, $c$. Weak instruments cause two problems as summarized by Hill et al. (2018, p. 522):

**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.

Initially, Staiger and Stock (1997) suggested that a critical value of $c = 10$ would be suitable for this test. This has become a rule-of thumb that has since been refined by further simulations.

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. 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.

### Example 10.6 in *POE5*

In the preceding example the IV estimator was used to estimate equation (10.4). The first stage regression is estimated and the external instruments are tested for their joint significance using the `omit` command.

```
fs10 <- ols educ instruments                # First stage regression
omit mothereduc fathereduc --test-only
```

The `--test-only` option suppresses the printout from the regressions; only the test results are printed. The output from **gretl** appears below:

```
Test on Model 19:

  Null hypothesis: the regression parameters are zero for the variables
    mothereduc, fathereduc
  Test statistic: F(2, 423) = 55.4003, p-value 4.26891e-022
```

The instruments appear to be fairly strong. Both `mothereduc` and `fathereduc` are individually significant at 5%. Using the rule-of-thumb critical value, a joint test $F$-statistic is $55.40 > 10$.

Whenever a model is estimated using two stage least squares, **gretl** computes the test statistic for detecting weak instruments. The results appear below the regression and are

```
Weak instrument test -
  First-stage F-statistic (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%
```

The rule-of-thumb $F > 10$ is refined to reflect the experimental design implied by the model and sample. The weak instrument test tables (e.g., Tables 10A.1 and 10A.2 in *POE5*) provide more specific information about the actual size of the weak instruments test. 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 IV estimator based on these very strong instruments to exhibit relatively small bias.

### 10.3.2 Partial Correlations

Valid instruments should be correlated with the endogenous regressor. However, the statistical properties of the IV estimator depend upon the strength of this correlation. Furthermore, it is strength of the **independent** correlation between the instrument and the endogenous regressor that matters. The higher, the better.

To get at this in a multiple regression model, 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** as the following script shows.

```
ols educ const exper sq_exper
series reduc = $uhat
ols mothereduc const exper sq_exper
series rmom = $uhat
ols reduc rmom --quiet
scalar c1 = corr(reduc, rmom)
printf "  Partial R2 = %.4f\n\
 The correlation between reduc and rmom = %.4f\n\
 The correlation squared is %.4f\n", $rsq, c1,c1^2
```

In line 1 education is regressed onto a `const`, `exper`, and `sq_exper`; the residuals are saved as `reduc` in line 2. The residuals contain all variation in `educ` not accounted for by the exogenous regressors. The variation in education due to the `const`, `exper`, and `sq_exper` has been **partialled out**.

The second regression does the same for the instrument, `mothereduc`. The correlation between mother's education and a `const`, `exper`, and `sq_exper` has been partialled out, resulting in residuals, `rmom`. Regressing `reduc` onto `rmom` yields, 0.26769. This is the same as 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. This demonstrates the Frisch-Waugh-Lovell Theorem.

The correlation between the two sets of residuals yields a **partial correlation**. This is a correlation between education and mother's education where the common effects of `const`, `exper`, and `sq_exper` have been removed.

```
Partial R2 = 0.1485
 The correlation between reduc and rmom = 0.3854
 The correlation squared is 0.1485
```

The partial correlation between `reduc` and `rmom` is 0.3854. Squaring this correlation yields a **partial-$R^2$**. In the Cragg-Donald F-test described below, partial correlations play a key role in testing for weak instruments.

Father's years of schooling (`fathereduc` ) is added to the instrument list and the exercise is repeated. `fathereduc` is regressed onto the exogenous regressors and the residuals are saved to `rdad`. The partial-$R^2$ from a regression of `reduc` onto `rmom` and `rdad` is the $R^2$ of the partialled-out endogenous variable on all partialled-out external IVs. The script

```
ols fathereduc const exper sq_exper
series rdad = $uhat
ols reduc rmom rdad
printf "Partial R2 = %.3f\n", $rsq
```

yields:

```
Partial R2 = 0.208
```

There are other useful specification tests to use with instrumental variables estimators. By default, Gretl computes each of these whenever you estimate a model using two-stage least squares.[2]

[2] See section 5.2.4 of *POE5* for some background on the Frisch-Waugh-Lovell (FWL) theorem.

### 10.3.3 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 $H_0$: $Cov(x_i, e_i) = 0$ against $H_1$: $Cov(x_i, e_i) \neq 0$. The first step is to use least squares to estimate the first stage of TSLS

$$x_i = \gamma_1 + \theta_1 z_{i1} + \theta_2 z_{i2} + \nu_i \tag{10.6}$$

and to save the residuals, $\hat{\nu}_i$. Then, add the residuals to the original model

$$y_i = \beta_1 + \beta_2 x_i + \delta \hat{\nu}_i + e_i \tag{10.7}$$

Estimate this equation using least squares and use the $t$-ratio on the coefficient $\delta$ to test the hypothesis. If it is significantly different from zero then the regressor, $x_i$ is not exogenous or predetermined with respect to $e_i$ and you should use the IV estimator (TSLS) to estimate $\beta_1$ and $\beta_2$. If it is not significant, then use the more efficient estimator, OLS.

The **gretl** script for the Hausman test applied to the wage equation using `mothereduc` and `fathereduc` as external instruments is:

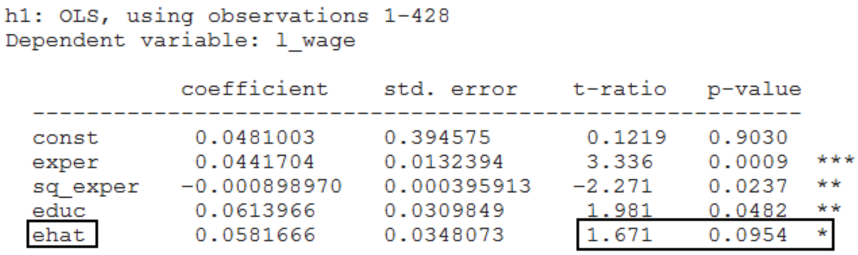
```
list xlist = const exper sq_exper educ
list instruments = const exper sq_exper mothereduc fathereduc
ols educ instruments --quiet
series ehat = $uhat
ols l_wage xlist ehat --quiet
omit ehat --test-only
```

The test outcome from the omit command is:

```
Null hypothesis: the regression parameter is zero for ehat
Test statistic: F(1, 423) = 2.79259, p-value 0.0954406
```

The $p$-value is $0.954 > 5\%$ and the exogeneity of education cannot be rejected at 5%. An equivalent way to generate this statistic if from the regression output itself. The regression result appear in (Figure 10.3). The $t$-ratio on `ehat`=1.671 has the same $p$-value as the $F$-statistic. Of course, it is not significant at the 5% level. We would conclude that the instruments are exogenous.

The model is **overidentified**. There are two additional instruments, mother's education and father's education, that are being used for only one endogenous regressor, `educ`. Overidentification means that you have more instruments than necessary to estimate the model. This provides an opportunity to conduct another test of the adequacy of the instruments themselves.



```
h1: OLS, using observations 1-428
Dependent variable: l_wage

             coefficient    std. error    t-ratio   p-value
  ---------------------------------------------------------
  const       0.0481003     0.394575       0.1219   0.9030
  exper       0.0441704     0.0132394      3.336    0.0009  ***
  sq_exper   -0.000898970   0.000395913   -2.271    0.0237  **
  educ        0.0613966     0.0309849      1.981    0.0482  **
  ehat        0.0581666     0.0348073      1.671    0.0954  *
```

Figure 10.3: Hausman test for endogeneity of regressor.

### 10.3.4 Sargan Test

The final test is the **Sargan test** of the overidentifying restrictions implied by an overidentified model. Recall that to be overidentified implies that you have more instruments than you have endogenous regressors. In our example there is one 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. $nR^2$ from this regression is approximately $\chi^2$ with degrees of freedom equal to the number of surplus instruments. Gretl does this easily since it saves $nR^2$ 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 tsls l_wage xlist; instruments
2 series uhat = $uhat
3 ols uhat instruments
4 scalar test = $rsq*$nobs
5 pvalue X 2 test
```

In line 1 the model is estimated using tsls with the variables in list `xlist` as regressors and those in `instruments` as the IVs. In line 2 the residuals are saved as `uhat`. Then in line 3 a regression is estimated by ordinary least squares using the residuals and instruments as regressors. $nR^2$ 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 correlated with the errors either because they are endogenous or because 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-square(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-square(1) > 0.378071) = 0.538637

Weak instrument test -
  First-stage F-statistic (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 a little 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 overidentification are the same. In conclusion, there is no need to compute any of these tests manually, unless you want to.

### 10.3.5 Multiple Endogenous Regressors and the Cragg-Donald F-test

When there are multiple endogenous regressors in the model, instrument strength cannot be tested using the F-statistic from first-stage regressions. Cragg and Donald (1993) have proposed a generalization of this approach that can be used to test for weak identification (i.e., weak instruments). In order to compute the CDF statistic manually, you must have a set of canonical correlations. These are not computed in **gretl** natively, but they are easy to compute using **hansl**. This is demonstrated below. 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.

### 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 $r_2$.

Gretl's matrix language is very powerful and you can easily get the canonical correlations from two sets of regressors. The following funcrion[3] 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)
   return sqrt(ret)
end function
```

The function is called `cc` and takes two lists as arguments. The lists contain 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 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.

### Cragg-Donald F statistic

A test for weak identification is based on the Cragg-Donald $F$-test statistic

$$\text{Cragg-Donald} - F = [(N - G - B)/L] \times [r_B^2/(1 - r_B^2)] \tag{10.8}$$

To compute the Cragg-Donald $F$, assemble the two sets of residuals and use the `cc` function to get the canonical correlations.

[3] Function supplied by **gretl** guru Riccardo Lucchetti.

```
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

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
```

The Cragg-Donald F (CDF) 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.

For this model the routine yields:

```
The Cragg-Donald Statistic is 0.1006.
```

This corresponds to the model and statistic in column (3) of Table 10A.4 in *POE5*.

To reproduce the other results from this exercise, we will use internal **gretl** commands.

```
# Weak IV Example 1
iv1 <- tsls hours x1 ; z1
rf1 <- ols mtr z1

# Weak IV Example 2
list z2 = z1 sq_exper largecity
iv2 <- tsls hours x1 ; z2
rf2 <- ols mtr z2
omit exper sq_exper largecity --test-only

# Weak IV Example 3
list z3 = kidsl6 nwifeinc mothereduc fathereduc const
iv3 <- tsls hours x1 ; z3
rf3_mtr <- ols mtr z3
omit mothereduc fathereduc --test-only
```

```
rf3_educ <- ols educ z3
omit mothereduc fathereduc --test-only

# Weak IV Example 4
list z4 = z3 exper
iv4 <- tsls hours x1 ; z4
rf4_mtr <- ols mtr z4
omit exper mothereduc fathereduc --test-only
rf4_educ <- ols educ z4
omit exper mothereduc fathereduc --test-only
```

We collect these into sets of model tables.

First, the four instrumental variables estimators for the hours equations:

TSLS estimates
Dependent variable: Hours Worked

| | IV1 | IV2 | IV3 | IV4 |
|---|---|---|---|---|
| const | 17423.72** | 14394.11** | −24491.60 | 18067.84** |
| | (3136.20) | (2532.79) | (79689.72) | (3534.91) |
| mtr | −18456.59** | −14934.37** | 29709.47 | −18633.92** |
| | (3636.53) | (2934.73) | (90487.78) | (3843.85) |
| educ | −145.29** | −118.88** | 258.56 | −189.86** |
| | (33.00) | (27.78) | (846.01) | (62.36) |
| kidsl6 | 151.02 | 58.79 | −1144.48 | 190.28 |
| | (141.01) | (122.04) | (2510.19) | (158.30) |
| nwifeinc | −103.90** | −85.19** | 149.23 | −102.15** |
| | (19.72) | (16.00) | (470.52) | (19.90) |
| $n$ | 428 | 428 | 428 | 428 |
| $R^2$ | 0.20 | 0.20 | 0.16 | 0.20 |
| $\ell$ | −4.4e+003 | | −6e+003 | |

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

Then the four marginal tax rate reduced form equations:

OLS estimates
Reduced Form Equations

Dependent variable: Marginal Tax Rate

| | RF-IV1 | RF-IV2 | RF-IV3 | RF-IV4 |
|---|---|---|---|---|
| const | $0.8793^{**}$ | $0.8847^{**}$ | $0.7991^{**}$ | $0.8296^{**}$ |
| | (0.0118) | (0.0123) | (0.0077) | (0.0089) |
| kidsl6 | $0.0204^{**}$ | $0.0204^{**}$ | $0.0219^{**}$ | $0.0156^{**}$ |
| | (0.0053) | (0.0052) | (0.0056) | (0.0054) |
| nwifeinc | $-0.0055^{**}$ | $-0.0054^{**}$ | $-0.0057^{**}$ | $-0.0058^{**}$ |
| | (0.0002) | (0.0002) | (0.0002) | (0.0002) |
| educ | $-0.0072^{**}$ | $-0.0069^{**}$ | | |
| | (0.0009) | (0.0009) | | |
| exper | $-0.0014^{**}$ | $-0.0022^{**}$ | | $-0.0017^{**}$ |
| | (0.0003) | (0.0008) | | (0.0003) |
| sq_exper | | 0.0000 | | |
| | | (0.0000) | | |
| largecity | | $-0.0116^{**}$ | | |
| | | (0.0043) | | |
| mothereduc | | | $-0.0011$ | $-0.0013^{*}$ |
| | | | (0.0008) | (0.0008) |
| fathereduc | | | $-0.0018^{**}$ | $-0.0020^{**}$ |
| | | | (0.0007) | (0.0007) |
| $n$ | 428 | 428 | 428 | 428 |
| $\bar{R}^2$ | 0.7102 | 0.7146 | 0.6573 | 0.6855 |
| $\ell$ | 758 | 762.3 | 722.1 | 741 |

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

The two reduced form equations for education are:

Reduced Form Equations
Dependent variable: Education

| | (RF-IV3) | (RF-IV4) |
|---|---|---|
| const | $8.7146^{**}$ | $8.1762^{**}$ |
| | (0.3374) | (0.4020) |
| kidsl6 | $0.6181^{**}$ | $0.7292^{**}$ |
| | (0.2432) | (0.2461) |
| nwifeinc | $0.0496^{**}$ | $0.0530^{**}$ |

| | | |
|---|---|---|
| | (0.0091) | (0.0091) |
| mothereduc | 0.1520** | 0.1560** |
| | (0.0345) | (0.0344) |
| fathereduc | 0.1637** | 0.1675** |
| | (0.0327) | (0.0325) |
| exper | | 0.0296** |
| | | (0.0122) |
| $n$ | 428 | 428 |
| $\bar{R}^2$ | 0.2622 | 0.2706 |
| $\ell$ | −893.5 | −890.5 |

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

Thankfully, **gretl** computes the CDF with each IV estimation.

### Example from Appendix 10B of *POE5*

In this example simulated data are used to illustrate that OLS fails and IV estimation 'works' when suitable instruments can be found. The data are from *ch10.gdt*, which were generated by the authors of *POE5*.

$$y = 1 + x + e \tag{10.9}$$

Thus, $\beta_1 = 1$ and $\beta_2 = 1$. The regressor and errors are generated to be correlated with one another, $Cov(x, e) \neq 0$. There are three instruments, $z_1$, $z_2$, and $z_3$. The correlation betweens $x$ and instruments $z_1$ and $z_2$ are 0.5 and 0.3, respectively. The other instrument, $z_3$, is not valid since it is correlated with $e$, i.e., $\rho_{z_3 e} = 0.3$.

We run five regressions for the regression model in equation (10.9): OLS, IV($z_1$). IV($z_2$), IV($z_3$), and IV($z_1$, $z_2$). The instruments used in the IV estimator are shown in parentheses.

The results:

Dependent variable: y

| | ols_1 | iv_1 | iv_2 | iv_3 | iv_4 |
|---|---|---|---|---|---|
| | OLS | TSLS | TSLS | TSLS | TSLS |
| const | 0.9789** | 1.1011** | 1.3451** | 0.9640** | 1.1376** |
| | (0.0883) | (0.1091) | (0.2555) | (0.0952) | (0.1164) |

| | | | | | |
|---|---|---|---|---|---|
| x | 1.7034** | 1.1924** | 0.1724 | 1.7657** | 1.0399** |
| | (0.0899) | (0.1945) | (0.7965) | (0.1722) | (0.1942) |
| $n$ | 100 | 100 | 100 | 100 | 100 |
| $R^2$ | 0.7854 | 0.7854 | 0.7854 | 0.7854 | 0.7854 |
| $\ell$ | −125.4 | −507.2 | −519.8 | −514.8 | |

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

The OLS estimator estimates $\beta_2$ as 1.7034, which is nearly twice as large as its true value of 1. The weak instrument, $z_2$ does not perform very well for estimating $\beta_2$. Its coefficient is well below 1, though 1 lies in its 95% confidence interval. The invalid instrument in column (iv_3) also performs poorly. The stronger instruments appear to perform better than the weaker one, but this is based on only 1 sample. In the next section we'll simulate this process to see how things behave in repeated samples.

The reduced form equation for IV($z_1$, $z_2$) is:

rf: OLS, using observations 1–100
Dependent variable: x

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 0.1947 | 0.07950 | 2.449 | 0.0161 |
| z1 | 0.5700 | 0.08879 | 6.420 | 0.0000 |
| z2 | 0.2068 | 0.07716 | 2.680 | 0.0087 |

| | | | |
|---|---|---|---|
| Mean dependent var | 0.239161 | S.D. dependent var | 0.956655 |
| Sum squared resid | 60.37887 | S.E. of regression | 0.788963 |
| $R^2$ | 0.333594 | Adjusted $R^2$ | 0.319854 |
| $F(2, 97)$ | 24.27844 | P-value($F$) | 2.83e–09 |
| Log-likelihood | −116.6673 | Akaike criterion | 239.3346 |
| Schwarz criterion | 247.1501 | Hannan–Quinn | 242.4977 |

Since the equation contains only external instruments as regressors, the joint test of their significance is equivalent to the overall-F statistic. Its value is 24.28 and the size of 5% tests are likely to be less than 10%.

Printing the results from column 5 allow us to test for endogeneity of x, weak instruments and for overidentification. The tests are:

iv_4: TSLS, using observations 1–100

Dependent variable: y
Instrumented: x
Instruments: const z1 z2

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: $\chi^2(1) = 38.5$
  with p-value = 5.47545e-010

Sargan over-identification test –
  Null hypothesis: all instruments are valid
  Test statistic: LM = 3.62757
  with p-value = $P(\chi^2(1) > 3.62757) = 0.0568296$

Weak instrument test –
  First-stage $F(2, 97) = 24.2784$

The Hausman test statistic is 38.5. This is exactly the square of the t-ratio on the RF residual included in the auxiliary regression (not shown). Weak instruments are rejected since 24.28 > 19.93, which indicates that the maximal size is less than 10%. The Sargan test for overidentification is not significant at 5% but it is at 10%.

Finally, the regression is repeated using all instruments (one of which is invalid).

```
iv_5 <- tsls y const x ; const z1 z2 z3
```

The Sargan test results indicate that at 5% the instruments are not valid.

iv_5: TSLS, using observations 1–100
Dependent variable: y
Instrumented: x
Instruments: const z1 z2 z3

Sargan over-identification test –
  Null hypothesis: all instruments are valid
  Test statistic: LM = 13.1107
  with p-value = $P(\chi^2(2) > 13.1107) = 0.00142246$

## 10.4 Simulation

In appendix 10C of *POE5*, the authors conduct a Monte Carlo experiment comparing the performance of OLS and TSLS. The basic simulation is based on the model

$$y_i = x_i + e_i \tag{10.10}$$

$$x_i = \theta z_{i1} + \theta z_{i2} + \theta z_{i3} + v_i \tag{10.11}$$

The $z_i$ are exogenous instruments that are each N(0,1). The errors, $e_i$ and $v_i$, are

$$\begin{pmatrix} e_i \\ v_i \end{pmatrix} \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right] \tag{10.12}$$

The parameter $\theta$ controls the strength of the instruments and is set to either 0.1 or 0.5. The parameter $\rho$ 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.

To reduce clutter in the body of the loop, I have written a function that determines whether the null hypothesis of exogeneity is rejected at 5% using a Hausman test. One of the things measured in the simulation is the rejection rate of this test and this will facilitate that.

The most novel thing here, at least for this manual, is how the endogenous variables are tracked for the first regression used in the test. list has a feature that makes this easy to do. In line 3 a list is created that subtracts one list from another. In this case, anything that is in x but not in z will be added to the endogenous variables list. This is likely what **gretl** is doing under the hood in the tsls command (but with error catching). At any rate, it seems to work. The rest of the function is just an execution of the regression based Hausman test of the exogeneity of regressors.

It returns a scalar (1 or 0) and its input arguments are `y` (series for the dependent variable of the model), `xvars` (a list of regressors in the model), and `zvars` (another list that contains all internal and external instruments). The standard normal is used to obtain the critical value for the test. Feel free to use the $t$ if you wish.

```
1  # Function returns a 1 if reject Hausman null
2  function scalar Hausman (series y, list xvars, list zvars)
3      list endogvars = xvars - zvars
4      ols endogvars zvars --quiet
5      series vhat = $uhat
6      ols y xvars vhat --quiet
7      scalar t = $coeff(vhat)/$stderr(vhat)
8      scalar reject = abs(t)>1.96
9      return reject
10 end function
```

The **gretl** script to perform the simulation appears below:

```
# Simulation
scalar N = 100
nulldata N --preserve
scalar rho = 0.0  # 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 zvars = const 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
    list xvars = const x
    ols x const zvars --quiet
    scalar f = $Fstat
    ols y xvars --quiet
    scalar b_ols = $coeff(x)
    scalar se_ols = $stderr(x)
    scalar t_ols = (b_ols-1)/se_ols
    scalar r_ols = abs(t_ols)>critical(t,$df,.025)
    tsls y xvars; zvars --quiet
    scalar b_tsls = $coeff(x)
    scalar se_tsls = $stderr(x)
    scalar t_tsls = (b_tsls-1)/se_tsls
    scalar r_tsls = abs(t_tsls)>critical(t,$df,.025)
    scalar a = Hausman(y, xvars, zvars)
    store coef.gdt b_ols se_ols r_ols b_tsls se_tsls r_tsls a f
    print b_ols se_ols r_ols b_tsls se_tsls r_tsls a f
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 $\rho$ and $\pi$ 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:

```
rho=0 (OLS efficient), theta=.5 (instruments strong)

Statistics for 10000 repetitions

                       mean         std. dev
   b_ols           0.999341        0.0789376
  se_ols          0.0784693       0.00766539
   r_ols          0.0540000         0.226018
  b_tsls           0.997716         0.125607
 se_tsls           0.124852        0.0184785
  r_tsls          0.0486000         0.215030
       a          0.0538000         0.225623
       f            23.1022          6.53108
```

With strong instruments, TSLS is basically unbiased. Least squares is unbiased and much more efficient that TSLS. The size of the Hausman test (Ho is true) is 0.0538, very close to the nominal level. Notice that the average value of the weak instrument test is 23.1, indicating the strong instruments. Try changing p and rho to replicate the findings in Table 10B.1 of *POE5*.

## 10.5 Script

```
set echo off
open "@workdir\data\mroz.gdt"
logs wage
square exper

# least squares and IV estimation of wage eq
# Example 10.1
m1 <- ols l_wage const educ exper sq_exper

# Example 10.2
smpl wage>0 --restrict
list x = const educ
list z = const mothereduc
m2 <- ols l_wage x
```

```
scalar se_educ_ols = $stderr(educ)
m3 <- tsls l_wage x; z
scalar se_educ_iv = $stderr(educ)

scalar a=corr(educ, mothereduc)
scalar ratio = se_educ_iv/se_educ_ols
scalar approx = 1/a
printf "\nThe correlation between mothers education\
and daughter is %.3f\n", a
printf "\nThe ratio of IV/OLS standard error for b2 is %.3f\n",\
        ratio
printf "\nReciprocal of the correlation is %.3f\n", approx

# Example 10.3
# tsls--manually
smpl wage>0 --restrict
m4 <- ols educ const mothereduc
series educ_hat = $yhat
m5 <- ols l_wage const educ_hat

# Example 10.4
# Simple regression, two instruments
list instruments = const mothereduc fathereduc
m6 <- ols educ instruments
series educ_hat = $yhat
m7 <- ols l_wage const educ
m8 <- ols l_wage const educ_hat
m9 <- tsls l_wage const educ; instruments

# Example 10.5
list xlist = const exper sq_exper educ
list instruments = const mothereduc fathereduc exper sq_exper
m10 <- tsls l_wage xlist ; instruments

fs10 <- ols educ instruments                # First stage regression
omit mothereduc fathereduc --test-only
# Example 10.6
# partial correlations--the FWL result
ols educ const exper sq_exper
series reduc = $uhat
ols mothereduc const exper sq_exper
series rmom = $uhat
ols reduc rmom --quiet
scalar c1 = corr(reduc, rmom)
printf "  Partial R2 = %.4f\n\
 The correlation between reduc and rmom = %.4f\n\
 The correlation squared is %.4f\n", $rsq, c1,c1^2

ols fathereduc const exper sq_exper
series rdad = $uhat
ols reduc rmom rdad
```

```
printf "Partial R2 = %.3f\n", $rsq

# Example 10.7
list xlist = const exper sq_exper educ
list instruments = const exper sq_exper mothereduc fathereduc

# Hausman test
ols educ instruments --quiet
series ehat = $uhat
h1 <- ols l_wage xlist ehat
omit ehat --test-only

# Sargan test
tsls l_wage xlist; instruments
series uhat = $uhat
ols uhat instruments
scalar test = $rsq*$nobs
pvalue X 2 test

# Cragg-Donald F
open "@workdir\data\mroz.gdt"
smpl wage>0 --restrict
logs wage
square exper
series nwifeinc = (faminc-wage*hours)/1000
list x1 = mtr educ kidsl6 nwifeinc const
list z1 = kidsl6 nwifeinc educ exper const

# Weak IV Example 1
iv1 <- tsls hours x1 ; z1
rf1 <- ols mtr z1

# Weak IV Example 2
list z2 = z1 sq_exper largecity
iv2 <- tsls hours x1 ; z2
rf2 <- ols mtr z2
omit exper sq_exper largecity --test-only

# Weak IV Example 3
list z3 = kidsl6 nwifeinc mothereduc fathereduc const
iv3 <- tsls hours x1 ; z3
rf3_mtr <- ols mtr z3
omit mothereduc fathereduc --test-only
rf3_educ <- ols educ z3
omit mothereduc fathereduc --test-only

# Weak IV Example 4
list z4 = z3 exper
iv4 <- tsls hours x1 ; z4
rf4_mtr <- ols mtr z4
omit exper mothereduc fathereduc --test-only
```

```
rf4_educ <- ols educ z4
omit exper mothereduc fathereduc --test-only

tsls hours x1 ; z3
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

# Example 10.8
# canonical correlations in gretl--Weak IV example 3
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

# Example 10.9
open "@workdir\data\ch10.gdt"
ols_1 <- ols y const x
iv_1 <- tsls y const x ; const z1
iv_2 <- tsls y const x ; const z2
iv_3 <- tsls y const x ; const z3
iv_4 <-  tsls y const x ; const z1 z2

rf <- ols x const z1 z2
iv_5 <- tsls y const x ; const z1 z2 z3

# Sampling Properties of 2sls
# Function returns a 1 if reject Hausman null
function scalar Hausman (series y, list xvars, list zvars)
```

```
    list endogvars = xvars - zvars
    ols endogvars zvars --quiet
    series vhat = $uhat
    ols y xvars vhat --quiet
    scalar t = $coeff(vhat)/$stderr(vhat)
    scalar reject = abs(t)>1.96
    return reject
end function

# Simulation
scalar N = 100
nulldata N --preserve
scalar rho = 0.0  # 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 zvars = const 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
    list xvars = const x
    ols x const zvars --quiet
    scalar f = $Fstat
    ols y xvars --quiet
    scalar b_ols = $coeff(x)
    scalar se_ols = $stderr(x)
    scalar t_ols = (b_ols-1)/se_ols
    scalar r_ols = abs(t_ols)>critical(t,$df,.025)
    tsls y xvars; zvars --quiet
    scalar b_tsls = $coeff(x)
    scalar se_tsls = $stderr(x)
    scalar t_tsls = (b_tsls-1)/se_tsls
    scalar r_tsls = abs(t_tsls)>critical(t,$df,.025)
    scalar a = Hausman(y, xvars, zvars)
    store coef.gdt b_ols se_ols r_ols b_tsls se_tsls r_tsls a f
    print b_ols se_ols r_ols b_tsls se_tsls r_tsls a f
endloop
```

# Chapter 11

# Simultaneous Equations Models

In Chapter 11 of *POE5* 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 "@workdir\data\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 at 5%. The overall $F$-statistics are 19.97 and 69.19, both significant at 5% as well.

$$\widehat{\mathrm{q}} = \underset{(3.243)}{7.895} + \underset{(0.1425)}{0.6564}\,\mathrm{ps} + \underset{(0.7005)}{2.167}\,\mathrm{di} - \underset{(0.1213)}{0.5070}\,\mathrm{pf}$$

$$n = 30 \quad \bar{R}^2 = 0.6625 \quad F(3,26) = 19.973 \quad \hat{\sigma} = 2.6801$$

(standard errors in parentheses)

$$\widehat{\mathrm{p}} = \underset{(7.984)}{-32.51} + \underset{(0.3509)}{1.708}\,\mathrm{ps} + \underset{(1.724)}{7.602}\,\mathrm{di} + \underset{(0.2985)}{1.354}\,\mathrm{pf}$$

$$n = 30 \quad \bar{R}^2 = 0.8758 \quad F(3,26) = 69.189 \quad \hat{\sigma} = 6.5975$$

(standard errors in parentheses)

Table 11.1: The least squares estimates of the reduced form equations.

## 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 "@workdir\data\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

Demand: TSLS, using observations 1–30
Dependent variable: q
Instrumented: p
Instruments: const ps di pf

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | −4.2795 | 5.5439 | −0.7719 | 0.4471 |
| p | −0.37446 | 0.16475 | −2.273 | 0.0315 |
| ps | 1.2960 | 0.35519 | 3.649 | 0.0012 |
| di | 5.0140 | 2.2836 | 2.196 | 0.0372 |

| | | | |
|---|---|---|---|
| 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 |

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 $F(1, 26) = 20.572$

Table 11.2: Two-stage least square estimates of the demand of truffles.

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 ($p$-

Supply: TSLS, using observations 1–30
Dependent variable: q
Instrumented: p
Instruments: const ps di pf

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 20.03 | 1.223 | 16.38 | 0.0000 |
| p | 0.3380 | 0.02492 | 13.56 | 0.0000 |
| pf | −1.001 | 0.08253 | −12.13 | 0.0000 |

| | | | |
|---|---|---|---|
| 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: $\chi^2(1) = 2.62751\text{e-}007$
  with p-value = 0.999591

Sargan over-identification test –
  Null hypothesis: all instruments are valid
  Test statistic: LM = 1.53325
  with p-value = $P(\chi^2(1) > 1.53325) = 0.215625$

Weak instrument test –
  First-stage $F(2, 26) = 41.49$

Table 11.3: Two-stage least square estimates of the demand of truffles.

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:

```
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 *POE5* for a nice explanation for this.

## 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 "@workdir\data\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 $\ln(Q)$ and $\ln(price)$ appear below:

RF_Qty: OLS, using observations 1–111
Dependent variable: lquan

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 8.810 | 0.1470 | 59.92 | 0.0000 |
| stormy | −0.3878 | 0.1437 | −2.698 | 0.0081 |
| mon | 0.1010 | 0.2065 | 0.4891 | 0.6258 |
| tue | −0.4847 | 0.2011 | −2.410 | 0.0177 |
| wed | −0.5531 | 0.2058 | −2.688 | 0.0084 |
| thu | 0.05369 | 0.2010 | 0.2671 | 0.7899 |

| | | | |
|---|---|---|---|
| $R^2$ | 0.193372 | Adjusted $R^2$ | 0.154961 |
| $F(5, 105)$ | 5.034295 | P-value($F$) | 0.000356 |

RF_Price: OLS, using observations 1–111

Dependent variable: lprice

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | −0.2717 | 0.076 | −3.557 | 0.001 |
| stormy | 0.3464 | 0.075 | 4.639 | 0.000 |
| mon | −0.1129 | 0.107 | −1.052 | 0.295 |
| tue | −0.0411 | 0.105 | −0.394 | 0.695 |
| wed | −0.0118 | 0.107 | −0.111 | 0.912 |
| thu | 0.0496 | 0.104 | 0.475 | 0.636 |

| | | | |
|---|---|---|---|
| Mean dependent var | −0.193681 | S.D. dependent var | 0.381935 |
| Sum squared resid | 13.17566 | S.E. of regression | 0.354235 |
| $R^2$ | 0.178889 | Adjusted $R^2$ | 0.139789 |
| $F(5, 105)$ | 4.575106 | P-value($F$) | 0.000816 |
| Log-likelihood | −39.22286 | Akaike criterion | 90.44572 |
| Schwarz criterion | 106.7029 | Hannan–Quinn | 97.04078 |

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.

The 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.

```
Test on Model 5:

  Null hypothesis: the regression parameters are zero for the variables
    mon, tue, wed, thu
  Test statistic: F(4, 105) = 0.618762, p-value 0.650111
```

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 lquan const lprice days ; zvars
```

to produce the result:

Demand: TSLS, using observations 1–111
Dependent variable: lquan
Instrumented: lprice
Instruments: const stormy mon tue wed thu

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 8.5059 | 0.166 | 51.189 | 0.000 |
| lprice | $-1.1194$ | 0.429 | $-2.612$ | 0.010 |
| mon | $-0.0254$ | 0.215 | $-0.118$ | 0.906 |
| tue | $-0.5308$ | 0.208 | $-2.552$ | 0.012 |
| wed | $-0.5664$ | 0.213 | $-2.662$ | 0.009 |
| thu | 0.1093 | 0.209 | 0.523 | 0.602 |

| | | | |
|---|---|---|---|
| Mean dependent var | 8.523430 | S.D. dependent var | 0.741672 |
| Sum squared resid | 52.09032 | S.E. of regression | 0.704342 |
| $R^2$ | 0.196499 | Adjusted $R^2$ | 0.158237 |
| $F(5, 105)$ | 4.717062 | P-value($F$) | 0.000631 |
| Log-likelihood | $-457.7821$ | Akaike criterion | 927.5643 |
| Schwarz criterion | 943.8214 | Hannan–Quinn | 934.1593 |

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: $\chi^2(1) = 2.4261$
  with p-value = 0.119329

Weak instrument test –
  First-stage $F(1, 105) = 21.517$

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 Systems of Equations

### Example 11.3 in *POE5*

In this example a system of macroeconomic equations of the U.S. economy is proposed and analyzed. This is one of the most widely used examples of systems estimation and the data and code for it are included with **gretl**. I present this here as an example of **gretl**'s `system` command.

```
system

Variants:   system method=estimator
            sysname <- system
Examples:   "Klein Model 1" <- system
```

```
system method=tsls
system method=liml
```

Either of two forms of the command may be given, depending on whether you wish to save the system for estimation in more than one way or just estimate the system once.

To save the system you should assign it a name using the assignment operator, as in the first example (if the name contains spaces it must be surrounded by double quotes). In this case you estimate the system using the `estimate` command. With a saved system of equations, you are able to impose restrictions (including cross-equation restrictions) using the `restrict` command.

Alternatively you can specify an estimator for the system using method= followed by a string identifying one of the supported estimators: `ols` (Ordinary Least Squares), `tsls` (Two-Stage Least Squares) `sur` (Seemingly Unrelated Regressions), `3sls` (Three-Stage Least Squares), `fiml` (Full Information Maximum Likelihood) or `liml` (Limited Information Maximum Likelihood). In this case the system is estimated once its definition is complete.

An equation system is terminated by the line `end system`. Within the system you can define an equation, a list of instruments, a list of endogenous variable or an identity. In the Klein I example we define equations to be estimated and identities that help to identify the equations in the system. Also, the model is assigned a name and therefore we use estimate commands to run the regressions. The code is:

```
open klein.gdt

series W = Wp + Wg
series A = t + (1918 - 1931)

# set the model up as a system
"Klein Model 1" <- system
equation C 0 P P(-1) W
equation I 0 P P(-1) K(-1)
equation Wp 0 X X(-1) A
identity P = X - T - Wp
identity W = Wp + Wg
identity X = C + I + G
identity K = K(-1) + I
endog C I Wp P W X K
end system

# and estimate it in various ways
estimate "Klein Model 1" method=ols
estimate "Klein Model 1" method=tsls
estimate "Klein Model 1" method=liml
```

Two series are created. Total wage, `W`, is composed of private wages + government wages. A time trend, A, is created to match the year of observation.

The system is assigned the name "Klein Model 1". This is followed by three structural equations (consumption, investments, and private wages) and a set of four identities. Additionally, the endogenous variable names are listed. There are seven endogenous variables and seven equations. Only three of those equations contain parameters and errors (i.e., they are structural).

Once the system has been defined, estimation can be carried out in a number of ways. Here we `estimate` the named model using `method= ols`, `tsls`, or `liml`. The LIML estimator is discussed in the next section.

```
# and estimate it in various ways
estimate "Klein Model 1" method=ols
estimate "Klein Model 1" method=tsls
estimate "Klein Model 1" method=liml
```

The results from **gretl** reveal:

Equation system, Klein Model 1
Estimator: Limited Information Maximum Likelihood

Equation 1: LIML, using observations 1921–1941 ($T = 21$)
Dependent variable: C

| | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | 17.1477 | 1.840 | 9.318 | 0.000 |
| P | −0.2225 | 0.202 | −1.103 | 0.270 |
| P_1 | 0.3960 | 0.174 | 2.281 | 0.023 |
| W | 0.8226 | 0.055 | 14.853 | 0.000 |

| | | | |
|---|---|---|---|
| Mean dependent var | 53.99524 | S.D. dependent var | 6.860866 |
| Sum squared resid | 40.88419 | S.E. of regression | 1.395301 |
| Log-likelihood | −132.4186 | Smallest eigenvalue | 1.498746 |

LR over-identification test: $\chi^2(4) = 8.4972$ [0.0750]

Equation 2: LIML, using observations 1921–1941 ($T = 21$)
Dependent variable: I

|  | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | 22.5908 | 8.546 | 2.643 | 0.008 |
| P | 0.0752 | 0.202 | 0.372 | 0.710 |
| P_1 | 0.6804 | 0.188 | 3.616 | 0.000 |
| K1 | −0.1683 | 0.041 | −4.124 | 0.000 |

|  |  |  |  |
|---|---|---|---|
| Mean dependent var | 1.266667 | S.D. dependent var | 3.551948 |
| Sum squared resid | 34.99649 | S.E. of regression | 1.290930 |
| Log-likelihood | −121.0536 | Smallest eigenvalue | 1.085953 |

LR over-identification test: $\chi^2(4) = 1.73161$ [0.7850]

Equation 3: LIML, using observations 1921–1941 ($T = 21$)
Dependent variable: Wp

|  | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | 1.5262 | 1.188 | 1.284 | 0.199 |
| X | 0.4339 | 0.068 | 6.387 | 0.000 |
| X_1 | 0.1513 | 0.067 | 2.257 | 0.024 |
| A | 0.1316 | 0.032 | 4.063 | 0.000 |

|  |  |  |  |
|---|---|---|---|
| Mean dependent var | 36.36190 | S.D. dependent var | 6.304401 |
| Sum squared resid | 10.02192 | S.E. of regression | 0.690821 |
| Log-likelihood | −136.8911 | Smallest eigenvalue | 2.468583 |

LR over-identification test: $\chi^2(4) = 18.9765$ [0.0008]

Cross-equation VCV for residuals
(correlations above the diagonal)

|  |  |  |
|---|---|---|
| 1.9469 | (0.555) | (−0.384) |
| 1.0006 | 1.6665 | (0.256) |
| −0.36970 | 0.22834 | 0.47723 |

log determinant = −0.557984

Breusch–Pagan test for diagonal covariance matrix:
$\chi^2(3) = 10.946$ [0.0120]

If you are following along in *POE5*, note that $E_t$ in *POE5* is labeled `X` by **gretl** .

## 11.6 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 *POE5*. 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 *POE5*.

```
open "@workdir\data\mroz.gdt"
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 Model_1 <- tsls hours x; z1 --liml
12 Model_2 <- tsls hours x; z2 --liml
13 Model_3 <- tsls hours x; z3 --liml
14 Model_4 <- tsls hours x; z4 --liml
15 Model_4_tsls <- tsls hours x; z4
```

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 estimates
Dependent variable: hours

| | Model_1 | Model_2 | Model_3 | Model_4 | Model_4_tsls |
|---|---|---|---|---|---|
| const | 17423.721** | 16191.333** | −24491.599 | 18587.906** | 18067.842** |
| | (3117.827) | (2979.234) | (79222.875) | (3662.026) | (3534.909) |
| mtr | −18456.589** | −17023.816** | 29709.468 | −19196.517** | −18633.922** |
| | (3615.228) | (3454.423) | (89957.674) | (3980.227) | (3843.850) |
| educ | −145.293** | −134.550** | 258.559 | −197.259** | −189.861** |
| | (32.810) | (31.407) | (841.058) | (64.243) | (62.355) |
| kidsl6 | 151.023 | 113.503 | −1144.478 | 207.553 | 190.275 |
| | (140.188) | (134.363) | (2495.489) | (162.296) | (158.305) |
| nwifeinc | −103.898** | −96.289** | 149.232 | −104.942** | −102.152** |
| | (19.602) | (18.735) | (467.761) | (20.565) | (19.899) |
| $n$ | 428 | 428 | 428 | 428 | 428 |
| $R^2$ | | | | | 0.199 |

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

The LIML results are easy to replicate using matrix commands. Doing so reveals some of

**hansl**'s power. 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 proposed function computes LIML, its standard errors and t-ratios. It takes four arguments. The first is a series for the dependent variable, next is a list of variables for the regression, next is a complete set of instruments that includes all exogenous variables in x, and finally a string. If the value of the string is "Fuller" then Fuller's modification is used for LIML estimation. If it is something else, it uses the regular LIML routing. For non Fuller modified problems I suggest using "no Fuller" since the value of this string makes its way to the screen as part of the output.

```
function void LIML (series depvar "dependent variable",
      list xvars "regressor list",
      list zvars "instrument list",
      string a   "Fuller or No Fuller")
    list endogvars = xvars - zvars
    list yvars = depvar endogvars
    matrix Y = { yvars }                   # All Endogenous vars, y and Y
    matrix y = { depvar }
    matrix w = { zvars }                   # w=All instruments
    matrix z = { xvars - endogvars }       # z=Internal instruments only
    matrix X = { xvars }

    matrix Mz = I($nobs)-z*invpd(z'*z)*z'  # Projection off of Z
    matrix Mw = I($nobs)-w*invpd(w'*w)*w'  # Projection off of w
    matrix Ez = Mz*Y                       # Residuals
    matrix Ew = Mw*Y                       # Residuals
    matrix W0 = Ez'*Ez                     # SSE
    matrix W1 = Ew'*Ew                     # SSE
    matrix G = inv(W1)*W0
    matrix l = eigengen(G, null)

    if a == "Fuller"
        scalar k=min(l)-(1/($nobs-nelem(xvars)))
    else
        scalar k=min(l)
    endif

    matrix kM = (I($nobs)-(k*Mw))
    matrix b =invpd(X'*kM*X)*X'*kM*y
    matrix sig2=(y-X*b)'*(y-X*b)/($nobs-nelem(xvars))
    matrix covmat = sig2*invpd(X'*kM*X)
    matrix se = sqrt(diag(covmat))
    matrix results = b~se~b./se

    cnameset(results, "Coeff Std_Error t-ratio")
    rnameset(results, "mtr educ kidsl6 nwifeinc const ")
    printf "\nThe LIML estimates using %s adjustment with k=%3f \n %12.3f\n", a, k, re
end function
```

It is called using:

```
LIML(hours, x, z1, "no Fuller")
LIML(hours, x, z2, "no Fuller")
LIML(hours, x, z3, "no Fuller")
LIML(hours, x, z4, "no Fuller")
```

Which produces the same results as did the `tsls` command using the `--liml` flag. An example of the output is shown below:

```
LIML(hours, x, z3, "no Fuller")

The LIML estimates using no Fuller adjustment with k=1.000000
                 Coeff   Std_Error     t-ratio
     mtr    29709.468   90487.777       0.328
    educ      258.559     846.014       0.306
  kidsl6    -1144.478    2510.194      -0.456
nwifeinc      149.232     470.517       0.317
   const   -24491.599   79689.720      -0.307
```

*POE5* also produces results for a Fuller-Modified LIML estimator. This requires that the string in our `LIML` function be set to ”Fuller”.

The result from the script for `LIML(hours, x, z3, "Fuller")` is:

```
The LIML estimates using Fuller adjustment with k=0.997636
                 Coeff   Std_Error     t-ratio
     mtr    -1304.857   15767.674      -0.083
    educ      -29.605     151.111      -0.196
  kidsl6     -287.791     445.963      -0.645
nwifeinc      -12.011      82.098      -0.146
   const     2817.572   13887.860       0.203
```

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.

## 11.7 Script

```
set verbose off
open "@workdir\data\truffles.gdt"
# reduce form estimation
list zvars = const ps di pf
RF_Q <- ols q zvars
RF_P <- ols p zvars

# demand and supply of truffles
Demand <- tsls q const p ps di; zvars
Supply <- tsls q const p pf; zvars

# Hausman test
ols p zvars
series v = $uhat
ols q const p pf v
omit v

# supply estimation by OLS
Supply_ols <- ols q const p pf

# Fulton Fish
open "@workdir\data\fultonfish.gdt"
#Estimate the reduced form equations
list days = mon tue wed thu
list z = const stormy days
RF_Qty <- ols lquan z
RF_Price <- ols lprice z
omit days --quiet

Demand <- tsls lquan const lprice days  ; z

# Example Klein I
open klein.gdt

series W = Wp + Wg
series A = t + (1918 - 1931)

# set the model up as a system
"Klein Model 1" <- system
equation C 0 P P(-1) W
equation I 0 P P(-1) K(-1)
equation Wp 0 X X(-1) A
identity P = X - T - Wp
identity W = Wp + Wg
identity X = C + I + G
identity K = K(-1) + I
endog C I Wp P W X K
end system

# and estimate it in various ways
estimate "Klein Model 1" method=ols
estimate "Klein Model 1" method=tsls
estimate "Klein Model 1" method=liml

# LIML
open "@workdir\data\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 exper mothereduc fathereduc

# LIML using tsls
Model_1 <- tsls hours x; z1 --liml
Model_2 <- tsls hours x; z2 --liml
Model_3 <- tsls hours x; z3 --liml
Model_4 <- tsls hours x; z4 --liml
Model_4_tsls <- tsls hours x; z4

# Optional Fuller Modified LIML a=1
function void LIML (series depvar "dependent variable",
      list xvars "regressor list",
      list zvars "instrument list",
      string a   "Fuller or No Fuller")
    list endogvars = xvars - zvars
    list yvars = depvar endogvars
    matrix Y = { yvars }                   # All Endogenous vars, y and Y
    matrix y = { depvar }
    matrix w = { zvars }                   # w=All instruments
    matrix z = { xvars - endogvars }       # z=Internal instruments only
    matrix X = { xvars }

    matrix Mz = I($nobs)-z*invpd(z'*z)*z'  # Projection off of Z
    matrix Mw = I($nobs)-w*invpd(w'*w)*w'  # Projection off of w
    matrix Ez = Mz*Y                       # Residuals
    matrix Ew = Mw*Y                       # Residuals
    matrix W0 = Ez'*Ez                     # SSE
    matrix W1 = Ew'*Ew                     # SSE
    matrix G = inv(W1)*W0
    matrix l = eigengen(G, null)

    if a == "Fuller"
        scalar k=min(l)-(1/($nobs-nelem(xvars)))
    else
        scalar k=min(l)
    endif

    matrix kM = (I($nobs)-(k*Mw))
    matrix b =invpd(X'*kM*X)*X'*kM*y
    matrix sig2=(y-X*b)'*(y-X*b)/($nobs-nelem(xvars))
    matrix covmat = sig2*invpd(X'*kM*X)
    matrix se = sqrt(diag(covmat))
    matrix results = b~se~b./se

    cnameset(results, "Coeff Std_Error t-ratio")
    rnameset(results, "mtr educ kidsl6 nwifeinc const ")
    printf "\nThe LIML estimates using %s adjustment with k=%3f \n %12.3f\n", a, k, re
end function

# LIML and Fuller modified LIML using matrices
LIML(hours, x, z1, "no Fuller" )
LIML(hours, x, z2, "no Fuller" )
LIML(hours, x, z3, "no Fuller" )
LIML(hours, x, z4, "no Fuller" )
LIML(hours, x, z1, "Fuller" )
LIML(hours, x, z2, "Fuller" )
LIML(hours, x, z3, "Fuller" )
LIML(hours, x, z4, "Fuller" )
```

# Chapter 12

# Regression with Time-Series Data: Nonstationary Variables

The main purpose this chapter is to explore the time-series properties of your data using **gretl**. 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 depends 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 most time-series regressions the data must be **stationary** in order for estimators to have desirable properties. This requires that the means, variances and covariances of the data series be independent 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 should 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 look at the data graphically. A time-series plot will reveal potential problems with your data and suggest ways to proceed statistically. As seen in Chapter 9, time-series plots are simple to generate using built-in functions that performs this task. Open the data file *gdp5.gdt* and create the first differences of GDP using the `diff` command. The first differences of the 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 "@workdir\data\gdp5.gdt"
diff gdp
setinfo gdp -d "= real US gross domestic product" -n "Real GDP"
setinfo d_gdp -d "= first difference of GDP" -n "D.GDP"
```

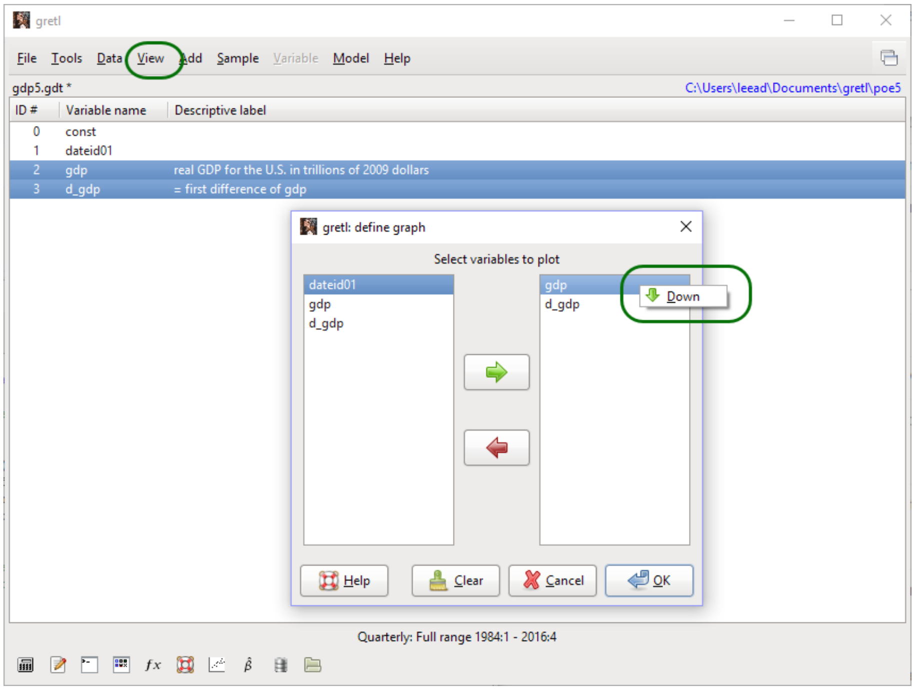
The `setinfo` command is used to add descriptions and labels for graphing. Recall, the `-d` switch changes the description and `-n` assigns a label to be used in graphs. Text needs to be enclosed in double quotes.

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 up to 16 variables at a time.

Use your mouse to select four of the series. I chose gdp and d_gdp. Once these are highlighted there are two ways to generate a simple graph. 1) right-click and choose **Time series plot** from the flyout menu. This opens a dialog box called **define graph** that allows you to choose whether to plot the series on a single graph or in separate small graphs. 2) Select **View>Multiple graphs>Time-series** from the pull-down menu. These variables should appear in the ‘Selected variables to plot’ box of a **define graph** dialog. 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_gdp below gdp in the list as shown in Figure 12.1.

When plotting two series, putting the series in the same graph can be useful. Below, the `plot` command is used to do this for the GDP and change in GDP series. In this example, separate scales are added to the right and left side $y$ axis since the scales of the two variables are so different. The left side scale is in $trillion and the right side is also in $trillion but varies on a much smaller scale.

```
string title = "U.S. GDP and Change in GDP"
string xname = "Year"
string yname = "GDP $Trillion"
string y2name = "Change in Quarterly GDP"
list plotmat =  gdp d_gdp
g1 <- plot plotmat
    options time-series with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
    printf "set y2label \"%s\"", y2name
end plot --output=display
```



gdp5.gdt *

C:\Users\leead\Documents\gretl\poe5

| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | dateid01 | |
| 2 | gdp | real GDP for the U.S. in trillions of 2009 dollars |
| 3 | d_gdp | = first difference of gdp |

Select variables to plot

| | |
|---|---|
| dateid01 | gdp |
| gdp | d_gdp |
| d_gdp | |

Quarterly: Full range 1984:1 - 2016:4

Figure 12.1: Choose **View>Multiple graphs>Time series** from the pull-down menu. Then select the desired variables to plot from the available list on the left-hand side by highlighting the variable and clicking on the green arrow. The ordering of the variables can be changed as desired by highlighting the variable in the right-side box, right-clicking and choosing up or down to move its position.

The output from this plot can be seen in Figure 12.2.

### Example 12.1 in *POE5* cont.

In this part of the example the inflation rate, the three year bond rate, and the fed funds rate are plotted. These series are recorded monthly and are found in the *usdata5.gdt* dataset. Load the data and add the differences of the three series to the dataset. If desired, change the series attributes using the `setinfo` commands with the `-d` and `-n` switches.

```
open "@workdir\data\usdata5.gdt"
diff br infn ffr                # take differences

# change series attributes
setinfo br -d "3-year Bond rate" -n "3-year Bond rate"
setinfo d_br -d "Change in the 3-year Bond rate" -n "D.bond rate"
setinfo infn -d "annual inflation rate" -n "inflation rate"
setinfo d_infn -d "Change in the annual inflation rate" -n "D.inflation"
setinfo ffr -d "federal funds rate" -n "Fed Funds Rate"
```

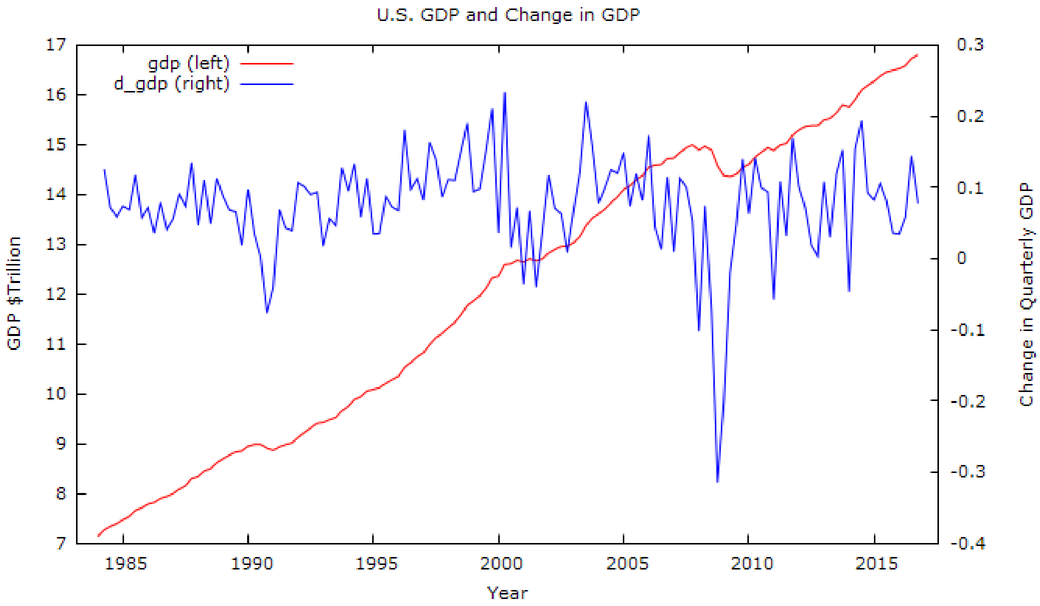


Figure 12.2: Two series are plotted in the same graph with different scales applied to left and right axes.

```
setinfo d_ffr -d "= first difference of f" -n "D.fed funds rate"
```

Finally, use the scatters command to plot each of the series and their differences.

```
g3 <- scatters infn d_infn br d_br ffr d_ffr --output=display
```

With a little editing in **gnuplot** this leads to Figure 12.3 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 12.4).

Comparing summary statistics of subsamples can be revealing. Stationarity implies that summary statistics (means, variances) should not change over time in stationary random variables. Below, the sample of GDP and of inflation and interest rates are split and simple summary statistics are generated. For quarterly GDP and its changes, the first subsample is from 1984:2-2000:3. The second subsample is 2000:4-2016:4.

```
open "@workdir\data\gdp5.gdt"
diff gdp
smpl 1984:2 2000:3
```

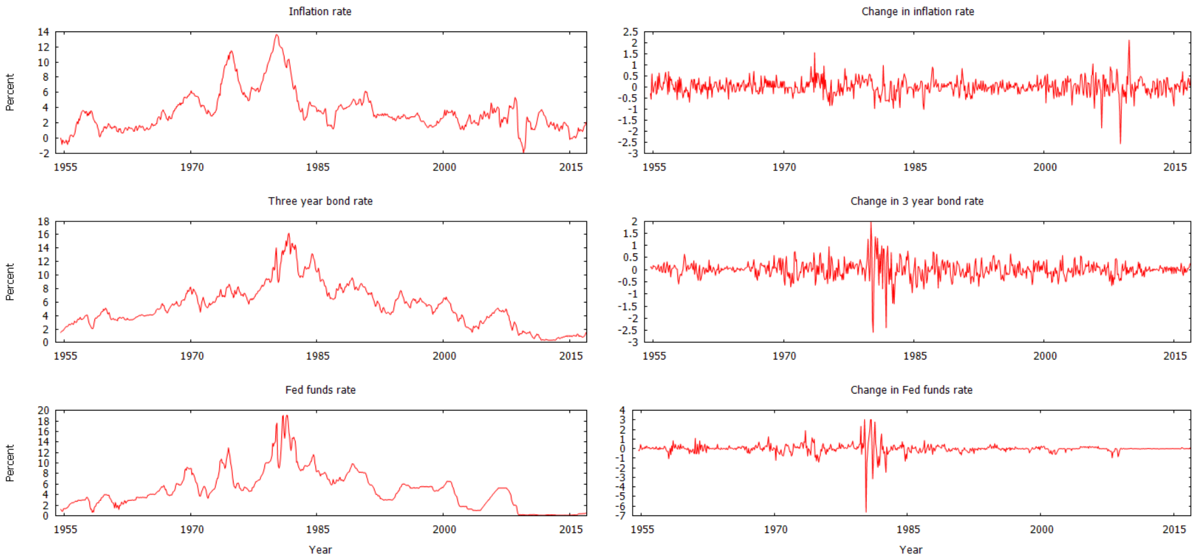


Figure 12.3: Plots of inflation, 3 year bond, and fed funds rates

```
summary gdp d_gdp --simple
smpl 2000:4 2016:4
summary gdp d_gdp --simple
smpl full
```

For the GDP series we get:

```
Full data range: 1984:1 - 2016:4 (n = 132)
Current sample: 1984:2 - 2000:3 (n = 66)

                  Mean      Median        S.D.         Min         Max
gdp              9.557       9.268       1.482       7.266       12.61
d_gdp          0.08283     0.08030     0.05287    -0.07650      0.2334

Current sample: 2000:4 - 2016:4 (n = 65)

                  Mean      Median        S.D.         Min         Max
gdp              14.68       14.75       1.159       12.64       16.80
d_gdp          0.06457     0.07780     0.08722     -0.3146      0.2203
```

The average GDP is increasing over time. In the latter period it is more than 3 standard deviations higher than in the early period. The averages of the differenced series do not appear to be changing significantly. This is consistent with stationarity in mean.

As discussed in Chapter 9, sample autocorrelations can reveal potential nonstationarity in a series. Nonstationary series tend to have large autocorrelations at long lags. This is evident for the

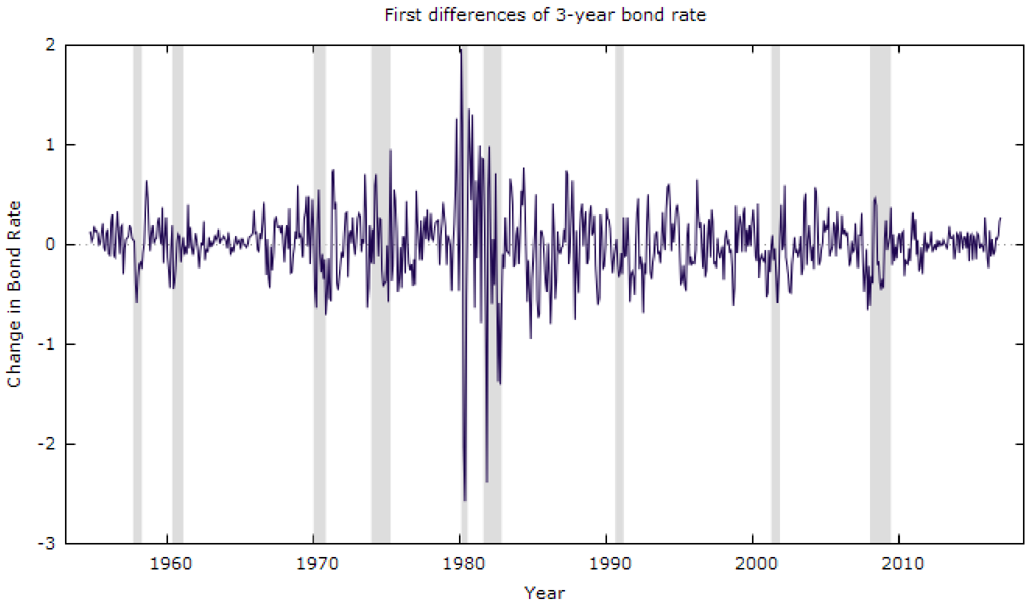


Figure 12.4: Individual plots can be edited using the **edit controls**. This one shows the first differences of the 3 year bond rate. Recessions are shaded grey.

GDP series as shown in Figure 12.5 The large autocorrelations for GDP persist beyond 24 lags, a clear sign that the series is nonstationary. Only the first two autocorrelations are significant for the changes series.

For the monthly inflation and interest rate series the subsamples are 1954:08-1985:10 and 1985:11-2016:12.

```
open "@workdir\data\usdata5.gdt"
diff br infn ffr                # take differences

list levels = infn ffr br       # Variable list for levels
list diffs = d_infn d_ffr d_br  # Variable list for differences
smpl 1954:8 1985:10
summary levels diffs --simple
smpl 1985:11 2016:12
summary levels diffs --simple
smpl full
```

```
Full data range: 1954:08 - 2016:12 (n = 749)
Current sample: 1954:08 - 1985:10 (n = 375)

                 Mean      Median        S.D.         Min         Max
infn            4.416       3.552       3.350     -0.8574       13.62
ffr             6.203       5.060       3.877      0.6300       19.10
```

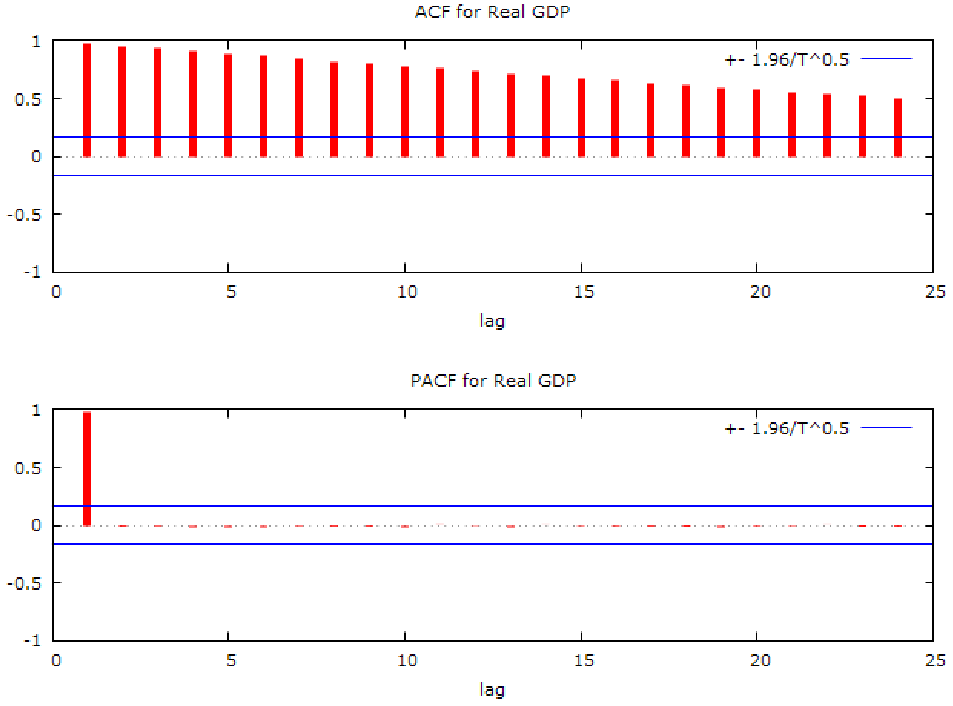


Figure 12.5: Autocorrelations and partial autocorrelations for GDP

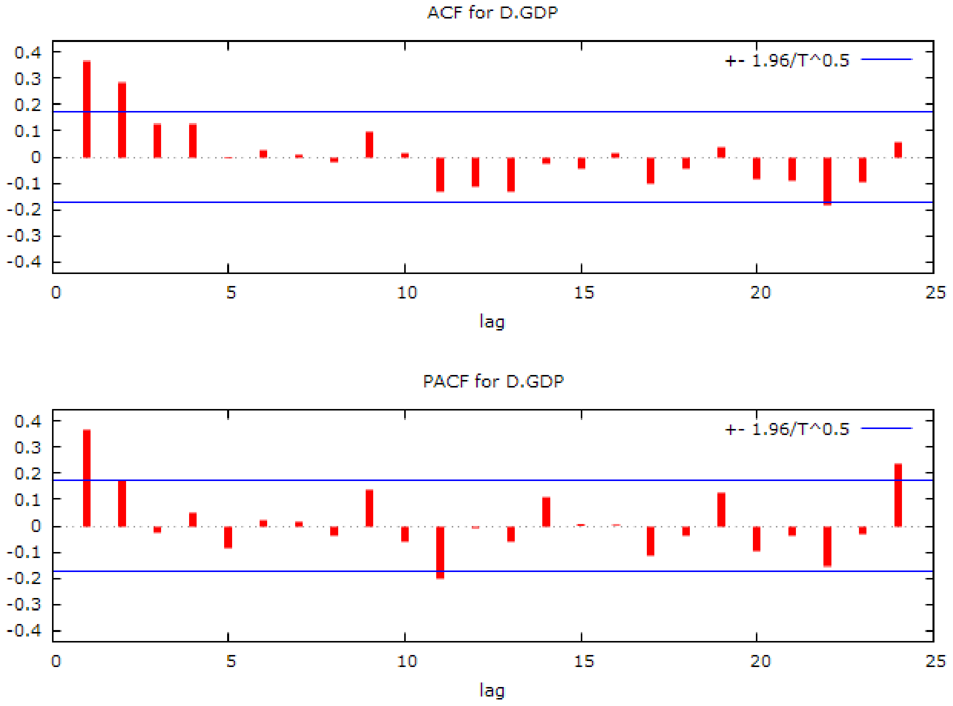


Figure 12.6: Autocorrelations and partial autocorrelations for changes in GDP.

```
br                6.562       5.880       3.236       1.490       16.22
d_infn         0.008513   -0.009750      0.3349     -0.9808       1.555
d_ffr           0.01810     0.03000      0.6927      -6.630       3.060
d_br            0.02075     0.05000      0.4230      -2.580       1.960

Current sample: 1985:11 - 2016:12 (n = 374)

                   Mean      Median        S.D.         Min         Max
infn              2.593       2.660       1.328      -1.978       6.184
ffr               3.650       3.990       2.788     0.07000       9.850
br                4.291       4.555       2.614      0.3300       9.610
d_infn        -0.002978   -0.006500      0.3850      -2.569       2.121
d_ffr          -0.01992      0.0000      0.1969     -0.9600      0.8700
d_br           -0.02075    -0.03000      0.2572     -0.8000      0.7400
```

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 are 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, issue a `smpl full` command after getting summary statistics for the first subset.

Another option that is useful in time-series data is `--contiguous`. The `--contiguous` form of `smpl` is intended for use with time-series data. It trims any observations at the start and end of the current sample range that contain missing values (either for the variables in `varlist`, or for all data series if no `varlist` is given). Then a check is performed to see if there are any missing values in the remaining range; if so, an error is flagged.

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 = infl ffr br
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 Deterministic Trends

Nonstationary variables that appear to wander up for a while and then down for a while are said to have **stochastic trends**. On the other hand, some trends are persistent and these are said to be **deterministic**. A time series may posses both types of trend. A simple deterministic trend for a series $y_t$ could be modeled:

$$y_t = c_1 + c_2 t + u_t$$

where $t$ is an index of time. A quadratic trend would be

$$y_t = c_1 + c_2 t + c_3 t^2 + u_t.$$

A trend in the percentage change could be modeled

$$\ln(y_t) = c_1 + c_2 t + u_t.$$

In each of these, the effect of time period is parameterized and can be estimated.

### Example 12.2 in *POE5*

In this example wheat production in Toodyay Shire Australia is studied. Wheat production depends on rainfall and productivity, which tends to improve over time. Thus, it is reasonable that yield might have a deterministic trend. Rainfall could also be changing over time as well, perhaps due to changes in global climate.

After loading the data, which are in *toody5.gdt*, add the natural logarithm of yield and the square of rainfall to the dataset.

```
open "@workdir\data\toody5.gdt"
logs y
square rain
```

Linear trends are estimated for both ln(*yield*) and *rainfall*. The series along with the estimated trends are plotted using the `plot` commands. First, wheat yield:

```
Model_1 <- ols l_y const year
series l_yhat = $yhat
list yield = l_y l_yhat year

# Graph of series against lags
string title = "Wheat Yield"
string xname = "Year"
string yname = "ln(Yield)"
g3 <- plot yield
    options --with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display
```

In line 1 the linear deterministic trend is estimated and the predictions saved as a series in line 2. The `plot` command (line 9) requires either a `matrix` or a `list` of variables to plot. A list is formed in line 3 and consists of the ln($yield$) and the predictions from the estimated model.

Titles and labels are created in lines 6-8 and the `plot` block begins in line 9. Line 10 is for the **gretl** option that replaces dots with lines in the plots. The next line is a **gnuplot** literal that alters the line types and colors. The plot appears in Figure 12.7. The fitted trend is positive.

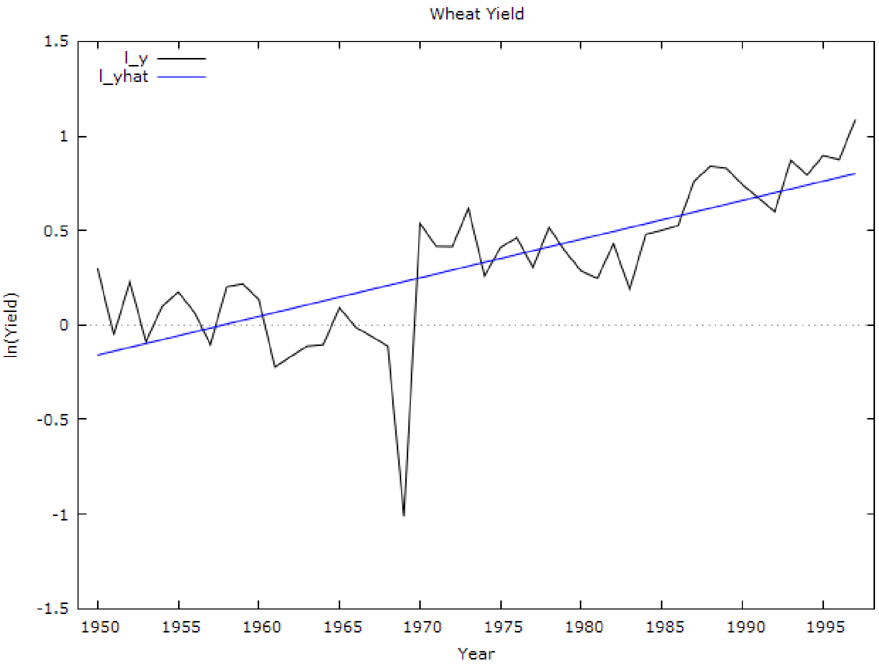


Figure 12.7: Plot of Wheat Yield for Toodyay Shire

The figure for rainfall was produced similarly:

```
Model_2 <- ols rain const year
series l_yhat = $yhat
list rainfall = rain l_yhat year

# Graph of series against lags
string title = "Rainfall during growing season 1950-1997"
string xname = "Year"
string yname = "Rain"
g4 <- plot rainfall
    options --with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
```

```
end plot --output=display
```

The plot appears in Figure 12.8. In this case there may be a slight negative trend, though the regression results do not suggest that the trend coefficient is significantly different from zero.

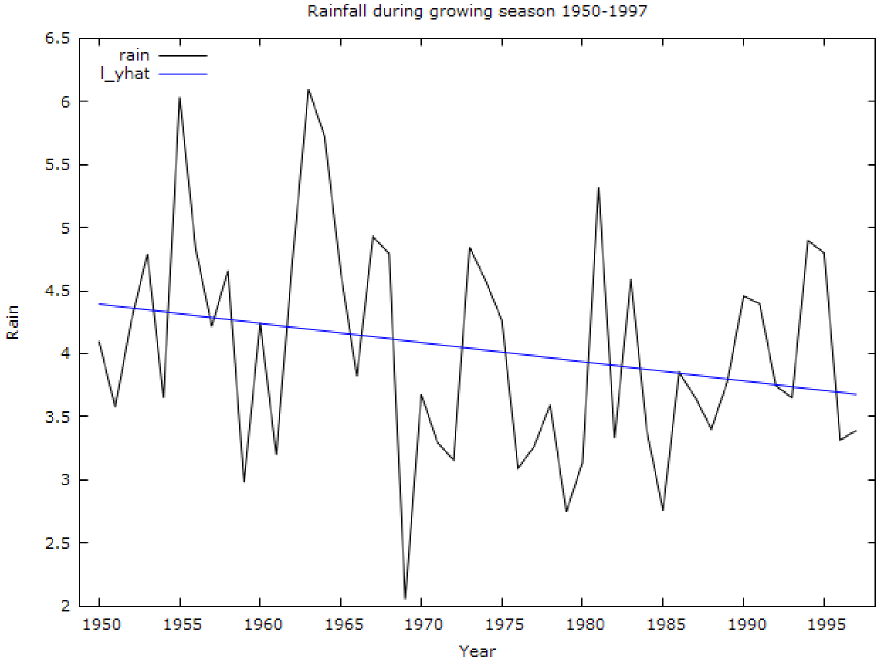


Figure 12.8: Plot of Rainfall for Toodyay Shire

A model of yield is estimated that includes a time-trend, rainfall and its square. The regression is estimated using:

```
list xvars = const year rain sq_rain
Trend <- ols l_y xvars
```

and yields

$$\widehat{\text{l\_y}} = \underset{(4.9754)}{-40.9210} + \underset{(0.0025194)}{0.0197080}\,\text{year} + \underset{(0.29036)}{1.14895}\,\text{rain} - \underset{(0.034610)}{0.134388}\,\text{sq\_rain}$$

$$T = 48 \quad \bar{R}^2 = 0.6408 \quad F(3,44) = 28.945 \quad \hat{\sigma} = 0.23264$$

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

Another approach that yields the same estimates is to detrend each of the series before estimating the regression. The results are the same as predicted by the FWL theorem already discussed on page (370).

Log yield, rainfall, and rainfall squared are detrended and the regression estimated:

```
1 # Detrend
2 ols l_y const t
3 series e_ly = $uhat
4 ols rain const t
5 series e_rain = $uhat
6 ols sq_rain const t
7 series e_rain2 = $uhat
8 Detrend <- ols e_ly e_rain e_rain2
9 scalar se_e_rain = sqrt(46/44)*$stderr(e_rain)
```

The detrended regression is

$$\widehat{\text{e\_ly}} = \underset{(0.28397)}{1.14895}\,\text{e\_rain} - \underset{(0.033850)}{0.134388}\,\text{e\_rain2}$$

$$T = 48 \quad \bar{R}^2 = 0.2473 \quad F(1,46) = 16.442 \quad \hat{\sigma} = 0.22753$$

(standard errors in parentheses)

Notice that the coefficients of rain and rain-squared are identical to those estimated in the deterministic trend model. The standard errors are slightly different due to how canned software counts degrees of freedom for the calculation of $\hat{\sigma}^2$. This is easily fixed (see line 9) as this output confirms:

```
    se_rain =  0.29035553

  se_e_rain =  0.29035553
```

## 12.3 Spurious Regressions

### Example 12.3 in *POE5*

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.

$$\begin{aligned} rw_1 &: y_t = y_{t-1} + v_{1t} \\ rw_2 &: x_t = x_{t-1} + v_{2t} \end{aligned} \tag{12.1}$$

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.

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 "@workdir\data\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 right-hand side box and click **OK**. The result appears in top part of Figure 12.9 (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 4.22. 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.

Finally, the residuals of the spurious regression are tested for 1st order autocorrelation using the *LM* test discussed in Chapter 9. The `modtest 1 --autocorr` command produces:

```
Breusch-Godfrey test for first-order autocorrelation
OLS, using observations 1-700
```

```
Dependent variable: uhat

             coefficient   std. error   t-ratio    p-value
  --------------------------------------------------------
  const       0.0657376    0.0968844      0.6785   0.4977
  rw2        0.00298902   0.00321967    0.9284   0.3535
  uhat_1      0.988357     0.00591374   167.1      0.0000  ***

  Unadjusted R-squared = 0.975654

Test statistic: LMF = 27932.065123,
with p-value = P(F(1,697) > 27932.1) = 0

Alternative statistic: TR^2 = 682.957879,
with p-value = P(Chi-square(1) > 682.958) = 1.52e-150

Ljung-Box Q' = 685.049,
with p-value = P(Chi-square(1) > 685.049) = 5.33e-151
```

The $LM$ test statistic is 682.95 and its $p$-value is well below the 5% threshold. The conclusions based on visual evidence are confirmed statistically. The errors are autocorrelated.

The **hansl** script to produce similar plots can be found in section 12.7 below.

## 12.4 Tests for Stationarity

The (augmented) Dickey-Fuller test can be used to test for the stationarity of data. The test is based on the following regression model The augmented version of the Dickey-Fuller test adds a number of lagged differences to the model. For the model with a constant and no deterministic trend this would be:

$$\Delta y_t = \alpha + \gamma y_{t-1} + \sum_{s=1}^{m} a_s \Delta y_{t-s} + v_t \tag{12.2}$$

To perform the 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. Plots are used to identify any deterministic trends in the series. If the trend in the series is quadratic then the differenced version of the series will have a linear trend in them.

In Figure 12.3 you can see that the fed funds rate appears to be trending downward and its difference appears to wander around a constant. 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 (red) Figure 12.2 may be slightly quadratic in time, although the financial crisis in 2007 may have broken the trend. The differenced version of the series that appears below it has a slight upward drift through 2006 and hence I would try an augmented Dickey-Fuller (ADF) test that includes a constant and a time trend. The time coefficient can be tested for significance and dropped if desired.

As you may have guessed, analyzing time series in this way is a bit like reading entrails and there is 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 number of lagged terms to include in the ADF regressions must also be determined. There are a number of ways to do this. In principle, the residuals from the ADF regression should be void of any autocorrelation. Include just enough lags of $\Delta y_{t-s}$ 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 the `--test-down=tstat` option is used. Others use a model selection rule for lag selection, as done in *POE5*. Gretl also reports the outcome of an autocorrelation test whenever the built-in ADF routines are used.

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.

One more thing should be said about the ADF test results. Gretl expresses the model in a slightly different way than *POE5*. The model is

$$(1-L)y_t = \beta_0 + (\alpha - 1)y_{t-1} + \alpha_1 \Delta y_{t-1} + e_t \tag{12.3}$$

The coefficient $\beta_0$ is included because the series may have a trend, $(\alpha - 1) = \gamma$ is the coefficient of interest in the Dickey-Fuller regression, and $\alpha_1$ 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, $(1-L)y_t = y_t - Ly_t = y_t - y_{t-1} = \Delta y_t$.

Conducting ADF tests in **gretl** is very simple. The `adf` command is quite thorough in its options. The syntax for the command is:

```
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[=criterion] (automatic lag order)
            --perron-qu (see below)
Examples:  adf 0 y
           adf 2 y --nc --c --ct
           adf 12 y --c --test-down
           See also jgm-1996.inp
```

The first argument is the number of augmented differences to include in the ADF regression. This is followed by a list (note more than one variable can be tested if put into a list.) Options allow choices of models with constant, constant and trend and other deterministic models. The `--difference` option will perform the test on differenced series. The `--test-down` option is useful in selecting a suitable number of lagged differences to include in the model. When using the `--test-down` the specified order of `adf` serves as the maximum lag to consider. See section 13.1.2 for a short discussion of this approach.

When testing down, **gretl** will select the lag length that minimizes the AIC criterion. The `--test-down=tstat` option follows the following 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 as we have done in the example by using the qualifiers `--test-down=AIC` or `--test-down=BIC`. Gretl also includes an estimate of the autocorrelation coefficient in the output. Thus, it serves as a final check of the adequacy of the ADF regression.

### Example 12.4 in *POE5*

In the example from *POE5*, the federal funds rate (`ffr`) and the 3-year bond rate (`br`) 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 deterministic 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.

The script to perform the ADF test is:

```
1 open "@workdir\data\usdata5.gdt" --preserve
2 diff br infn ffr            # take differences
3 list levels = ffr br
4
5 adf 19 levels --c --test-down=BIC
6 adf 2 levels --c --verbose
```

In line 5 the lag lengths for both series are found by testing down from a maximum of 19 lags using the BIC criterion. A constant is included in the regression.

The test results are quite informative. For the fed funds rate thirteen lagged values were selected (from a maximum of 19) to include in the model.[1] It reveals that the first set of statistics is for a test based on a regression with a constant. It provides you with the $t$-ratio for the ADF test and it's approximate $p$-value. It also reports a calculation of first-order residual autocorrelation, which should be small if you have chosen the correct number of lags in the ADF regression.

As mentioned in *POE5*, 13 lags is a awful lot for data of this frequency. For the bond rate, only 2 lags were selected.

```
Augmented Dickey-Fuller test for ffr
testing down from 19 lags, criterion BIC
sample size 735
unit-root null hypothesis: a = 1

  test with constant
  including 13 lags of (1-L)ffr
  model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0117889
  test statistic: tau_c(1) = -2.46008
  asymptotic p-value 0.1255
  1st-order autocorrelation coeff. for e: -0.004
  lagged differences: F(13, 720) = 19.453 [0.0000]


Augmented Dickey-Fuller test for br
testing down from 19 lags, criterion BIC
sample size 746
unit-root null hypothesis: a = 1

  test with constant
  including 2 lags of (1-L)br
  model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.00635098
  test statistic: tau_c(1) = -1.69527
  asymptotic p-value 0.4337
```

[1] 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 $k_{max} = int[4(T+1)/100]^{0.25}$.

```
    1st-order autocorrelation coeff. for e: 0.010
    lagged differences: F(2, 742) = 74.518 [0.0000]
```

Here, the test statistic for the stationarity of the fed funds rate is $-2.460$ which has a $p$-value of 0.1255. Nonstationarity of the fed funds rate can not be rejected in this case at the usual 5 or 10% levels of significance.

For the bond rate, the nonstationary null cannot be rejected either. The $t$-ratio is -1.695 and has a $p$-value of .4337.

To replicate the results in *POE5*, we limit the lags to 2 and repeat the exercise, this time with the `--verbose` option to show the regression results. This confirms that the `tau_c(1)` statistic reported in the test result is the same as the $t$-ratio on the lagged level of the variable being tested. Since testing down led to the ADF(2) model, I only show the result for the fed funds rate.

```
Augmented Dickey-Fuller test for ffr
including 2 lags of (1-L)ffr
sample size 746
unit-root null hypothesis: a = 1

  test with constant
  model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0117728
  test statistic: tau_c(1) = -2.47497
  asymptotic p-value 0.1216
  1st-order autocorrelation coeff. for e: -0.003
  lagged differences: F(2, 742) = 75.854 [0.0000]

Augmented Dickey-Fuller regression
OLS, using observations 1954:11-2016:12 (T = 746)
Dependent variable: d_ffr

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const       0.0580131    0.0290229     1.999    0.0460    **
  ffr_1      0.0117728    0.00475674   2.475    0.1216
  d_ffr_1     0.444301     0.0361201    12.30     8.87e-032 ***
  d_ffr_2    0.147091     0.0363172    4.050    5.66e-05  ***

  AIC: 976.543   BIC: 995.002   HQC: 983.658
```

You can see that the $t$-ratio is $-2.475$ and equal to `tau_c(1)` in the test result. The $p$-value is .12 and hence nonstationarity of `ffr` cannot be rejected at 5%.

#### Using the GUI

The GUI can be used as well. 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 to open the `adf` dialog box shown in Figure 12.10. Select the maximum lag to consider (if testing down) or set the desired number of augmented terms to include in the regression. Choose whether to include a constant, trend, trend-squared, seasonal indicators, etc.

We have chosen to start with a maximum lag of 19 and to allow **gretl** to test-down to the number of lags required. In testing down, one has a choice of criterion to use. In *POE5* the authors use SC, which is equivalent to the BIC in **gretl**.

Also, chose to suppress the regression results by unchecking **show regression results** box. To make the results a bit more transparent it is often a good idea to check the regression results that generate the test statistics. Finally, At the bottom of the dialog you one choose whether to use the level or the difference of the variable in the regressions. Choosing **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 in its levels.

```
Augmented Dickey-Fuller test for ffr
testing down from 18 lags, criterion BIC
sample size 735
unit-root null hypothesis: a = 1

  test with constant
  including 13 lags of (1-L)ffr
  model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0117889
  test statistic: tau_c(1) = -2.46008
  asymptotic p-value 0.1255
  1st-order autocorrelation coeff. for e: -0.004
  lagged differences: F(13, 720) = 19.453 [0.0000]
```

Testing down selected 13 lags, the value of $\tau = -2.46$ and it's asymptotic $p$-value is .125.

### 12.4.1 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 *POE5*, 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.

The `adf` command is used with the `--gls` option:

```
adf 2 levels --c --gls
```

For the levels of the fed funds rate:

```
Augmented Dickey-Fuller (GLS) test for ffr
including 2 lags of (1-L)ffr
sample size 746
unit-root null hypothesis: a = 1

  test with constant
  model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.00603286
  test statistic: tau = -1.77353
  asymptotic p-value 0.07239
  1st-order autocorrelation coeff. for e: -0.003
  lagged differences: F(2, 743) = 75.202 [0.0000]
```

The test statistic is $-1.7735$ and has a $p$-value of .0723, which is in the 10% rejection region for the test. At 10%, the series is stationary.

For the levels of the bond rate:

```
Augmented Dickey-Fuller (GLS) test for br
including 2 lags of (1-L)br
sample size 746
```

```
unit-root null hypothesis: a = 1

  test with constant
  model: (1-L)y = b0 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.00267845
  test statistic: tau = -1.10861
  asymptotic p-value 0.2435
  1st-order autocorrelation coeff. for e: 0.010
  lagged differences: F(2, 743) = 74.387 [0.0000]
```

The test statistic is $-1.10861$, which is not in the 10% rejection region for the test. The series is nonstationary.

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 (Cottrell and Lucchetti, 2018, p. 241) itself is very simple

$$\eta = \frac{\sum_{i=1}^{T} S_t^2}{T^2 \tilde{\sigma}^2} \tag{12.5}$$

where $S_t = \sum_{s=1}^{t} e_s$ 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 9.9.3 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 negative number for the bandwidth, **gretl** will compute an automatic bandwidth of $4(T/100)^{1/4}$.

```
kpss -1 levels
```

The KPSS statistics using automatic bandwidth selection results in:

```
T = 749
Lag truncation parameter = 6
Test statistic = 2.48632
```

```
                   10%      5%      1%
Critical values: 0.348   0.462   0.743
P-value < .01


KPSS test for br

T = 749
Lag truncation parameter = 6
Test statistic = 2.85568

                   10%      5%      1%
Critical values: 0.348   0.462   0.743
P-value < .01
```

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 6.

### Example 12.5 in *POE5*

Is GDP trend stationary? The data are from *gdp5.gdt* and we explore whether log-GDP is stationary around a linear trend.

```
open "@workdir\data\gdp5.gdt"
adf 5 gdp --ct --test-down --verbose
```

We test down from a maximum of 5 lags and print the regression results using the `--verbose` option to facilitate comparison with *POE5*.

The test results:

```
Augmented Dickey-Fuller test for gdp
testing down from 5 lags, criterion AIC
sample size 129
unit-root null hypothesis: a = 1

  with constant and trend
  including 2 lags of (1-L)gdp
  model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0330036
  test statistic: tau_ct(1) = -1.99928
  asymptotic p-value 0.6012
  1st-order autocorrelation coeff. for e: -0.002
  lagged differences: F(2, 124) = 13.639 [0.0000]
```

The model selection rule chose 2 lags (same as *POE5*) and the test statistic for the ADF test is $-1.999$. Its $p$-value is 0.6 and nonstationarity around the trend is not rejected at 5%.

The regression results:

```
Augmented Dickey-Fuller regression
OLS, using observations 1984:4-2016:4 (T = 129)
Dependent variable: d_gdp

             coefficient   std. error   t-ratio   p-value
  -------------------------------------------------------
  const       0.266100     0.114246      2.329    0.0215  **
  gdp_1      0.0330036    0.0165078    1.999    0.6012
  d_gdp_1     0.311502     0.0871190     3.576    0.0005  ***
  d_gdp_2     0.201903     0.0883952     2.284    0.0241  **
  time        0.00248604   0.00126186    1.970    0.0511  *

  AIC: -328.068   BIC: -313.769   HQC: -322.258
```

### Example 12.6 in *POE5*

Is wheat yield trend stationary? The data are from *toody.gdt* and we explore whether log-yield is stationary around a linear trend.

```
open "@workdir\data\toody5.gdt"
logs y
adf 5 l_y --ct --test-down --verbose
```

The test results (with regression output suppressed) are:

```
Augmented Dickey-Fuller test for l_y
testing down from 5 lags, criterion AIC
sample size 47
unit-root null hypothesis: a = 1

  with constant and trend
  including 0 lags of (1-L)l_y
  model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + e
  estimated value of (a - 1): -0.745285
  test statistic: tau_ct(1) = -5.23971
  p-value 0.0004713
  1st-order autocorrelation coeff. for e: -0.051
```

The model selection rule chose 0 lags (same as *POE5*) and the test statistic for the ADF test is $-1.5297$. Its $p$-value is $0.00047 < .05$ and nonstationarity around the trend is rejected at 5%. Notice that even with no lagged differences included in the model, the residual autocorrelation is very small ($-0.051$).

## 12.5 Integration and 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. In the next example we examine the stationarity of their differences using the ADF test.

### Example 12.7 in *POE5*

In example 12.4 we concluded that the two interest rate series *ffr* and *br* were nonstationary in levels. In this example, we consider whether taking the series differences leads to stationarity.

The script is

```
open "@workdir\data\usdata5.gdt"
list levels = ffr br
adf 5 levels --c --test-down=BIC --difference --verbose
```

Notice that the `--difference` option is added. The results for the fed funds rate are:

```
  test statistic: tau_c(1) = -17.7491
  asymptotic p-value 1.237e-042
  1st-order autocorrelation coeff. for e: -0.004

Augmented Dickey-Fuller regression
OLS, using observations 1954:11-2016:12 (T = 746)
Dependent variable: d_d_ffr

              coefficient    std. error    t-ratio     p-value
  -------------------------------------------------------------
  const       0.000202807   0.0170601      0.01189    0.9905
  d_ffr_1     0.714831      0.0402742     17.75       1.24e-042 ***
  d_d_ffr_1   0.156785      0.0362296      4.328      1.71e-05  ***
```

and for the 3-year bond:

```
    test statistic: tau_c(1) = -19.8239
    asymptotic p-value 4.526e-047
    1st-order autocorrelation coeff. for e: 0.009

  Augmented Dickey-Fuller regression
  OLS, using observations 1954:11-2016:12 (T = 746)
  Dependent variable: d_d_br

               coefficient    std. error    t-ratio     p-value
    ------------------------------------------------------------
    const      0.000249556   0.0117418     0.02125   0.9830
    d_br_1     0.810547      0.0408873    19.82      4.53e-047 ***
    d_d_br_1   0.234677      0.0356773     6.578     9.01e-011 ***
```

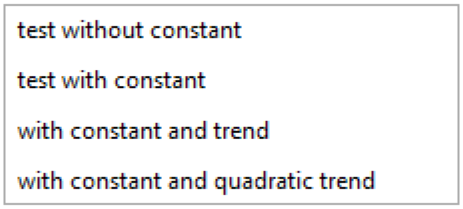
One lag was selected for both differenced series and both adf statistics are significant at 5%. The differences appear to be stationary.

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. Engle and Granger used simulations to determine the correct critical values for the test and the test is named for them.

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. (2018), 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, 2018, chapter 30).

Figure 12.11 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 indicate how many lags are wanted in the initial Dickey-Fuller regressions on each of the variables, which variables you want to include in the cointegrating relationship, and whether a constant, trend, or quadratic trend is required in the regressions. Testing down from the maximum lag order is chosen via a check-box. To select these additional modeling options click on the down arrow button indicated in Figure 12.11. This reveals the four choices:



The default, a model that contains a constant, is chosen. For the 3-year bond rate and the fed funds rate series we get the result shown in Figure 12.12.

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 `ffr` onto a constant and `br`. 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[=criterion] (automatic lag order)
            --verbose (print extra details of regressions)
            --silent (don't print anything)
Examples:coint 4 y x1 x2
            coint 0 y x1 x2 --ct --skip-df
```

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. This works just as it did with the `adf` command. By default 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 2 br ffr --skip-df
```

I chose to skip the Dickey-Fuller tests for stationarity of ffr and br since these have already been done and discussed above.

The cointegrating regression is:

```
Cointegrating regression -
OLS, using observations 1954:08-2016:12 (T = 749)
Dependent variable: br

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const       1.32769      0.0592746     22.40    2.08e-085 ***
  ffr         0.832029     0.00970676    85.72    0.0000    ***
```

The Engle-Granger test results, which looks very similar to the ADF output, are:

```
Step 2: testing for a unit root in uhat

Augmented Dickey-Fuller test for uhat
including 2 lags of (1-L)uhat
sample size 746
unit-root null hypothesis: a = 1

  model: (1-L)y = (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0817175
  test statistic: tau_c(2) = -5.52613
  asymptotic p-value 1.254e-005
  1st-order autocorrelation coeff. for e: 0.004
  lagged differences: F(2, 743) = 28.801 [0.0000]
```

The key difference is in how the $p$-value for the $\tau$ statistic is computed. The values are based on the distribution of the Engle-Granger $\tau$ rather than that of the ADF.

The asymptotic $p$-value is very small and easily less than conventional a 5% threshold. This suggests that the two series are indeed cointegrated.

## 12.6 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 *POE5* for details)

$$\Delta y_t = -(1 - \theta_1)(y_{t-1} - \beta_1 - \beta_2 x_{t-1}) + \delta_0 \Delta x_t + v_t \tag{12.8}$$

The term in the second set of parentheses is a cointegrating relationship where 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**. It is an empirical matter as to how many lags of $\Delta x_t$ and $\Delta y_t$ to add as regressors. In example 12.9 of *POE5* the authors add two lags of $\Delta br_t$ and four lags of $\Delta f\!fr_t$ to the model. Again, enough lags should be added to remove autocorrelation in the estimated residuals.

### Example 12.9 in *POE5*

The error correction model to be estimated is:

$$\begin{aligned}\Delta br_t = &-\alpha(br_{t-1} - \beta_1 - \beta_2 f\!fr_{t-1}) + \gamma_1 \Delta br_{t-1} + \gamma_2 \Delta br_{t-2}\\ &+ \delta_0 \Delta f\!fr_t + \delta_1 \Delta f\!fr_{t-1} + \delta_2 \Delta f\!fr_{t-2} + \delta_3 \Delta f\!fr_{t-3} + \delta_4 \Delta f\!fr_{t-4} + e_t\end{aligned}$$

Nonlinear least squares requires starting values. The cointegrating regression is used to initialize $\beta_1$ and $\beta_2$. Residuals are obtained and lagged to include in a linear regression to initialize the other parameters. The error correction parameter is initialized at zero. The initialization is thus:

```
open "@workdir\data\usdata5.gdt"
diff br ffr
ols br const ffr
series res = $uhat

ols d_br res(-1) d_br(-1 to -2) d_ffr(0 to -4)
scalar g1 = $coeff(d_br_1)
scalar g2 = $coeff(d_br_2)
scalar d0 = $coeff(d_ffr)
scalar d1 = $coeff(d_ffr_1)
scalar d2 = $coeff(d_ffr_2)
scalar d3 = $coeff(d_ffr_3)
scalar d4 = $coeff(d_ffr_4)

ols br const ffr
scalar b1 = $coeff(const)
scalar b2 = $coeff(ffr)
scalar a = 0
```

Once stating values are obtained, a `nls` block is constructed to estimate the model above.

```
nls d_br=-a*(br(-1)-b1-b2*ffr(-1))+ g1*d_br(-1) + g2*d_br(-2) + \
         d0*d_ffr + d1*d_ffr(-1) + d2*d_ffr(-2) + d3*d_ffr(-3) + \
         d4*d_ffr(-4)
    params a b1 b2 g1 g2 d0 d1 d2 d3 d4
end nls
```

Estimation yields:

```
Using numerical derivatives
Tolerance = 1.81899e-012
Convergence achieved after 34 iterations

Model 6: NLS, using observations 1955:01-2016:12 (T = 744)
d_br = -a*(br(-1)-b1-b2*ffr(-1))+ g1*d_br(-1) + g2*d_br(-2)
+ d0*d_ffr+d1*d_ffr(-1) + d2*d_ffr(-2) + d3*d_ffr(-3) + d4*d_ffr(-4)

             estimate    std. error   t-ratio    p-value
  --------------------------------------------------------
a           0.0463811   0.0118836     3.903    0.0001    ***
b1          1.32333     0.385968      3.429    0.0006    ***
b2          0.832977    0.0634870    13.12     1.76e-035 ***
g1          0.272365    0.0374505     7.273    9.06e-013 ***
g2         -0.242108    0.0378320    -6.400    2.78e-010 ***
d0          0.341783    0.0240428    14.22     1.09e-040 ***
d1         -0.105320    0.0275191    -3.827    0.0001    ***
d2          0.0990586   0.0273312     3.624    0.0003    ***
d3         -0.0659749   0.0244972    -2.693    0.0072    ***
d4          0.0560408   0.0228173     2.456    0.0143    **

Mean dependent var  -0.000430   S.D. dependent var   0.351277
Sum squared resid    58.72060   S.E. of regression   0.282844
R-squared            0.359525   Adjusted R-squared   0.351672
Log-likelihood      -111.0891   Akaike criterion     242.1783
Schwarz criterion    288.2987   Hannan-Quinn         259.9561

GNR: R-squared = 0, max |t| = 7.54859e-008
Convergence seems to be reasonably complete
```

These match the results in *POE5.* The cointegration parameter estimates are very close to the ones obtained by a simple regression of *br* onto *ffr* and a constant.

Finally, the estimated cointegration parameters $\beta_1$ and $\beta_2$ are used to compute residuals and these are tested for stationarity (a.k.a. Engle-Granger). The coint command cannot be used since it will not use the nonlinear least squares residuals for its computations. Instead, the adf command is pressed into service and the test statistic has to be compared to the proper critical value from Table 12.4 in *POE5*.

The script to compute $\hat{\theta}_1$ and the Engle-Granger statistic is:

```
scalar theta1 = 1-$coeff(a)
series ehat = br-$coeff(b1)-$coeff(b2)*ffr
adf 2 ehat --nc --verbose
```

Notice that the `--nc` constant switch is applied in order to suppress the constant in the `adf` regression. The regression results are:

```
Augmented Dickey-Fuller regression
OLS, using observations 1954:11-2016:12 (T = 746)
Dependent variable: d_ehat

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  ehat_1     0.0818818    0.0147997    5.533    5.34e-08  ***
  d_ehat_1    0.223567     0.0355366     6.291    5.38e-010 ***
  d_ehat_2   0.176794     0.0360954    4.898    1.19e-06  ***
```

The $t$-ratio on the lagged residual is -5.33. From Table 12.4 in *POE5* the 5% critical value is $-3.37$. Note, the cointegrating relationship contains an intercept. This determines which set of critical values to use from Table 12.4. The conclusion is that the bond rate and fed funds rate are cointegrated.

### Example 12.10

When two series are difference stationary (e.g., I(1)) but not cointegrated, then estimating the differenced equations separately is a useful strategy. In the next chapter, I consider how to estimate these as a system using vector autoregression.

Example 12.10 examines the stationarity of consumption and income using the *cons_inc.gdt* data used first in Example 9.16. These quarterly data begin in 1959:3.

```
open "@workdir\data\cons_inc.gdt"
diff cons y
```

The consumption and income series are plotted to identify trends.

```
g6 <- scatters cons y --with-lines
```

After some minor editing the plots are seen in Figure 12.13: Both series are trending upward. So, a trend will be included in the ADF regression, which begins with observations in 1985:1 and is indicated by the vertical line in the plots.

```
smpl 1985:1 2016:3
list vars = cons y
adf 1 vars --ct --verbose
```

The results are:

```
Augmented Dickey-Fuller test for cons
including one lag of (1-L)cons
sample size 127
unit-root null hypothesis: a = 1

  with constant and trend
  model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0192737
  test statistic: tau_ct(1) = -1.70241
  asymptotic p-value 0.7505
  1st-order autocorrelation coeff. for e: -0.024

Augmented Dickey-Fuller regression
OLS, using observations 1985:1-2016:3 (T = 127)
Dependent variable: d_cons

             coefficient    std. error    t-ratio   p-value
  ---------------------------------------------------------
  const      -1041.92        748.189       -1.393    0.1663
  cons_1        -0.0192737     0.0113214   -1.702    0.7505
  d_cons_1       0.244088      0.0864560    2.823    0.0055  ***
  time          29.4330       14.5330       2.025    0.0450  **

Augmented Dickey-Fuller test for y
including one lag of (1-L)y
sample size 127
unit-root null hypothesis: a = 1

  with constant and trend
  model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0409207
  test statistic: tau_ct(1) = -2.14283
  asymptotic p-value 0.5212
  1st-order autocorrelation coeff. for e: -0.020

Augmented Dickey-Fuller regression
OLS, using observations 1985:1-2016:3 (T = 127)
Dependent variable: d_y
```

```
             coefficient      std. error    t-ratio  p-value
  ------------------------------------------------------------
  const      -3199.99         1949.26        -1.642    0.1032
  y_1           -0.0409207       0.0190966   -2.143    0.5212
  d_y_1          0.248436        0.0859090    2.892    0.0045   ***
  time          80.0444         35.3259       2.266    0.0252   **
```

The ADF statistic is $-1.70$ in the consumption regression and the 5% critical value is 3.41, therefore we cannot reject nonstationarity for consumption. For income the ADF statistic is 2.143 which is also not significant. Both series are nonstationary in levels.

To determine whether the differenced series are stationary, repeat the ADF tests using the `--difference` option. Recall that differencing a series that includes a constant and a linear trend converts the trend to a constant.

```
adf 0 vars --verbose --difference
```

The $\tau$ statistics for `d_cons` and `d_y` are $-8.13995$ and $-8.6799$, respectively. Both are significant and nonstationarity is rejected at 5%.

Next, check for cointegration using the Engle-Granger test. Include a constant and a deterministic linear trend; also suppress the ADF test steps the nonstationarity of the levels has already been established.

```
coint 1 vars --ct --skip-df
```

The cointegrating regression is:

```
Cointegrating regression -
OLS, using observations 1985:1-2016:3 (T = 127)
Dependent variable: cons

               coefficient     std. error    t-ratio    p-value
  -------------------------------------------------------------
  const      -19166.5         2311.83       -8.291    1.57e-013 ***
  y               0.467725       0.022827   20.49     1.27e-041 ***
  time          420.439         42.3988       9.916    2.02e-017 ***
```

and the test outcome is:

```
Augmented Dickey-Fuller test for uhat
including one lag of (1-L)uhat
sample size 125
unit-root null hypothesis: a = 1

  model: (1-L)y = (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.121072
  test statistic: tau_ct(2) = -2.92971
  asymptotic p-value 0.296
  1st-order autocorrelation coeff. for e: -0.007
```

The $p$-value for the Engle-Granger $\tau$ is 0.296 which is not significantly different from zero at 5%. Therefore, the series do not appear to be cointegrated. This means that the ARDL can be estimated by least squares using the differenced data. A constant will be included since the levels series show a trend.

```
m1 <- ols d_cons const d_cons(-1) d_y
```

$$\widehat{\text{d\_cons}} = \underset{(127.6)}{785.8} + \underset{(0.08467)}{0.2825}\ \text{d\_cons\_1} + \underset{(0.02766)}{0.05730}\,\text{d\_y}$$

$$T = 127 \quad \bar{R}^2 = 0.1138 \quad F(2, 124) = 9.0906 \quad \hat{\sigma} = 810.04$$

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

## 12.7 Script

```
set verbose off

open "@workdir\data\gdp5.gdt"
diff gdp
setinfo gdp -d "real US gross domestic product" -n "Real GDP"
setinfo d_gdp -d "= first difference of GDP" -n "D.GDP"

string title = "U.S. GDP and Change in GDP"
string xname = "Year"
string yname = "GDP $Trillion"
string y2name = "Change in Quarterly GDP"
list plotmat =  gdp d_gdp
g1 <- plot plotmat
    options time-series with-lines
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
    printf "set y2label \"%s\"", y2name
end plot --output=display

# summary statistics for subsamples and full sample
smpl 1984:2 2000:3
summary gdp d_gdp --simple
smpl 2000:4 2016:4
summary gdp d_gdp --simple
smpl full

GDP <- corrgm gdp   24 --plot=display
D_GDP <- corrgm d_gdp 24 --plot=display

open "@workdir\data\usdata5.gdt"
diff br infn ffr               # take differences

# change series attributes
setinfo br -d "3-year Bond rate" -n "3-year Bond rate"
setinfo d_br -d "Change in the 3-year Bond rate" -n "D.bond rate"
setinfo infn -d "annual inflation rate" -n "inflation rate"
setinfo d_infn -d "Change in the annual inflation rate" -n "D.inflation"
setinfo ffr -d "federal funds rate" -n "Fed Funds Rate"
setinfo d_ffr -d "= first difference of f" -n "D.fed funds rate"

# multiple time series plots
g3 <- scatters infn d_infn br d_br ffr d_ffr --output=display

g4 <- gnuplot d_br --time-series --with-lines
# summary statistics for subsamples and full sample
list levels = infn ffr br
list diffs = d_infn d_ffr d_br
smpl 1954:8 1985:10
summary levels diffs --simple
smpl 1985:11 2016:12
summary levels diffs --simple
smpl full

# Example 12.2
open "@workdir\data\toody5.gdt"
logs y
square rain

Model_1 <- ols l_y const year
series l_yhat = $yhat
list yield = l_y l_yhat year

# Graph of series against lags
string title = "Wheat Yield"
string xname = "Year"
string yname = "ln(Yield)"
```

```
g3 <- plot yield
    options --with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

Model_2 <- ols rain const year
series l_yhat = $yhat
list rainfall = rain l_yhat year

# Graph of series against lags
string title = "Rainfall during growing season 1950-1997"
string xname = "Year"
string yname = "Rain"
g4 <- plot rainfall
    options --with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

list xvars = const year rain sq_rain
Model_3 <- ols l_y xvars

# Detrend
ols l_y const t
series e_ly = $uhat
ols rain const t
series e_rain = $uhat
ols sq_rain const t
series e_rain2 = $uhat

Trend <- ols l_y xvars
scalar se_rain = $stderr(rain)
Detrend <- ols e_ly e_rain e_rain2
scalar se_e_rain = sqrt(46/44)*$stderr(e_rain)
print se_rain se_e_rain

# Example 12.3
# spurious regression
open "@workdir\data\spurious.gdt"
setobs 1 1 --special-time-series
Spurious <- gnuplot rw1 rw2 --with-lines --time-series
ols rw1 rw2 const

# Graph of series against lags
string title = "rw1 vs rw2 (with least squares fit)"
string xname = "Random Walk 2"
string yname = "Random Walk 1"
list plotvars = rw1 rw2
Spurious_series <- plot plotvars
    options --fit=linear
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
end plot --output=display

modtest 1 --autocorr

# Example 12.4
# adf tests
open "@workdir\data\usdata5.gdt" --preserve
diff br infn ffr            # take differences
list levels = ffr br

adf 19 levels --c --test-down=BIC
adf 2 levels --c --verbose

adf 2 levels --c --gls
kpss -1 levels
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
print mlag

# Example 12.5
open "@workdir\data\gdp5.gdt"
adf 5 gdp --ct --test-down --verbose

# Example 12.6
open "@workdir\data\toody5.gdt"
logs y
adf 5 l_y --ct --test-down --verbose

# Example 12.7
open "@workdir\data\usdata5.gdt"
list levels = ffr br
adf 5 levels --c --test-down=BIC --difference --verbose

# Example 12.8
open "@workdir\data\usdata5.gdt"
ols br const ffr
series res = $uhat
diff res br ffr
ols d_res res(-1) d_res(-1 to -2)

coint 2 br ffr --skip-df

# Example 12.9
open "@workdir\data\usdata5.gdt"
```

```
diff br ffr
ols br const ffr
series res = $uhat

ols d_br res(-1) d_br(-1 to -2) d_ffr(0 to -4)
scalar g1 = $coeff(d_br_1)
scalar g2 = $coeff(d_br_2)
scalar d0 = $coeff(d_ffr)
scalar d1 = $coeff(d_ffr_1)
scalar d2 = $coeff(d_ffr_2)
scalar d3 = $coeff(d_ffr_3)
scalar d4 = $coeff(d_ffr_4)

ols br const ffr
scalar b1 = $coeff(const)
scalar b2 = $coeff(ffr)
scalar a = 0
nls d_br=-a*(br(-1)-b1-b2*ffr(-1))+ g1*d_br(-1) + g2*d_br(-2)\
    + d0*d_ffr+d1*d_ffr(-1) + \
    d2*d_ffr(-2) + d3*d_ffr(-3) + d4*d_ffr(-4)
    params a b1 b2 g1 g2 d0 d1 d2 d3 d4
end nls
scalar theta1 = 1-$coeff(a)
series ehat = br-$coeff(b1)-$coeff(b2)*ffr
adf 2 ehat --nc --verbose

# Example 12.10
open "@workdir\data\cons_inc.gdt"
diff cons y
g6 <- scatters cons y --with-lines

smpl 1985:1 2016:3
list vars = cons y
adf 1 vars --ct --verbose

adf 0 vars --verbose --difference
coint 1 vars --ct --skip-df
m1 <- ols d_cons const d_cons(-1) d_y
```

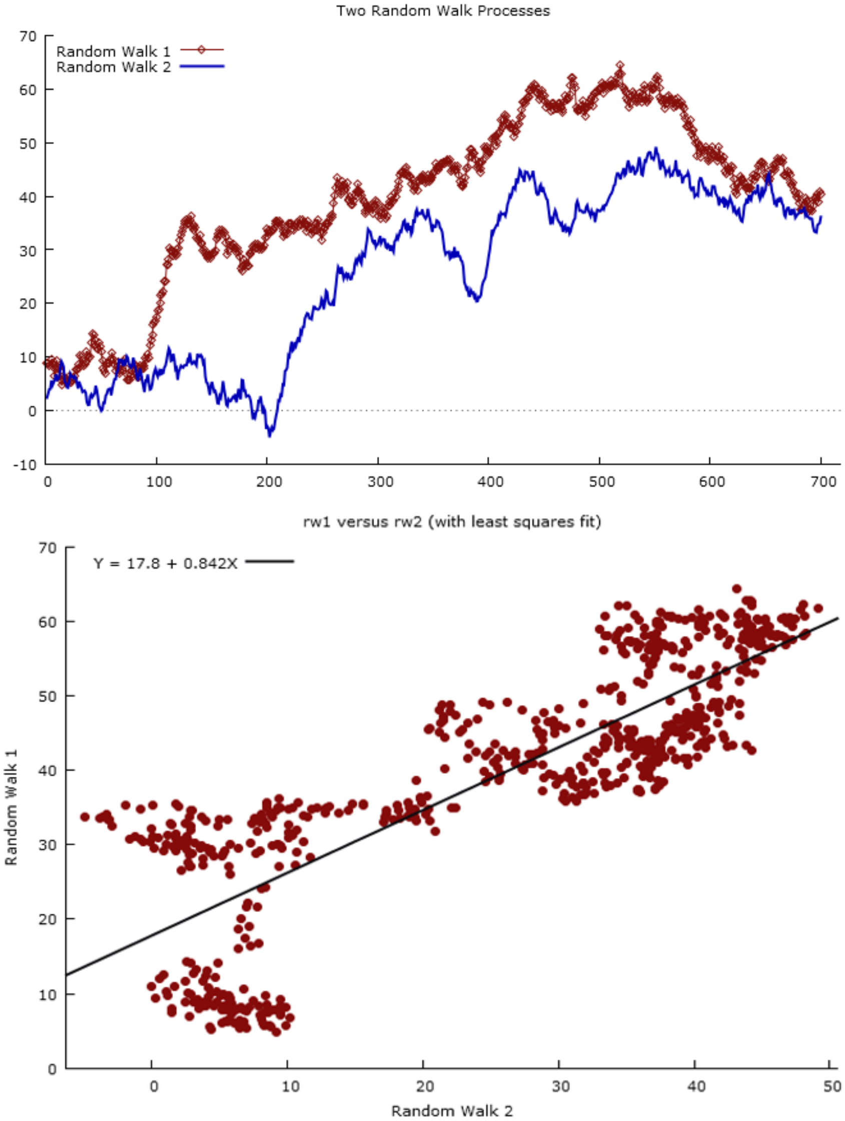
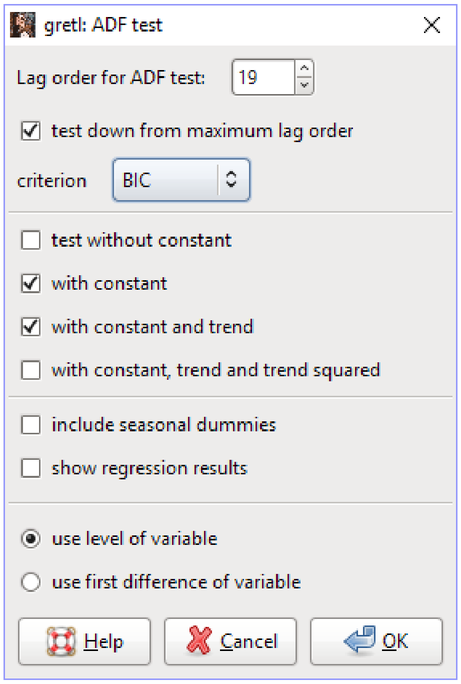


Figure 12.9: The two independent random walk series appear to be related. The top graph is a simple time-series plot and the bottom is an XY scatter with least squares fit.



Lag order for ADF test: 19

☑ test down from maximum lag order

criterion BIC

☐ test without constant

☑ with constant

☑ with constant and trend

☐ with constant, trend and trend squared

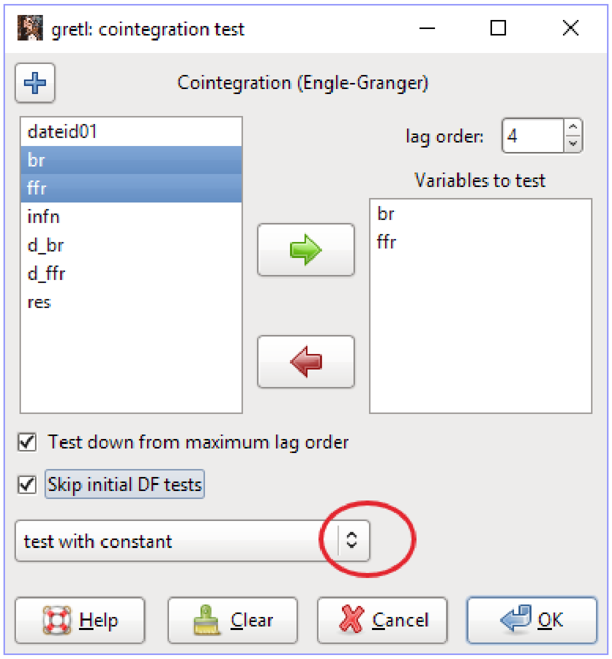
☐ include seasonal dummies

☐ show regression results

● use level of variable

○ use first difference of variable

Figure 12.10: ADF test dialog box



lag order: 4

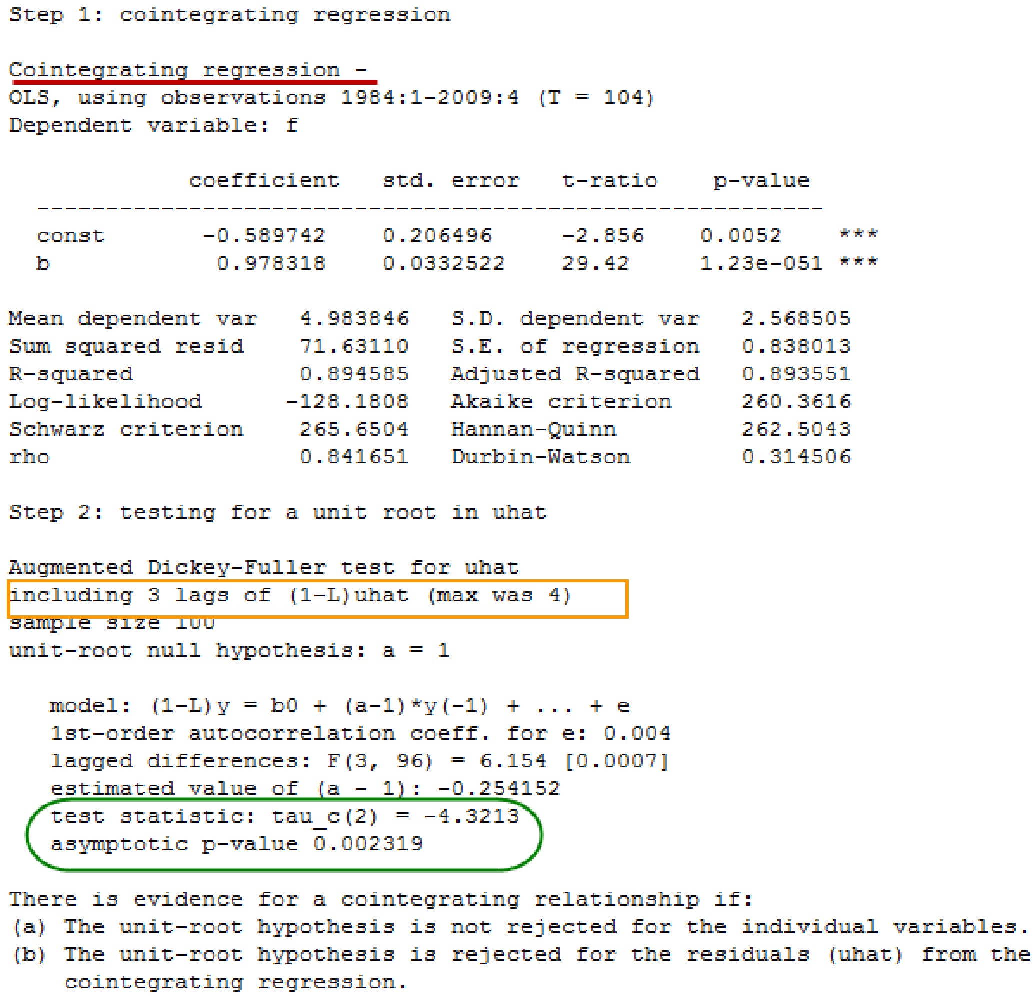
Variables to test
br
ffr

☑ Test down from maximum lag order

☑ Skip initial DF tests

test with constant

Figure 12.11: 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 12.12: The results from the Engle-Granger test. The output from the Dickey-Fuller regressions is suppressed using the `--skip-df` option.

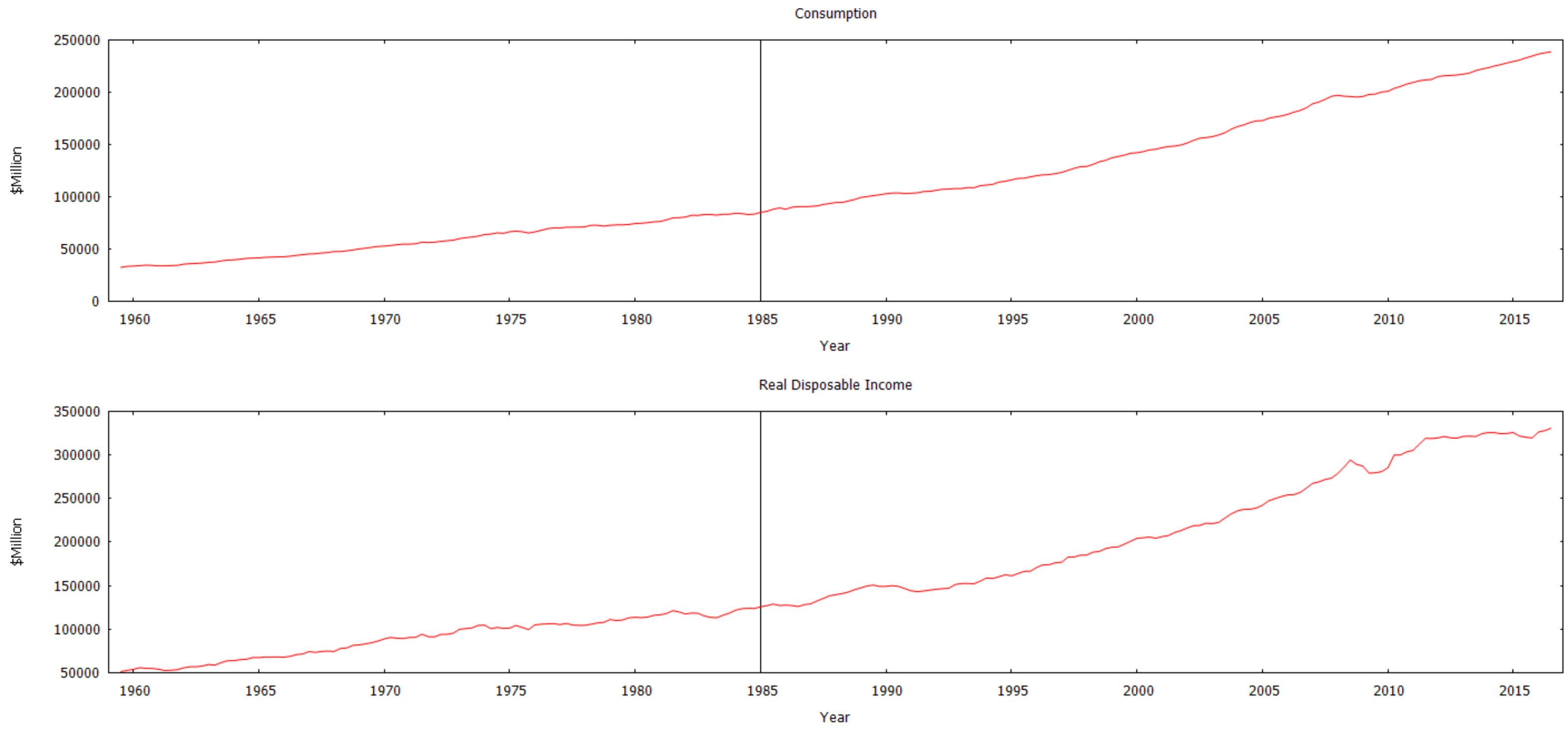


Figure 12.13: Plots of Real Consumption Expenditures and Real Disposable Income.

# Chapter 13

# Vector Error Correction and Vector Autoregressive Models

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 the 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:

$$\begin{aligned}\Delta y_t =&\beta_{11}\Delta y_{t-1} + \beta_{12}\Delta x_{t-1} + v_t^{\Delta y} \qquad (13.3)\\ \Delta x_t =&\beta_{21}\Delta y_{t-1} + \beta_{22}\Delta x_{t-1} + v_t^{\Delta x} \qquad (13.4)\end{aligned}$$

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 variables. Introducing the cointegrating relationship leads to a model known as the **vector error correction** (VEC) model.

In this example from *POE5*, 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 have real GDP of 100 in the year 2000. A vector error correction model is used because (1) the time series are not stationary in their levels but are in their differences and (2) the variables are cointegrated.

The authors of *POE5* don't discuss how they 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 approaches this. There are several 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 - 2000:4.[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 "@workdir\data\gdp.gdt"
setobs 4 1970:1 --time-series
```

One purpose of the plots is to help determine whether the Dickey-Fuller regressions should contain constants, trends or squared trends. The simplest way to do this is from a script or the console using the `scatters` command.

```
g1 <- scatters usa diff(usa) aus diff(aus) --output=display
```

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] *POE5* refers to these variables as $U$ and $A$, respectively.

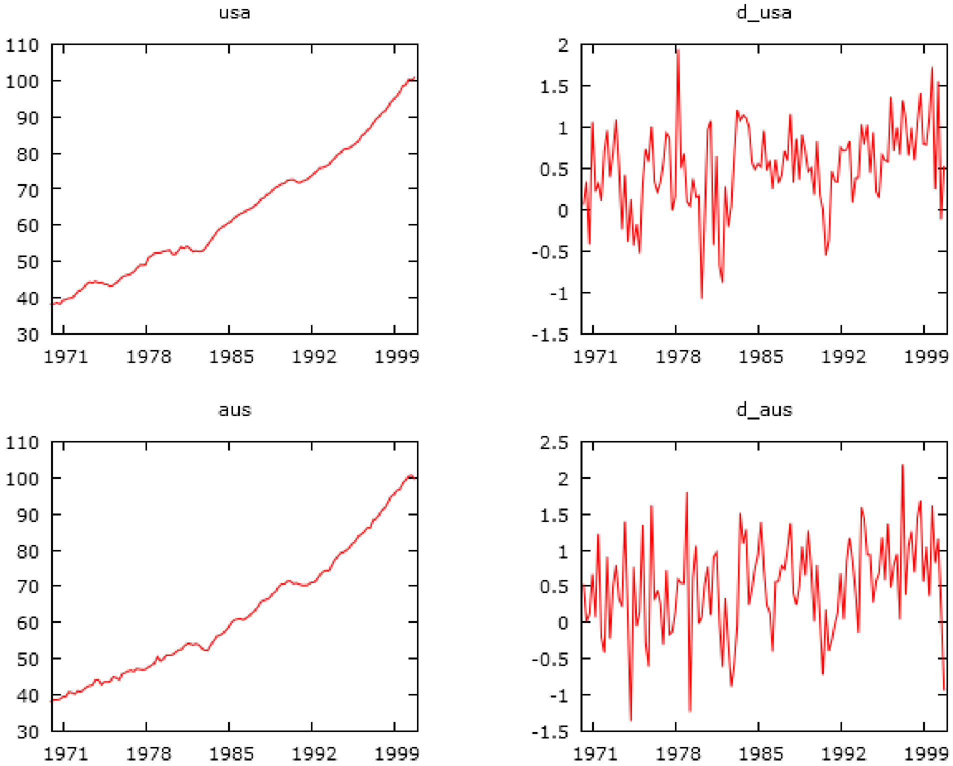


Figure 13.1: The levels of Australian and U.S. GDP appear to be nonstationary and cointegrated. The difference plots have a nonzero mean, indicating a constant in their ADF regressions.

This takes two steps from the pull-down menu. First, use the mouse to highlight the two series and 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 upward trend. This means that the augmented Dickey-Fuller (ADF) regressions may require 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 some 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=ttest` option for the augmented Dickey-Fuller regressions which was used in section 12.4. 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 uses the `--test-down` option. When the ttest method is used the USA series contains a very long significant lag twelve periods into the past. This seems unlikely to be true, so we opt to test down using the more parsimonious BIC criterion in this example.

```
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
adf mlag usa --ctt --test-down=BIC --verbose
adf mlag aus --ctt --test-down=BIC --verbose
```

Using this criterion, the USA series contains only two lags and the Australian series has none.

```
Augmented Dickey-Fuller test for usa
testing down from 12 lags, criterion BIC
sample size 121
unit-root null hypothesis: a = 1

  with constant, linear and quadratic trend
  including 2 lags of (1-L)usa
  model: (1-L)y = b0 + b1*t + b2*t^2 + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0916804
  test statistic: tau_ctt(1) = -2.97815
  asymptotic p-value 0.3067
  1st-order autocorrelation coeff. for e: -0.015
  lagged differences: F(2, 115) = 7.744 [0.0007]

Augmented Dickey-Fuller test for aus
testing down from 12 lags, criterion BIC
sample size 123
unit-root null hypothesis: a = 1

  with constant, linear and quadratic trend
  including 0 lags of (1-L)aus
  model: (1-L)y = b0 + b1*t + b2*t^2 + (a-1)*y(-1) + e
  estimated value of (a - 1): -0.09385
  test statistic: tau_ctt(1) = -2.49745
  p-value 0.5637
  1st-order autocorrelation coeff. for e: 0.081
```

The *p*-values of the ADF statistics are 0.3067 and 0.5637, both insignificant at the 5% or 10% level and indicating the series are nonstationary.

This is repeated for the differenced series using the commands:

```
adf mlag usa --ct --test-down=BIC --difference
adf mlag aus --ct --test-down=BIC --difference
```

Testing down selects models with no lags for both series and both ADF statistics are significant at the 5% level and we conclude that the differences are stationary and that the series are I(1).

```
Augmented Dickey-Fuller test for d_usa
       testing down from 12 lags, criterion BIC
       sample size 122
       unit-root null hypothesis: a = 1

         with constant and trend
         including 0 lags of (1-L)d_usa
         model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + e
         estimated value of (a - 1): -0.770169
         test statistic:  τ_ct(1) = -8.62481
         p-value 6.209e-011
         1st-order autocorrelation coeff. for e: -0.040

       Augmented Dickey-Fuller test for d_aus
       testing down from 12 lags, criterion BIC
       sample size 122
       unit-root null hypothesis: a = 1

         with constant and trend
         including 0 lags of (1-L)d_aus
         model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + e
         estimated value of (a - 1): -0.956746
         test statistic:  τ_ct(1) = -10.1139
         p-value 1.21e-013
         1st-order autocorrelation coeff. for e: 0.002
```

### Testing Up

The other strategy is to test the residuals from the augmented Dickey-Fuller regressions for autocorrelation. Starting with a small model, test the residuals of the Dickey-Fuller regression for autocorrelation using an *LM* (or *LMF*) 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, quit. 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 must estimate the ADF equations manually using the `ols` command. Since the data series has a constant and quadratic trend, define a time trend (`genr time`) and possibly trend squared (`square time`) to include in the regressions.[2] Note this is one of the cases (see page (17)) that requires `genr` instead of `series`. The `genr time` is a special function for the `genr` command. Other cases include `genr dummy` and `genr unitdum`.

You also must generate the differences using `diff`. The script to add time, squares and differences to the data:

```
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 section 9.8. To reduce output, I put the test results into a matrix that is printed once the loop finishes.

```
matrix mat = zeros(12,3)
loop i=1..12 --quiet
    ols d_usa(0 to -i) usa(-1)  const time sq_time --quiet
    modtest 1 --autocorr --silent
    mat[i,]= i ~ $test ~ $pvalue
endloop
cnameset(mat, " Lags LMF P-value " )
printf "%10.4g\n", mat
```

Only the *LMF* test statistic and *p*-value are returned from `modtest`, but those are sufficient for our needs. Notice that the `cnameset` command is used to add proper names to the matrix columns and that printing is handled by `printf`. Also, output is suppressed using `--quiet` flags for both the loop and the regression. The `--silent` flag is peculiar to `modtest`. It suppresses everything in this case.

The first `ols` regression in the loop 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 --silent`. If the *p*-value is greater than 0.10 then, this is your model. If not, consider the outcome of the next

[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.

loop which has added another lag of d_usa to the model. Stop when the $p$-value is greater than 0.10.

The results are:

```
Lags       LMF   P-value
   1     6.742   0.01064
   2    0.4536     0.502
   3     1.291    0.2582
   4   0.07833    0.7801
   5    0.5533    0.4586
   6    0.5548     0.458
   7     0.537    0.4653
   8     4.387   0.03868
   9     2.118    0.1487
  10      0.84    0.3617
  11     3.547   0.06268
  12   0.03076    0.8612
```

The p-value for lag 1 is significant, but not for lag 2. This indicates that 2 lagged differences are required for the ADF.

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; this is the same result obtained by testing down based on the BIC model selection rule.

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 *POE5.* 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" of Hill et al. (2018) and from Table 12.4 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.

[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.

```
ols aus usa
series uhat = $uhat
ols diff(uhat) uhat(-1)
```

The result is:

$$\Delta\widehat{e_t} = \underset{(0.044279)}{-0.127937}\hat{e}_{t-1} \tag{13.7}$$

$$T = 123 \quad \bar{R}^2 = 0.0640 \quad F(1, 122) = 8.3482 \quad \hat{\sigma} = 0.5985$$
$$\text{(standard errors in parentheses)}$$

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

#### Example 3.1 in *POE5*

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:

$$\begin{aligned}\Delta aus_t &= \beta_{11} + \beta_{12}\hat{e}_{t-1} + v_{1t} \qquad (13.8)\\ \Delta usa_t &= \beta_{21} + \beta_{22}\hat{e}_{t-1} + v_{2t} \qquad (13.9)\end{aligned}$$

and the estimates

$$\begin{aligned}\Delta\widehat{aus}_t &= \underset{(8.491)}{0.491706} + \underset{(-2.077)}{-0.0987029}\hat{e}_{t-1}\\ \Delta\widehat{usa}_t &= \underset{(10.924)}{0.509884} + \underset{(0.790)}{+0.0302501}\hat{e}_{t-1}\end{aligned}$$
$$\text{($t$-statistics in parentheses)}$$

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 30 of Cottrell and Lucchetti (2018) 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 *POE5* 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)
            --uc (unrestricted constant)
            --crt (constant and restricted trend)
            --ct (constant and unrestricted trend)
            --seasonals (include centered seasonal dummies)
            --quiet (skip output of individual equations)
            --silent (don't print anything)
            --impulse-responses (print impulse responses)
            --variance-decomp (print variance decompositions)
Examples:   vecm 4 1 Y1 Y2 Y3
            vecm 3 2 Y1 Y2 Y3 --rc
                  vecm 3 2 Y1 Y2 Y3 ; X1 --rc
```

The order parameter to this command represents the lag order of the VAR system. The number of lags in the VECM itself (where the dependent variable is given as a first difference) is one less than order.

The rank parameter represents the cointegration rank, or in other words the number of cointegrating vectors. This must be greater than zero and less than or equal to (generally, less than) the number of endogenous variables given in *ylist*.

After some experimentation I use a third order model with only 1 cointegrating vector. 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 resolve this tricky issue.

The model is estimated via a script:

```
vecm 3 1 aus usa
series ec_unrest = aus + $jbeta[2,1]*usa
```

The top portion of the results are:

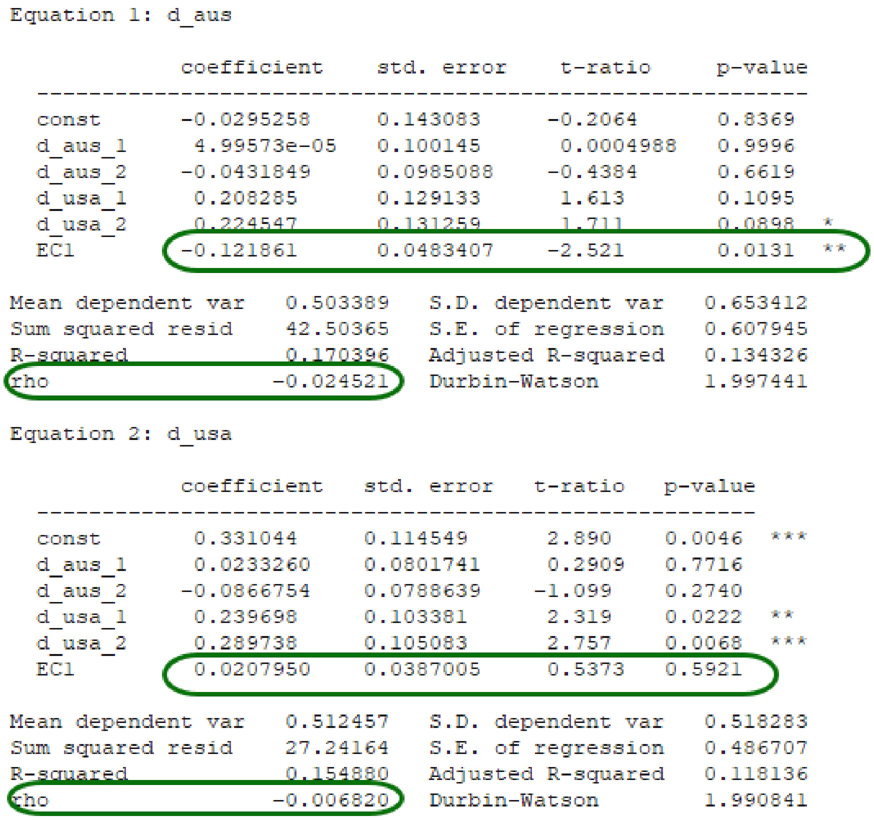
```
VECM system, lag order 3
Maximum likelihood estimates, observations 1970:4-2000:4 (T = 121)
Cointegration rank = 1
Case 3: Unrestricted constant

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
```

This reveals important information. First, the VECM is estimated by maximum likelihood, and not least squares as we have done previously. This is apparent in the estimation of the cointegrating vector, which is normalized on Australia and estimates the USA coefficient to be 1.0268. This is larger than the OLS estimate of .985.

Also, the output informs us of our choices: lag order of 3, a single cointegrating vector, and an unrestricted constant in the VECM. Next are the estimates from the cointegrating equation. The adjustment vectors are actually the coefficients on the lagged residuals from the cointegrating relationship. Generally, these should have opposite signs in two variable models, otherwise the adjustments to shocks may not be equilibrating. Finally, some model selection statistics (not shown here) appear at the bottom that may be useful in determining the order of the VECM.

The remaining regression results appear in Figure 13.2. The error correction coefficient is negative and different from zero for the USA. Autocorrelation in the residuals is not evident. For Australia, the error correction term is not significantly different from zero and there is no remaining autocorrelation.

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



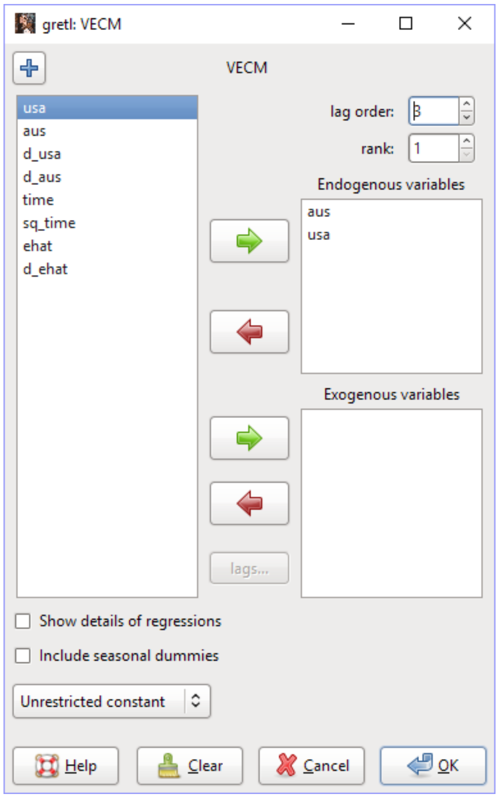
```
Equation 1: d_aus

             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 13.2: The output from the `vecm 3 1 aus usa` command. In the USA equation, the error correction coefficient is negative and different from zero. Autocorrelation in the residuals is not evident. For Australia, the error correction term is not significantly different from zero and there is no remaining autocorrelation.

output. Check 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.

The dialog boxes are also useful. Choose **Model>Time-Series>VECM** to bring up the appropriate dialog box shown in Figure 13.3. It allows you to add endogenous variables to the VAR, exogenous variables (which must be I(0)), choose lags, number of cointegrating vectors, and a 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



lag order: 3

rank: 1

Endogenous variables: aus, usa

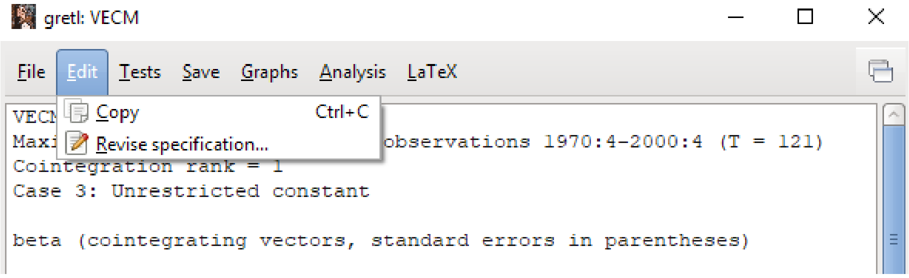
☐ Show details of regressions

☐ Include seasonal dummies

Unrestricted constant

Figure 13.3: The VECM dialog box

access to tests, plots, and additional tools for analysis. Furthermore, there is also a handy facility that allows quick respecification 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
Maxi                               observations 1970:4-2000:4 (T = 121)
Cointegration rank = 1
Case 3: Unrestricted constant

beta (cointegrating vectors, standard errors in parentheses)
```

One more thing is worth checking. Plot the error correction terms, which are shown in Figure 13.4. 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 $-0.985$. 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.

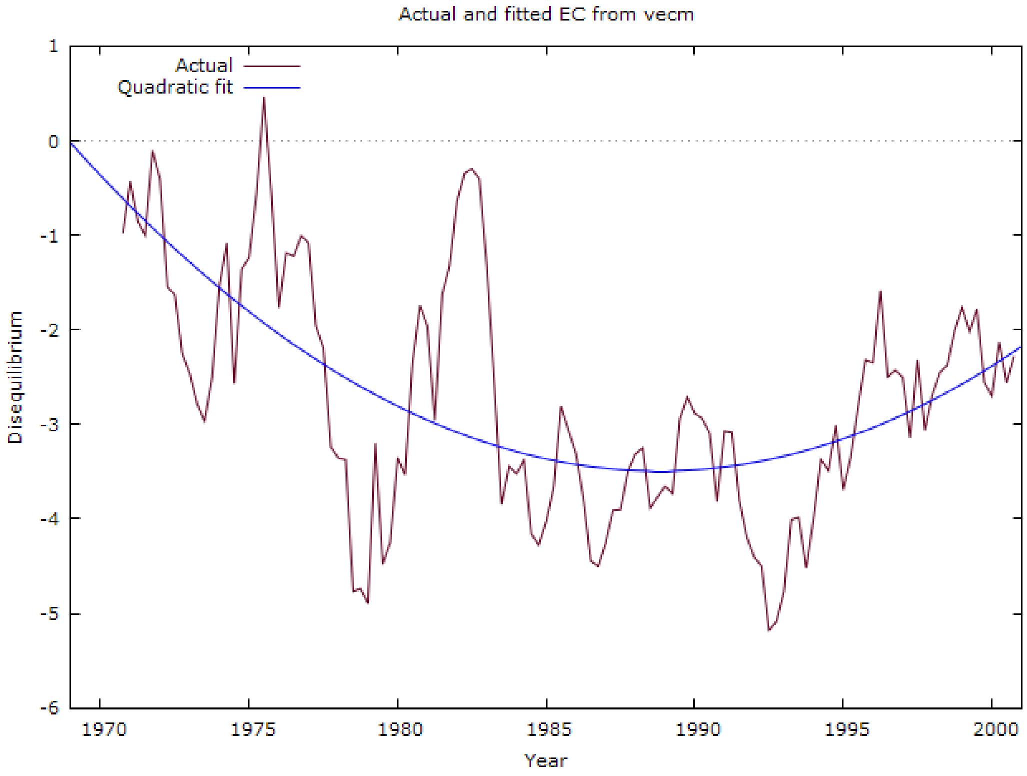


Figure 13.4: Plot of the error correction terms from the `vecm 3 1 aus usa` command.

```
vecm 3 1 aus usa
restrict --full
    b[1]+b[2]=0
end restrict
series ec_rest = $ec
```

Note, if $\beta_1 + \beta_2 = 0$ it implies that $usa - aus = 0$. Also, the residuals are saved as a series using the accessor `$ec`. Gretl performs a likelihood ratio test that is distributed $\chi^2(1)$ if the restriction is true. The result is:

```
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
```

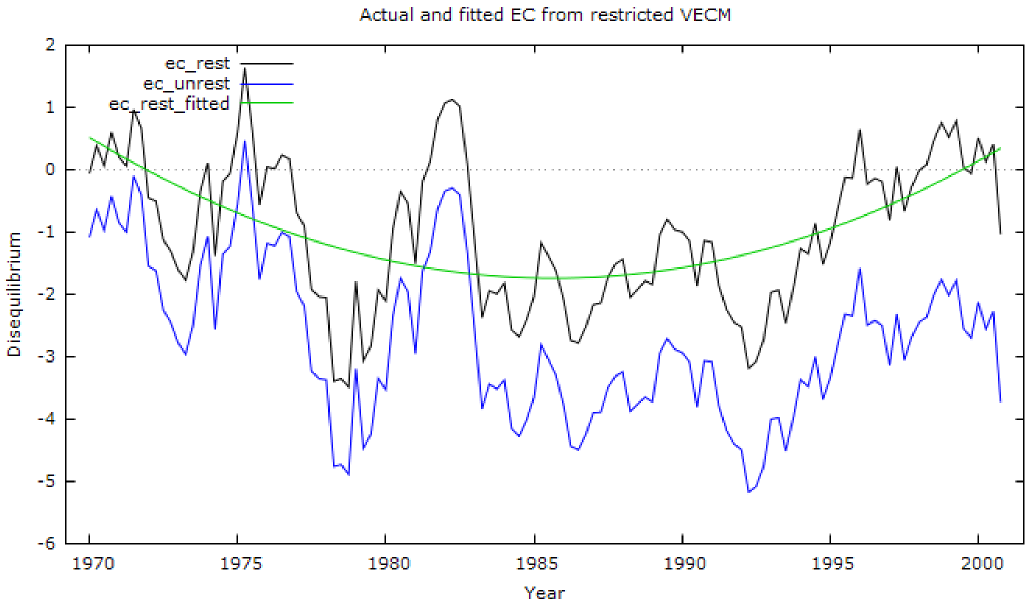


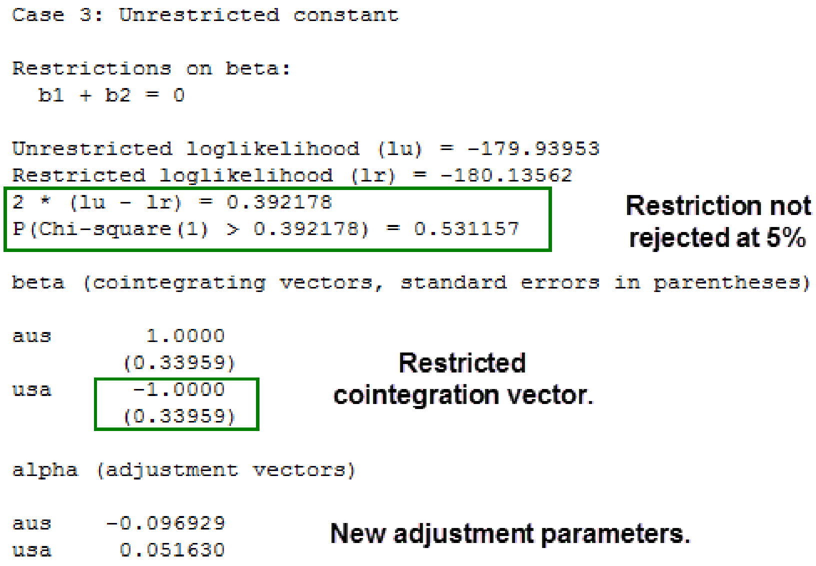
Figure 13.5: Plots of the error correction terms from restricted and unrestricted VECMs estimated by maximum likelihood. The restricted residuals (black) are fitted to a quadratic trend (green). The restricted cointegrating relationship is $aus = usa$.

which is not significant at 5% ($p$-value is .53 > .05). The restriction is imposed and the plot recast as shown in Figure 13.5 alongside the unrestricted plot (blue).

The `plot` command was used to control **gnuplot**. The script assumes that you have estimated both versions of the VECM and saved the restricted and unrestricted error correction terms into `ec_rest` and `ec_unrest`, respectively. Then a quadratic trend is fitted to the restricted residuals and plotted.

```
string title = "Actual and fitted EC from restricted VECM"
string xlabel = "Year"
string ylabel = "Disequilibrium"
ols ec_rest const time sq_time
series ec_rest_fitted = $yhat
list plotvars = ec_rest ec_unrest ec_rest_fitted
g2 <- plot plotvars
    options time-series with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xlabel
    printf "set ylabel \"%s\"", ylabel
end plot --output=display
```

You can see that the error correction terms from the restricted model have the same basic shape as in Figure 13.4, but the now there are many more positive disequilibria. The regression output from the restricted VECM appears below in Figure 13.6: 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.



```
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 13.6: Output from the restricted VECM model. The cointegrating relationship is AUS=USA.

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` even though the `$ec` accessor is available. 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
# error correction terms using the accessor
series ec = $ec
```

## 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.

### Example 13.2 in *POE5*

In the example from *POE5*, 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 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 "@workdir\data\fred5.gdt"
scatters c diff(c) y diff(y)
```

The plots appear in Figure 13.7. 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.

```
list yvars = consn y
adf 12 yvars --ct --test-down=BIC --verbose
```

After some experimentation, the decision was made to leave a trend in the ADF regressions. The

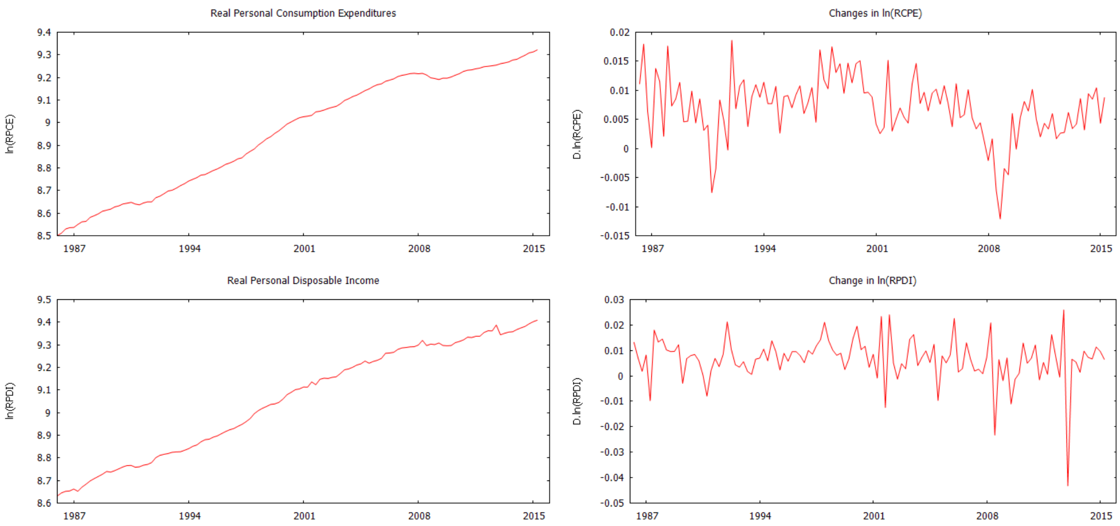


Figure 13.7: Natural logs of consumption and income and their differences.

term was significant for both series. The test-down procedure chose 3 lagged differences of $c$ in the first model and 1 lagged difference of $y$ in the second. In both cases, the unit root hypothesis could not be rejected at 10%. The first order autocorrelation coefficient for the residuals is very small in both cases. See Figures 13.8 and 13.9. It is a good idea to confirm that the differences are stationary, since VAR in differences will require this.

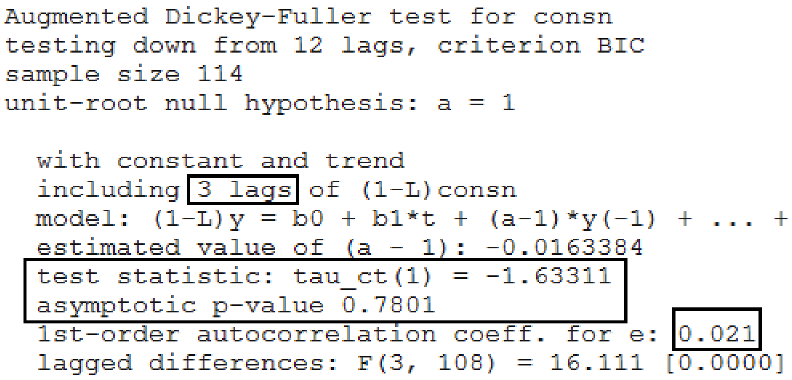
```
adf 12 yvars --c  --test-down=BIC --difference
```

which produces, in part:

```
test with constant
including 2 lags of (1-L)d_consn
test statistic: tau_c(1) = -2.83728
asymptotic p-value 0.05311
1st-order autocorrelation coeff. for e: 0.033

test with constant
including 0 lags of (1-L)d_y
test statistic: tau_c(1) = -13.0482
p-value 3.021e-018
1st-order autocorrelation coeff. for e: 0.022
```

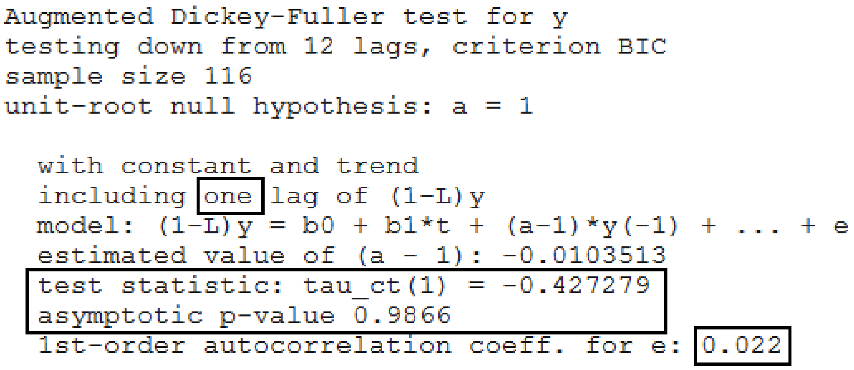
The $p$-value for `d_consn` is 0.053, which is significant at 10%.

If *consumption* and *income* are cointegrated then estimate a VECM. The Engle-Granger tests reveals that they are not.



```
Augmented Dickey-Fuller test for consn
testing down from 12 lags, criterion BIC
sample size 114
unit-root null hypothesis: a = 1

  with constant and trend
  including 3 lags of (1-L)consn
  model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0163384
  test statistic: tau_ct(1) = -1.63311
  asymptotic p-value 0.7801
  1st-order autocorrelation coeff. for e: 0.021
  lagged differences: F(3, 108) = 16.111 [0.0000]
```

Figure 13.8: ADF tests of ln(RPCE)



```
Augmented Dickey-Fuller test for y
testing down from 12 lags, criterion BIC
sample size 116
unit-root null hypothesis: a = 1

  with constant and trend
  including one lag of (1-L)y
  model: (1-L)y = b0 + b1*t + (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0103513
  test statistic: tau_ct(1) = -0.427279
  asymptotic p-value 0.9866
  1st-order autocorrelation coeff. for e: 0.022
```

Figure 13.9: ADF tests of ln(RPDI)

```
coint 8 yvars --test-down=BIC --nc
```

This produces the result:

```
Augmented Dickey-Fuller test for uhat
testing down from 8 lags, criterion BIC
sample size 116
unit-root null hypothesis: a = 1

  model: (1-L)y = (a-1)*y(-1) + ... + e
  estimated value of (a - 1): -0.0923617
  test statistic: tau_nc(2) = -2.26548
  asymptotic p-value 0.1481
  1st-order autocorrelation coeff. for e: -0.006
```

The $p$-value on the test statistic is 0.1481. 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)
            --quiet (skip output of individual equations)
            --silent (don't print anything)
            --impulse-responses (print impulse responses)
            --variance-decomp (print variance decompositions)
            --lagselect (show criteria for lag selection)
Examples:   var 4 x1 x2 x3 ; time mydum
            var 4 x1 x2 x3 --seasonals
            var 12 x1 x2 x3 --lagselect
```

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 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     781.15658           -14.764887   -14.613232*  -14.703434*
   2     786.28675  0.03626  -14.786414   -14.533656   -14.683992
   3     791.93877  0.02335  -14.817881   -14.464020   -14.674490
   4     796.37888  0.06416  -14.826264*  -14.371300   -14.641904
   5     796.91787  0.89775  -14.760340   -14.204273   -14.535011
```

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 `modtest p --autocorr` 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 below:

VAR system, lag order 1
OLS estimates, observations 1986:3–2015:2 ($T = 116$)

Equation 1: d_consn
HAC standard errors, bandwidth 3 (Bartlett kernel)

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 0.00367073 | 0.00122096 | 3.006 | 0.0033 |
| d_consn_1 | 0.348192 | 0.129414 | 2.691 | 0.0082 |
| d_y_1 | 0.131345 | 0.0497994 | 2.637 | 0.0095 |

| | | | |
|---|---|---|---|
| Mean dependent var | 0.006980 | S.D. dependent var | 0.005284 |
| Sum squared resid | 0.002534 | S.E. of regression | 0.004736 |
| $R^2$ | 0.210822 | Adjusted $R^2$ | 0.196854 |
| $F(2, 113)$ | 8.351617 | P-value($F$) | 0.000414 |
| $\hat{\rho}$ | −0.121496 | Durbin–Watson | 2.210652 |

F-tests of zero restrictions

| | | |
|---|---|---|
| All lags of d_consn | $F(1, 113) = 7.23896$ | [0.0082] |
| All lags of d_y | $F(1, 113) = 6.95635$ | [0.0095] |

Equation 2: d_y
HAC standard errors, bandwidth 3 (Bartlett kernel)

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 0.00438419 | 0.00113034 | 3.879 | 0.0002 |
| d_consn_1 | 0.589538 | 0.124973 | 4.717 | 0.0000 |
| d_y_1 | −0.290937 | 0.106648 | −2.728 | 0.0074 |

| | | | |
|---|---|---|---|
| Mean dependent var | 0.006580 | S.D. dependent var | 0.008786 |
| Sum squared resid | 0.007496 | S.E. of regression | 0.008145 |
| $R^2$ | 0.155517 | Adjusted $R^2$ | 0.140571 |
| $F(2, 113)$ | 11.28591 | P-value($F$) | 0.000034 |
| $\hat{\rho}$ | −0.051177 | Durbin–Watson | 2.101542 |

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.

## 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.

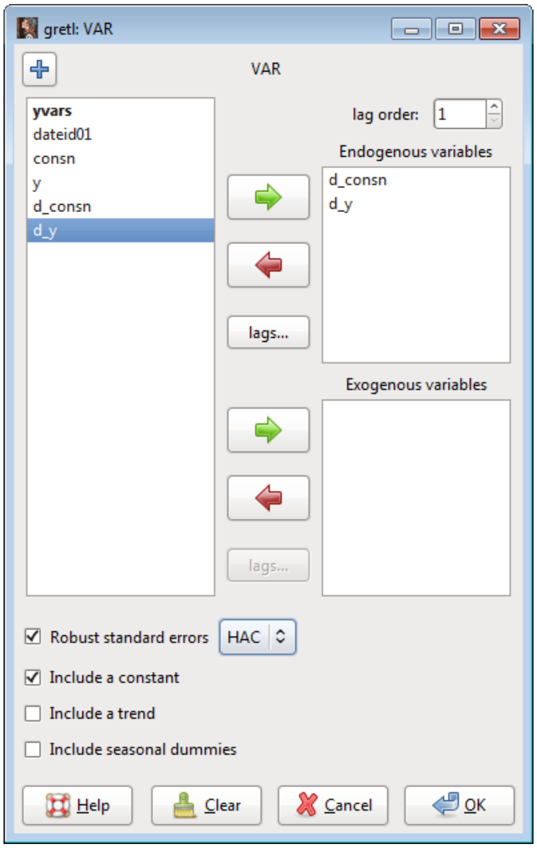
Impulse response functions (IRFs) and forecast error variance decompositions (FEVD) can be produced after using the `var` or `vecm` commands. The results can be presented in a table or a graph.

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 13.10. 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 checked and click **OK**. Also, choose HAC standard errors if desired.

Impulse responses can be generated 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 13.3.

Graphs can also be generated 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 13.11. which yields the graph shown in Figure 13.12. The forecast error variance decompositions (FEVD) are obtained similarly. Select **Analysis**>**Forecast variance decomposition** from the vector autoregression model window to obtain the result shown in Table 13.6.

To generate IRFs and the FEVDs using a script, add the options `--impulse-responses` and `--variance-decomp` to the `var` command. These can be used with the `vecm` command as well.



lag order: 1

Endogenous variables
d_consn
d_y

Exogenous variables

☑ Robust standard errors HAC

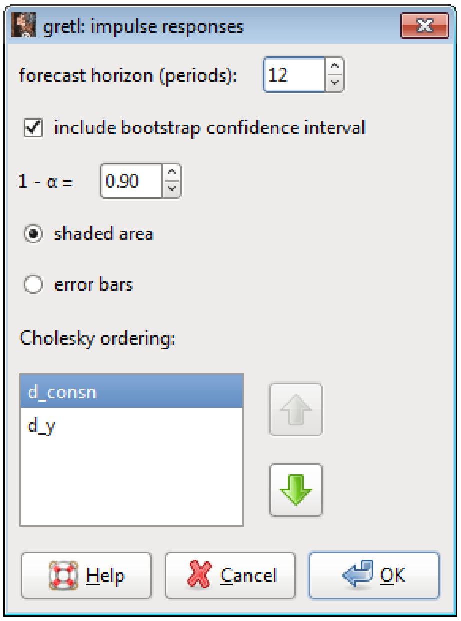
☑ Include a constant

☐ Include a trend

☐ Include seasonal dummies

Figure 13.10: From the main **gretl** window, choose **Model>Time series>Vector Autoregression** to bring up the VAR dialog box.

```
var 1 diff(c) diff(y) --impulse-responses --variance-decomp
```



forecast horizon (periods): 12

☑ include bootstrap confidence interval

1 - α = 0.90

● shaded area

○ error bars

Cholesky ordering:

d_consn
d_y

Figure 13.11: Select **Graphs>Impulse responses (combined)** from the VAR results window brings up this dialog box.

## 13.4 Script

```
set verbose off

open "@workdir\data\gdp.gdt"
setobs 4 1970:1 --time-series
# plot multiple time-series
g1 <- scatters usa diff(usa) aus diff(aus) --output=display

# ADF tests with test down
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
coint 0 aus usa --nc

# ADF tests with test down
scalar mlag = int(12*(($nobs+1)/100)^(0.25))
adf mlag usa --ctt --test-down=BIC --verbose
adf mlag aus --ctt --test-down=BIC --verbose

adf mlag usa --ct --test-down=BIC --difference
adf mlag aus --ct --test-down=BIC --difference

# manually testing down based on LM tests
# USA
genr time
square time
diff usa aus
```

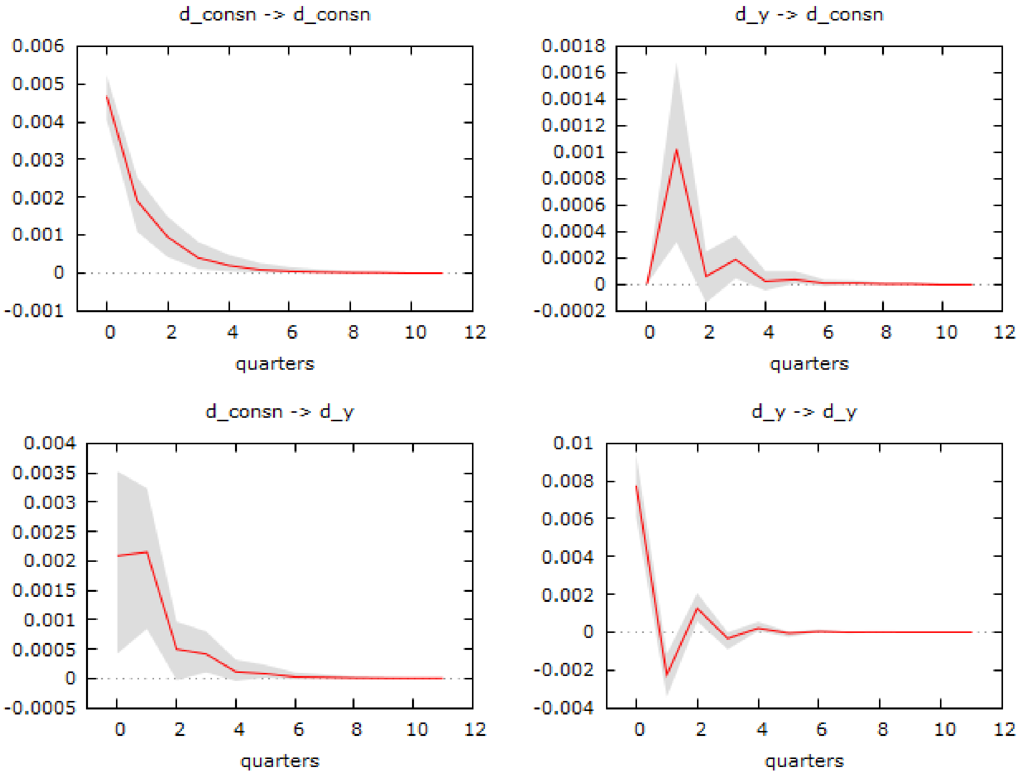


Figure 13.12: U.S. ln(*RDPI*) and ln(*RPCE*) impulse responses

```
matrix mat = zeros(12,3)
loop i=1..12 --quiet
    ols d_usa(0 to -i) usa(-1)  const time sq_time --quiet
    modtest 1 --autocorr --silent
    mat[i,]= i ~ $test ~ $pvalue
endloop
cnameset(mat, " Lags LMF P-value " )
printf "%10.4g\n", mat
# Australia
loop i=1..12
    ols d_aus(0 to -i) aus(-1)  const time sq_time --quiet
    modtest 1 --autocorr --quiet
    mat[i,]= i ~ $test ~ $pvalue
    printf "%10.4g\n", mat
endloop
cnameset(mat, " Lags LMF P-value " )
printf "%10.4g\n", mat

# Example 13.1 in POE5
ols aus usa
series ehat = $uhat
ols diff(ehat) ehat(-1)
ols diff(aus) const ehat(-1)
```

```
modtest 1 --autocorr
ols diff(usa) const ehat(-1)
modtest 1 --autocorr

ols aus usa
series ehat = $uhat

string title = "Actual and fitted EC from VECM--OLS"
string xlabel = "Year"
string ylabel = "Disequilibrium"
ols ehat const time sq_time
series fitted = $yhat
list plotvars = ehat fitted
g1 <- plot plotvars
    options time-series with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xlabel
    printf "set ylabel \"%s\"", ylabel
end plot --output=display


# Engle-Granger test
coint 8 aus usa --test-down --nc

# restricted VECM
vecm 3 1 aus usa
series ec_unrest = aus + $jbeta[2,1]*usa
restrict --full
    b[1]+b[2]=0
end restrict
series ec_rest = $ec

# collecting error correction terms from restricted model
matrix a = $vecGamma
matrix b =$jbeta
printf "\nCoefficients on the lagged differences of the cointegrated\
 variables:\n%10.3g\n", a
printf "\nThe cointegration matrix:\n %.3f\n", b

string title = "Actual and fitted EC from restricted VECM"
string xlabel = "Year"
string ylabel = "Disequilibrium"
ols ec_rest const time sq_time
series ec_rest_fitted = $yhat
list plotvars = ec_rest ec_unrest ec_rest_fitted
g2 <- plot plotvars
    options time-series with-lines
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xlabel
    printf "set ylabel \"%s\"", ylabel
end plot --output=display

# VAR estimation
open "@workdir\data\fred5.gdt"
g1 <- scatters consn diff(consn) y diff(y) --output=display

list yvars = consn y
adf 12 yvars --ct --test-down=BIC
adf 12 yvars --c  --test-down=BIC --difference

# Engle-Granger test
coint 8 yvars --test-down=BIC --nc

# If not cointegrated estimate var of differences
m1 <- var 12 diff(consn) diff(y) --lagselect
m2 <- var 1 diff(consn) diff(y) --robust-hac
modtest 1 --autocorr

m3 <- var 1 diff(consn) diff(y) --impulse-responses --variance-decomp
```

Responses to a one-standard error shock in d_consn

| period | d_consn | d_y |
|---:|---:|---:|
| 1 | 0.00467417 | 0.00208478 |
| 2 | 0.00190133 | 0.00214907 |
| 3 | 0.000944298 | 0.000495666 |
| 4 | 0.000393900 | 0.000412492 |
| 5 | 0.000191332 | 0.000112210 |
| 6 | 8.13583e–005 | 8.01513e–005 |
| 7 | 3.88558e–005 | 2.46448e–005 |
| 8 | 1.67662e–005 | 1.57369e–005 |
| 9 | 7.90483e–006 | 5.30590e–006 |
| 10 | 3.44930e–006 | 3.11651e–006 |
| 11 | 1.61036e–006 | 1.12678e–006 |
| 12 | 7.08710e–007 | 6.21543e–007 |

Responses to a one-standard error shock in d_y

| period | d_consn | d_y |
|---:|---:|---:|
| 1 | 0.000000 | 0.00776375 |
| 2 | 0.00101973 | −0.00225876 |
| 3 | 5.83843e–005 | 0.00125833 |
| 4 | 0.000185605 | −0.000331675 |
| 5 | 2.10620e–005 | 0.000205918 |
| 6 | 3.43799e–005 | −4.74922e–005 |
| 7 | 5.73292e–006 | 3.40855e–005 |
| 8 | 6.47313e–006 | −6.53697e–006 |
| 9 | 1.39529e–006 | 5.71800e–006 |
| 10 | 1.23686e–006 | −8.41003e–007 |
| 11 | 3.20203e–007 | 9.73856e–007 |
| 12 | 2.39403e–007 | −9.45591e–008 |

Table 13.3: Impulse response functions (IRF)

Decomposition of variance for d_consn

| period | std. error | d_consn | d_y |
|---:|---:|---:|---:|
| 1 | 0.00467417 | 100.0000 | 0.0000 |
| 2 | 0.00514809 | 96.0764 | 3.9236 |
| 3 | 0.0052343 | 96.1922 | 3.8078 |
| 4 | 0.00525238 | 96.0935 | 3.9065 |
| 5 | 0.00525591 | 96.0971 | 3.9029 |
| 6 | 0.00525665 | 96.0939 | 3.9061 |
| 7 | 0.0052568 | 96.0940 | 3.9060 |
| 8 | 0.00525683 | 96.0939 | 3.9061 |
| 9 | 0.00525684 | 96.0939 | 3.9061 |
| 10 | 0.00525684 | 96.0939 | 3.9061 |
| 11 | 0.00525684 | 96.0939 | 3.9061 |
| 12 | 0.00525684 | 96.0939 | 3.9061 |

Decomposition of variance for d_y

| period | std. error | d_consn | d_y |
|---:|---:|---:|---:|
| 1 | 0.00803879 | 6.7257 | 93.2743 |
| 2 | 0.00862222 | 12.0587 | 87.9413 |
| 3 | 0.00872764 | 12.0917 | 87.9083 |
| 4 | 0.00874367 | 12.2700 | 87.7300 |
| 5 | 0.00874682 | 12.2776 | 87.7224 |
| 6 | 0.00874732 | 12.2846 | 87.7154 |
| 7 | 0.00874742 | 12.2851 | 87.7149 |
| 8 | 0.00874743 | 12.2854 | 87.7146 |
| 9 | 0.00874744 | 12.2854 | 87.7146 |
| 10 | 0.00874744 | 12.2854 | 87.7146 |
| 11 | 0.00874744 | 12.2854 | 87.7146 |
| 12 | 0.00874744 | 12.2854 | 87.7146 |

Table 13.6: Forecast Error Variance Decompositions (FEVD)

# Chapter 14

# Time-Varying Volatility and ARCH Models

In this chapter several models in which the variance of the dependent variable changes over time are estimated. These are broadly referred to as ARCH (autoregressive conditional heteroskedasticity) models and there are many variations upon the theme.

### Example 14.1 in *POE5*

First the problem is examined graphically using data on stock returns. The data are stored in the **gretl** dataset *returns5.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 2015:12. Notice that with monthly data, the suffix has two digits, that is 1988:01 is January (01) in the year 1988.

Simple scatter plots appear below. To make the plots more informative, a set of graph labels was added to the series. In preceding examples of this, a simple syntax was used by invoking the `-n` switch of `setinfo`. In this example, the long-form syntax is used, which is marginally more informative of what it does that the `-n` switch. The plots can also be generated using the GUI as described on page 407, or using the `scatters` command.

```
setinfo allords --graph-name="Australia: All Ordinaries"
setinfo nasdaq  --graph-name="United States: Nasdaq"
setinfo ftse    --graph-name="United Kingdom: FTSE"
setinfo nikkei  --graph-name="Japan: Nikkei"
scatters nasdaq allords ftse nikkei --output=display
```

This yields Figure 14.1. It is pretty clear that there are periods of low and high volatility in these series.

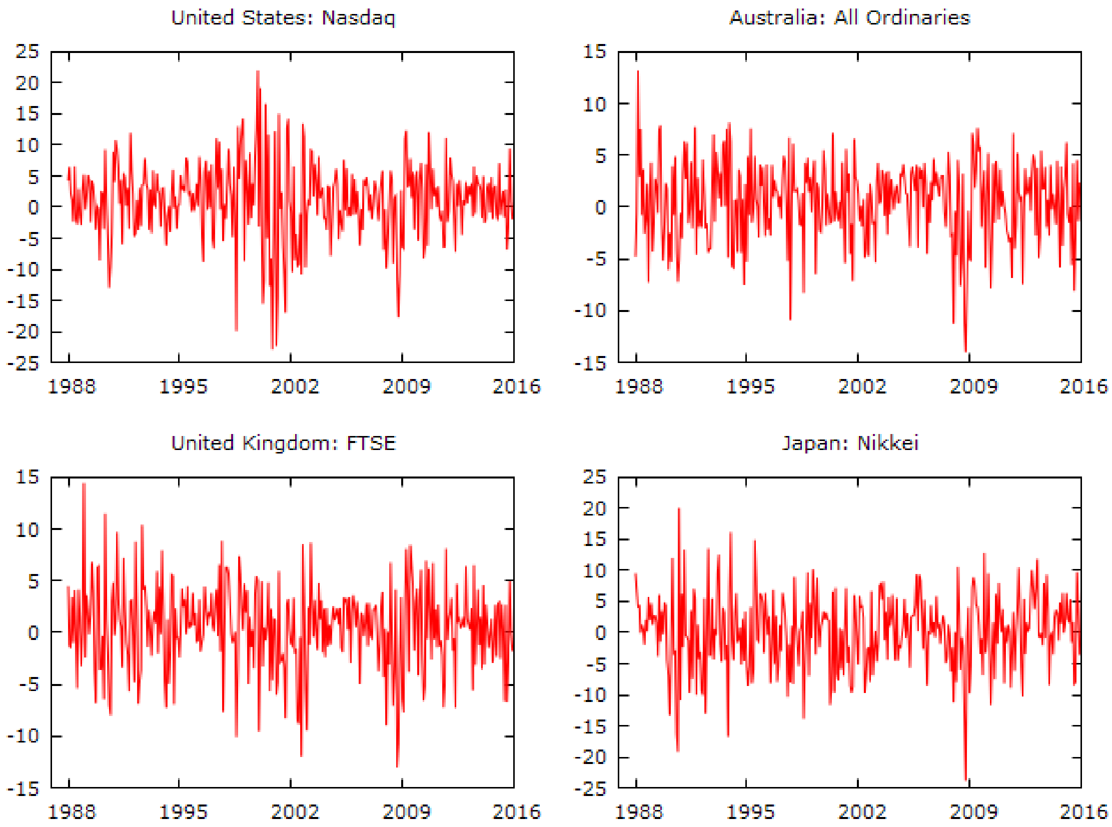


Figure 14.1: Times series of stock indices

Next, the histograms are plotted using the `freq` command. The output is sent to *.png* bitmap graphics files and later combined using the commercial software, Snagit.

```
freq nasdaq  --normal --plot=f1.png
freq allords --normal --plot=f2.png
freq ftse    --normal --plot=f3.png
freq nikkei  --normal --plot=f4.png
```

These plots appear in Figure 14.2.

Relative to the normal distribution, the series tend to have more observations around the mean and in the extremes of the tails. This is referred to as being **leptokurtic**.

Summary statistics found below confirm this.

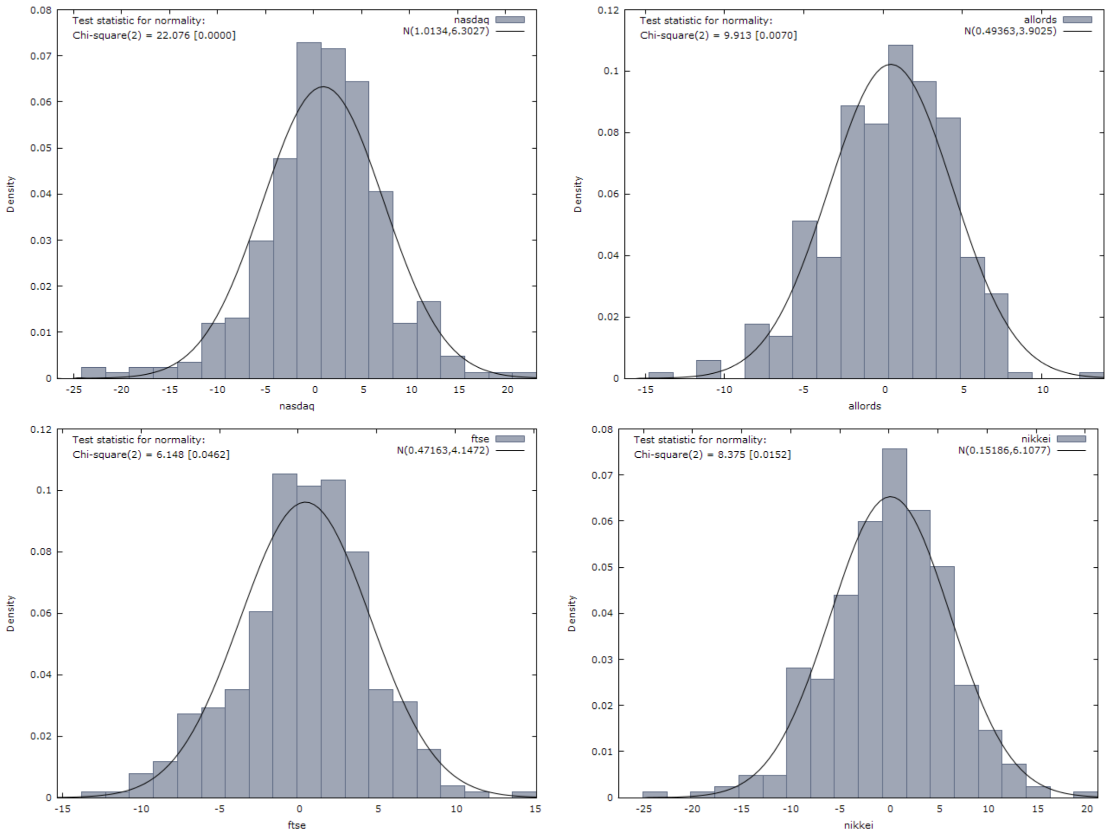


Figure 14.2: Histograms of stock indices.

```
list indices = nasdaq allords ftse nikkei
summary indices
```

```
                Std. Dev.         C.V.       Skewness   Ex. kurtosis
nasdaq             6.3027       6.2196       -0.42162         1.5034
allords            3.9025       7.9057       -0.42852        0.44463
ftse               4.1472       8.7934       -0.26929        0.51427
nikkei             6.1077       40.220       -0.28338        0.69405
```

The skewness of each is negative, with the Nasdaq and Allords being more skewed than the others. The Nasdaq also is the most leptokurtotic ($1.50 > 0$). Recall that the excess kurtosis is measured as $\hat{\mu}_4 - 3$ so positive numbers imply that the series are leptokurtotic.

## 14.1 ARCH and GARCH

The ARCH(1) model can be expressed as:

$$
\begin{aligned}
y_t &= \beta + e_t && (14.1)\\
e_t|I_{t-1} &\sim N(0, h_t) && (14.2)\\
h_t &= \alpha_0 + \alpha_1 e_{t-1}^2 && (14.3)\\
\alpha_0 > 0,&\ 0 \le \alpha_1 < 1
\end{aligned}
$$

The first equation describes the behavior of the mean of the time series. In this case, equation (14.1) indicates that the time series varies randomly about its mean, $\beta$. If the mean of the time series drifts over time or is explained by other variables, add these elements to the 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 available from 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 $\alpha$ 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 $h_t = \alpha_0 + \alpha_1 e_{t-1}^2 + \alpha_2 e_{t-2}^2$. When a regression model has ARCH errors you must specify this order.

ARCH is treated as a special case of a more general model 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:

$$
\begin{aligned}
y_t &= \beta + e_t\\
e_t|I_{t-1} &\sim N(0, h_t)\\
h_t &= \delta + \alpha_1 e_{t-1}^2 + \beta_1 h_{t-1} && (14.4)
\end{aligned}
$$

The difference between ARCH (14.3) and its generalization (14.4) is the term $\beta_1 h_{t-1}$, a function of the lagged variance. In higher order GARCH($p$, $q$) models, $q$ refers to the number of lags of $e_t$ and $p$ refers to the number of lags of $h_t$ to include in the regression's variance.

### Example 14.2 in *POE5*

In this example two simulated samples are created and compared. One has constant variance, $h_t = 1$. The second has ARCH(1) errors with $h_t = \alpha_0 + \alpha_1 e_{t-1}^2 = 1 + 0.8e_{t-1}^2$.

Gretl generates these series easily as shown below.

```
1  nulldata 200
2  setobs 1 1 --special-time-series
3
4  set seed 1010198
5  series e_arch = 0
6  series e = normal(0,1)
7  series e_arch= e*sqrt(1 + .8*(e_arch(-1))^2)
8
9  series y = e
10 series y_arch = e_arch
```

The `nulldata` command creates an empty dataset with the given number (200) observations. These are set as time series using `setobs`. A `seed` for the pseudo-random number generator is given and a series of zeros is created to initialize the contents of the ARCH errors.

The errors for the constant variance errors are generated in line 6 and the ARCH(1) errors with the desired parameters in line 7. Notice that both are constructed using the same random draw from the normal, `e`. Since the ARCH for the simulation has zero mean, the `y` variables are simply equal to their respective errors.

Scatter plots of the two series are created and the plots appear in Figure 14.3. Histograms (Figure 14.4) are also generated using the `freq` command.

```
f5 <- freq y --normal --plot=display
f6 <- freq y_arch --normal --plot=display
```

At 5%, the constant variance series is not significantly non-normal, but the ARCH series is. Also the ARCH series appears to be leptokurtotic.

## 14.2 Testing for ARCH

### Example 14.3 in *POE5*

This example uses *byd.gdt* data on a hypothetical company called Brighten Your Day Lighting. There are 500 observations. These are loaded and the `setobs` command is used to structure them as time series.

First plot the returns and their histograms. Then examine the summary statistics for evidence of leptokurtosis.

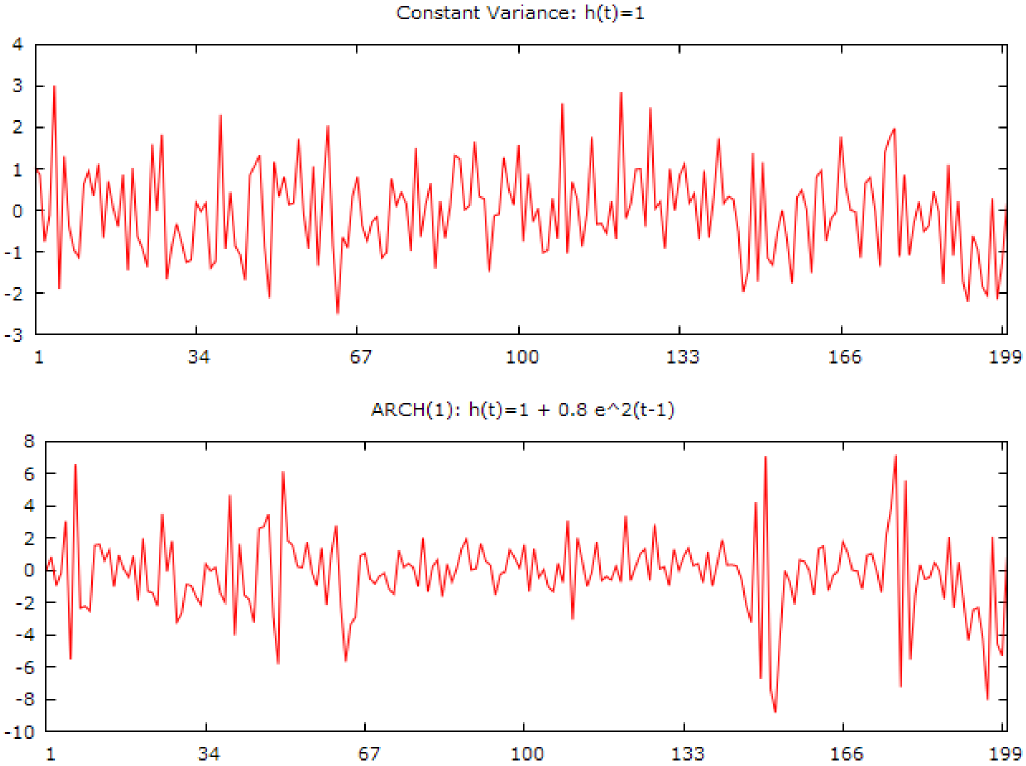


Figure 14.3: Simulated examples of constant and time-varying variance.

```
open "@workdir\data\byd.gdt"
setobs 1 1 --special-time-series

gnuplot r time --output=display --with-lines
freq r --normal --plot=display
summary r
```

The --special-time-series switch which identifies the series as being sequential only. It is a catch-all used when observations occur regularly in time, but not in standard frequencies like weeks, months, or years. The first number identifies the time period of the first observation and the second is the periodicity of the data. Our set starts at 1 and increments by 1 from there.

The series plot appears in Figure 14.5. It certainly appears that the variance is evolving over time.

The histogram of BYD returns is shown in Figure 14.6. Relative to the normal distribution, there appears to be more observations than expected around the mean of 1 as well as a few very large outliers in the right tail. The summary statistics confirm this:

```
Summary statistics, using the observations 1 - 500
```

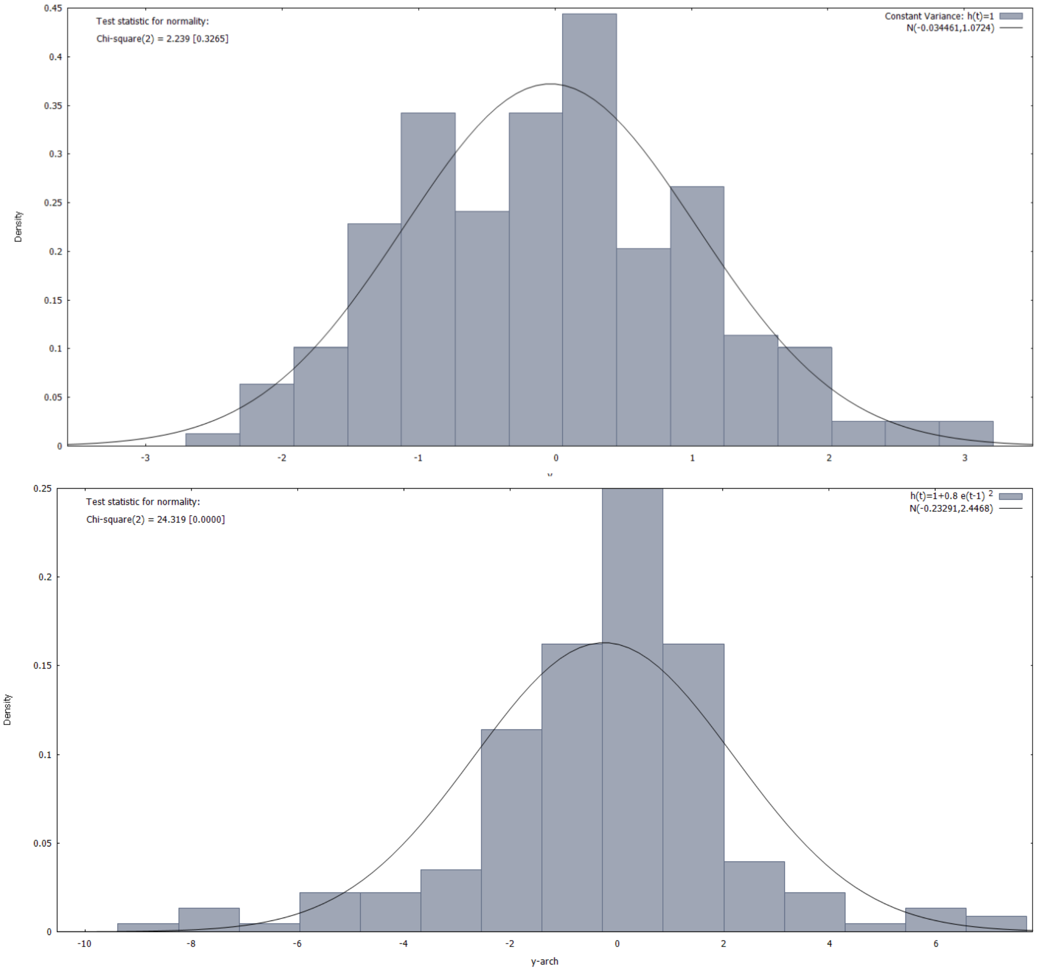


Figure 14.4: Histograms for simulated examples of constant and time-varying variance.

```
for the variable 'r' (500 valid observations)

  Mean                            1.0783
  Median                          1.0293
  Minimum                        -2.7686
  Maximum                         7.0089
  Standard deviation              1.1850
  C.V.                            1.0990
  Skewness                       0.40117
  Ex. kurtosis                    1.4701
  5% percentile                 -0.71215
  95% percentile                  3.2728
  Interquartile range             1.3941
  Missing obs.                         0
```

The data are positively skewed and have substantial excess kurtosis.

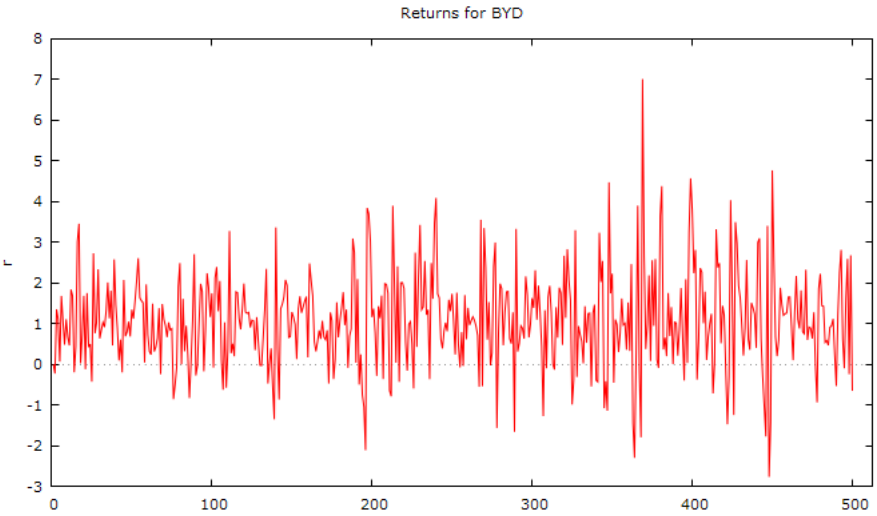
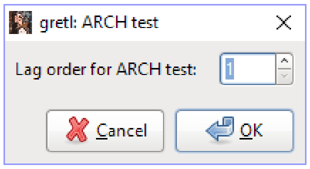


Figure 14.5: Returns for BYD Lighting

Testing for the presence of ARCH in the errors of a 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 from the GUI. Then choose the **Tests>ARCH** from the model's pull-down menu. This opens a dialog box that allows you to enter the desired number of ARCH lags to include in the alternative hypothesis.



Lag order for ARCH test: 1

Choose a lag order of 1 and click OK to produce:

```
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
```

The *LM* statistic is 62.1595 and its *p*-value is well below 5%. We conclude that the model's errors

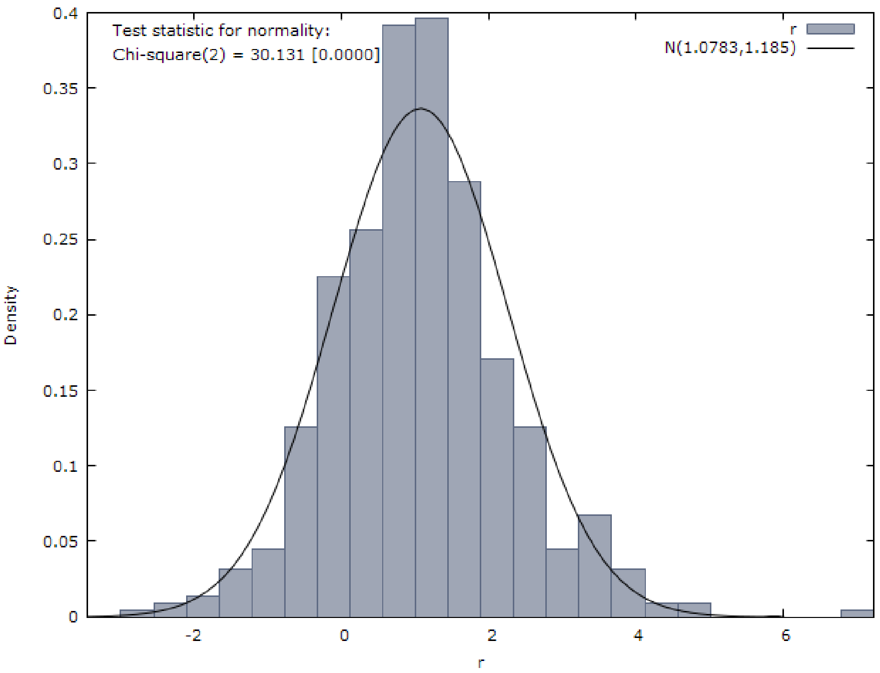


Figure 14.6: Returns for BYD Lighting

exhibit ARCH(1).

This test can be executed from a script using the `modtest` command. The test for ARCH(q) is

```
modtest q --arch
```

where `q` is the number of lags for $H_1$. This yields the same output as obtained using the GUI.

Manually, the first step is to estimate the regression

$$r_t = \beta + e_t \tag{14.5}$$

using least squares. The residuals are saved in `ehat`, and the squared residuals saved as `ehat2`. Next, estimate the regression

$$\hat{e}_t = \alpha_1 + \alpha_2 \hat{e}_{t-1} + u_t \tag{14.6}$$

Take $TR^2$ from least squares estimation of this regression as your test statistic, which has a $\chi^2(1)$ if the errors do not have ARCH.

The script to carry this out manually is straightforward. Estimate the model, save the squared residuals to a series and then regress these on their lagged value and a constant.

```
ols r const --quiet
series ehat2 = $uhat^2
arch_test <- ols ehat2 const ehat2(-1)
printf "LM = %.4f with p-value = %.3f\n", $trsq, pvalue(X,1,$trsq)
```

Recall that `ehat2(-1)` lags `ehat2` by one period. Everything is combined in the final line which prints the statistic and its $p$-value from the $\chi^2(1)$ distribution.

The result:

```
LM = 62.1595 with p-value = 0.000
```

which matches the results from `modtest` exactly.

### Example 14.4 in *POE5*

Estimating ARCH models is relatively straightforward in **gretl**. Once the data are loaded open the dialog for estimating ARCH or GARCH in **gretl** using **Model>Time series>GARCH** from the main **gretl** window.[1] This reveals a dialog box where the model is specified (Figure 14.7). To estimate the ARCH(1) model, place the time-series `r` into the dependent variable box and set `q=1` and `p=0`.

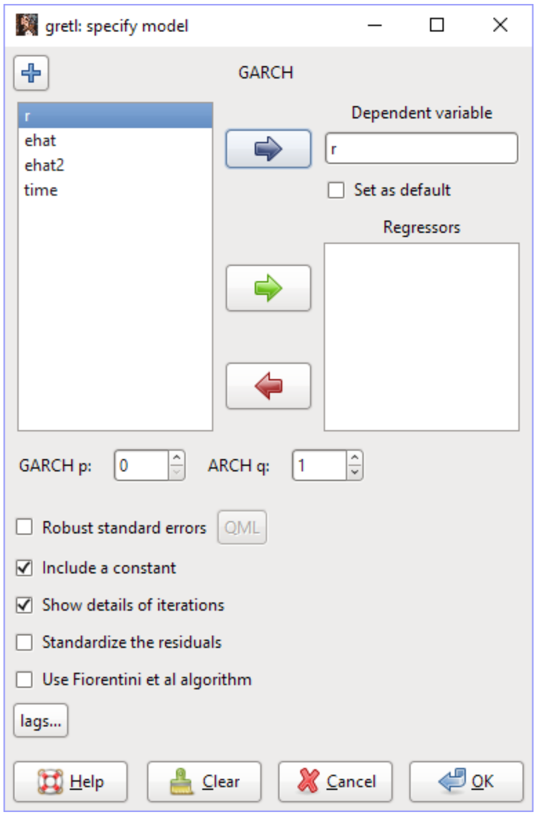
ARCH(1), using observations 1–500
Dependent variable: r
Standard errors based on Hessian

| | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | 1.06394 | 0.0399241 | 26.65 | 0.0000 |
| $\alpha_0$ | 0.642139 | 0.0648195 | 9.907 | 0.0000 |
| $\alpha_1$ | 0.569347 | 0.0913142 | 6.235 | 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] **gretl** also contains a simpler ARCH option. 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.



Dependent variable: r

□ Set as default

GARCH p: 0    ARCH q: 1

□ Robust standard errors

☑ Include a constant

☑ Show details of iterations

□ Standardize the residuals

□ Use Fiorentini et al algorithm

Figure 14.7: Estimating ARCH using the dialog box in **gretl** .

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 *POE5*. To get closer to these, change the default variance-covariance computation using `set garch_vcv op` before running the script. Although this gets you closer, using the `set garch_vcv op` is not usually recommended. To restore the **gretl** default, use `set garch_vcv unset`.

In fact, **gretl** gives you other methods for estimating the variance-covariance matrix. And, as expected, the choice yields different standard errors and $t$-ratios. The `set garch_vcv` command allows you to choose among five alternatives: `unset`–which restores the default, `hessian` (the default), `im` (information matrix), `op` (outer product of gradient matrix), `qml` (QML estimator), or `bw` (Bollerslev-Wooldridge). If the `--robust option` is given for the `garch` command, QML is used.

With maximum likelihood, the model's parameters are estimated using numerical optimization techniques. All of the techniques should produce 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 numerical procedure. Once the algorithm finds the maximum of the function, the curvature measure is often 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.

```
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)
Examples:   garch 1 1 ; y
            garch 1 1 ; y 0 x1 x2 --robust
```

The series are characterized by random, rapid changes and are considered **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.

### Example 14.5 in *POE5*

Once the model is estimated, the behavior of the variance, $h_t$, can be plotted. The forecasted variances are stored in memory and accessed using the accessor, `$h`. Then plot them using **gnuplot** :

```
m2_arch <- garch 0 1; r const
series ht = $h
g_arch <- gnuplot ht time --output=display --with-lines
```

After a little editing, the result is shown in Figure 14.8. To modify the graph, right-click on the graph and choose **Edit**. You can add labels, change the colors or line style, add titles, and more.

## 14.3 GARCH

### Example 14.6 in *POE5*

A GARCH(1,1) model as shown in equation (14.4) includes a lag of the variance in the model of $h_t$. Its parameter is labeled $\beta_1$. In this example a single GARCH term is added and the model estimated via maximum likelihood. The estimated average return and variance are plotted.

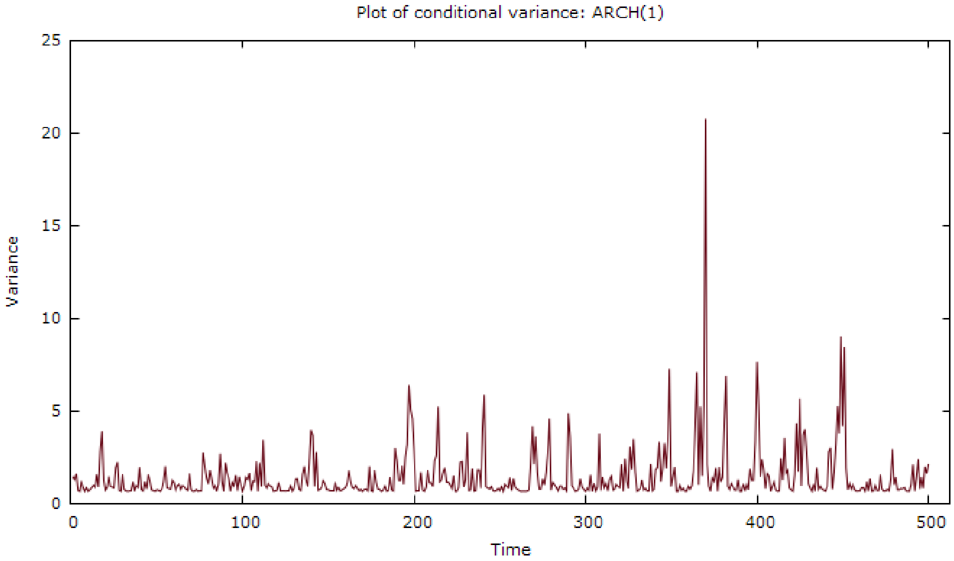


Figure 14.8: Plot of the variances after estimating the ARCH(1) using the BrightenYourDay returns. Right click on the graph to bring up the menu shown. Then choose `edit` to modify the graph.

The script used is:

```
GARCH_11 <- garch 1 1 ; r const
series yhat = $yhat
series ht = $h

gnuplot ht time --with-lines --output=display
gnuplot yhat time --with-lines --output=display
```

The estimated GARCH model is:

$$\widehat{\mathrm{r}} = \underset{(0.03950)}{1.050}$$

$$\hat{\sigma}^2_t = \underset{(0.08438)}{0.40105} + \underset{(0.08589)}{0.4910}\ \varepsilon^2_{t-1} + \underset{(0.09046)}{0.2380}\ \sigma^2_{t-1}$$

$$T = 500 \quad \ln L = -736.0281 \quad \hat{\sigma} = 1.2166$$

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

These results are very close to the ones from *POE5*.

Although the plain **gnuplot** plots are functional, I took a little extra time to dress it up a little and to combine predicted mean and variance into a single plot.

```
list vars = Return Variance
string title = "GARCH(1,1)"
string xname = "Time"
string yname = "Return"
string y2name = "Variance"
g3 <- plot vars
    options with-lines time-series
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
    printf "set y2label \"%s\"", y2name
end plot --output=display
```

This effort yielded Figure 14.9: The average return is constant and equal to $\hat{\beta} = 1.05$; the graph scale for this is on the left y-axis. The variance is measured on the right and plotted in blue. There is an obvious large increase in variance between observations 350 and 400. This is characteristic of GARCH and ARCH.

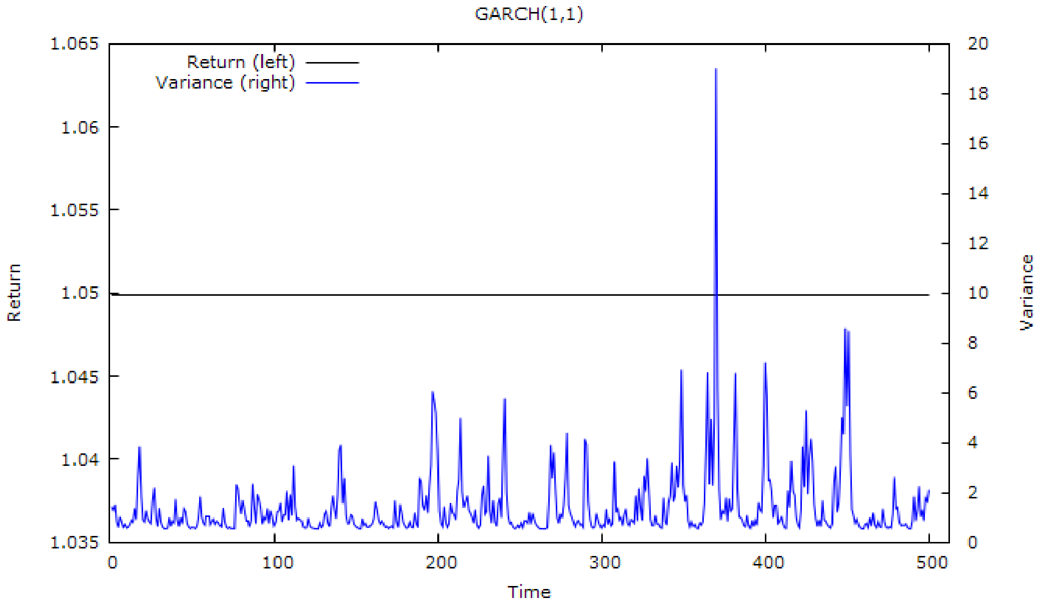


Figure 14.9: Predicted returns and variance from a GARCH(1,1)

## 14.4 Threshold ARCH

### Example 14.7 un *POE5*

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. (2018).

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, a log-likelihood function must be specified. Also, 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 (e.g., the log-likelihood) to iteratively find the minimum or maximum of the function. If desired, the analytical derivatives of the log-likelihood function with respect to each of the parameters can be specified; if analytical derivatives are not supplied, **gretl** tries to compute a numerical approximation. The actual results 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 (or copy and paste) in the program that appears in Figure 14.10.

Lines 4-8 of the script give starting values for the model's parameters. This is essential and picking good starting values increases the chances of success. At the very least, you must start the numerical optimization at a feasible point. For instance, you cannot start the model with a negative variance.

The second part of the script, starting on line 10, contains the algebraic expression of the log-likelihood function. Line 10 `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.

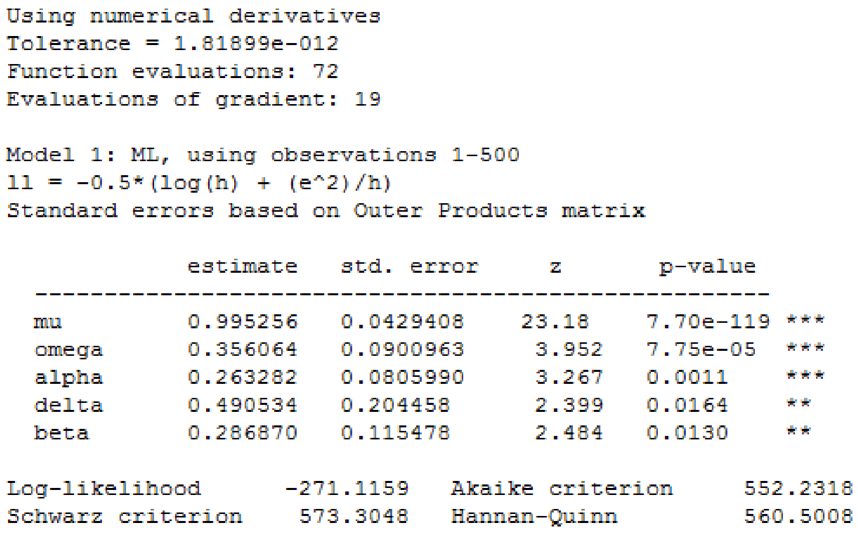
Next, we have to specify an initial guess for the variances of the model, and these are set to the empirical variance of the series using `var(r)`. Then, the errors are generated, squared, and the threshold term is created using `series e2m = e2 * (e<0)`; the expression `(e<0)` takes the value of 1 for negative errors, `e`, and is zero otherwise. Then in lines 15 and 16, which use the line continuation command, 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 14.11. The coefficient estimates are very close to those

```
open "@workdir\data\byd.gdt"
setobs 1 1 --special-time-series

scalar mu = 0.5          # Starting values
scalar omega = .5
scalar alpha = 0.4
scalar delta = 0.1
scalar beta = 0

mle ll = -0.5*(log(h) + (e^2)/h)     # Log-likelihood function
   series h = var(r)                 # Initialization of variances
   series e = r - mu                 # Model's residuals
   series e2 = e^2                   # Squared residuals
   series e2m = e2 * (e<0)           # Create the threshold
   series h = omega + alpha*e2(-1)\
       + delta*e2m(-1) + beta*h(-1) # TARCH variance
   params mu omega alpha delta beta # Parameters
end mle
```

Figure 14.10: Threshold GARCH script

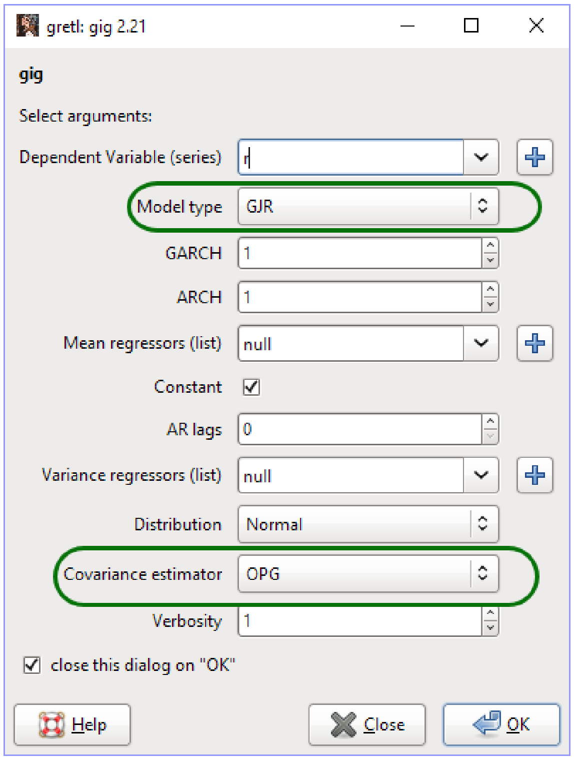
printed in *POE5*, and the standard errors and corresponding $t$-ratios are similar. As discussed above it is common for estimates produced by different software to produce this kind of variation when estimating nonlinear models numerically. Different software may use different algorithms to numerically maximize the log-likelihood function. Most will offer some options that may get you closer. What is amazing here is that **gretl** does such a fine job without having to specify the analytic derivatives of the log-likelihood. It is 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:



```
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 14.11: Threshold ARCH results



gretl: gig 2.21

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 equivalent to the TARCH discussed above. Estimating it with the OPG[2] covariance estimator yields very similar results to the ones in *POE5*. This module offers several other variants of GARCH, but you will have to use the **gretl** documentation to be sure of what you

[2] OPG stands for **o**uter **p**roduct of the **g**radient.

are estimating. For instance, the TARCH option does not estimate the TARCH model specified in *POE5*. It pays to examine the actual algebraic form of the model being estimated, since different authors use different terms to abbreviate the model's name or purpose. There are also different parameterizations use by different authors that further obscure things for the user.

```
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
```

In point of fact, notice that in *gig 2.22* $\alpha$ and $\gamma$ refer to two different ways to parameterize the model. The alternative parameterization is the one discussed in *POE5*.

The *gig* package will produce graphs as well. Below in Figure 14.12 is a plot of the unanticipated return and estimated standard error based on TARCH estimates. The return in this model is constant and equal to .9954, so a negative residual means that actual returns fell below average. This is what I'm referring to as *unanticipated*, which is probably a misnomer but conventional in econometric practice.

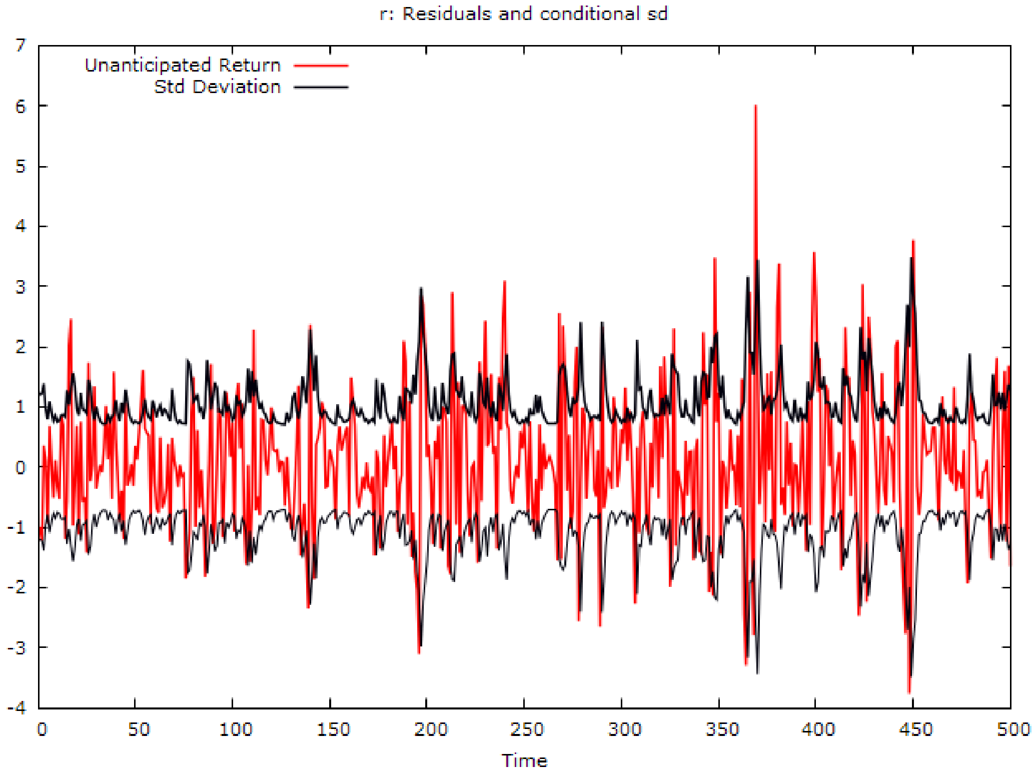


Figure 14.12: Plot produced by gig for TARCH residuals and $\pm\sqrt{\hat{h}_t}$

## 14.5 Garch-in-Mean

### Example 14.8 in *POE5*

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 the 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 the threshold term remains in the model. The errors are normally distributed with zero mean and variance $h_t$.

The parameters of the 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 14.13) notice that a function is used to populate the log-likelihood.[3] The function is called `gim_filter`, it returns a matrix when called, 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, initiate a series for the variances and the errors; these have been called `lh` and `le`, respectively. The log-likelihood function 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]
endloop
```

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 `endloop` statement tells **gretl** the point at which to return to the top of the loop and advance the increment i. As usual, the `--quiet` option reduces the amount of output that is written to the screen.

[3] Actually, the is a very slightly modified version of one provided by **gretl** genius Professor 'Jack' Lucchetti, whose help is very much appreciated!

Within the loop itself, lag the index and create an indicator variable that takes the value of 1 when the news is bad ($e_{t-1} < 0$) and is zero otherwise. 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. Both are combined and place into a matrix that is returned when the function is called. The function is closed using end function.

If this looks too complicated, 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, initialize each parameter just as done in TGARCH. The series that will eventually hold the variances also must be initialized. The latter is done using series h, which creates the empty series h. 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. Since E returns both residuals and variances, pull the residuals out and place them into a series as in line 27. This series is used by the normal kernel in line 25. 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 below.

```
Tolerance = 1.81899e-012

Function evaluations: 82
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         0.814459   0.0677497    12.02    2.74e-033 ***
  theta      0.200802   0.0610726     3.288   0.0010    ***
  delta      0.370791   0.0658589     5.630   1.80e-08  ***
  alpha      0.296681   0.0735687     4.033   5.51e-05  ***
  gam        0.313665   0.128547      2.440   0.0147    **
  beta       0.279001   0.0544579     5.123   3.00e-07  ***

Log-likelihood      724.4610   Akaike criterion     1460.922
Schwarz criterion    1486.210   Hannan-Quinn         1470.845
```

This is a difficult likelihood to maximize and **gretl** may take a few seconds to compute the estimates. A better set of starting values will reduce the number of warnings that the script throws

```
function matrix gim_filter(series y, \
      scalar mu, scalar theta, scalar delta, scalar alpha, \
      scalar gam, scalar beta, series *h)

    series lh = var(y)               # initialize the variance series
    series le = y - mu               # initialize the residual series
    scalar T = $nobs                 # Number of Observations
    loop for i=2..T --quiet
        scalar ilag = $i - 1
        scalar d = (le[ilag]<0)      # Create the negative threshold
        scalar e2lag = le[ilag]^2    # Square the residual
        lh[i] = delta + alpha*e2lag + gam*e2lag*d + beta*lh[ilag] # ht
        le[i] = le[i] - theta*lh[i]  # residual
    endloop

scalar mu = 0.8               # set starting values for parameters
scalar theta = 0.1
scalar delta = .5
scalar alpha = 0.4
scalar gam = .1
scalar beta = 0

series h                      # initialize the series h

mle ll = -0.5*(log(2*pi) + log(h) + (e^2)/h)
    matrix E = gim_filter(r, mu, theta, delta, alpha, gam, beta, &h)
    series e = E[,1]          # First column of E contains residuals
   params mu theta delta alpha gam beta
end mle --robust
```

Figure 14.13: The MGARCH `mle` script includes a function to computes the residuals and variances for the log-likelihood.

off the first time it runs. 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.

The `gim_filter` function was written so that it returns both the estimated residuals and variances for the model. This allows plotting as shown in Figure 14.14.

To generate this plot I created series from the matrix return of `gim_filter` and added descriptions that can be used by plot. The series are put into a list and titles and labels added.

```
series Residual = E[,1]
series Variance = E[,2]
setinfo Residual -d "Unanticipated Return" -n "Unanticipated Return"
setinfo Variance -d "GARCH-in-mean variance" -n "GARCH-in-mean variance"
list vars = Residual Variance
```

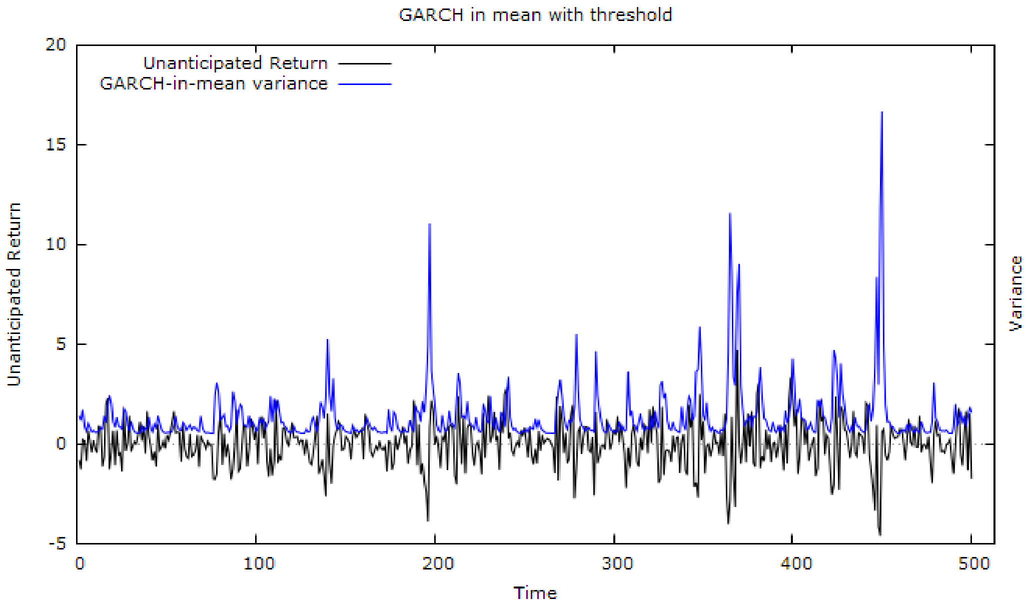


Figure 14.14: Estimated mean and variances of GARCH-in-mean that includes a threshold.

```
string title = "GARCH in mean with threshold"
string xname = "Time"
string yname = "Unanticipated Return"
string y2name = "Variance"
g3 <- plot vars
    options with-lines time-series
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
    printf "set y2label \"%s\"", y2name
end plot --output=display
```

The plot produced is quite interesting. A negative residual suggests an unanticipated negative return. Thus, from the plot we see that when the unexpected return is negative, this coincides or precedes slightly a high level of volatility.

Summary statistics reveal that the average return is negative, but the median return is positive. A few large negative returns are impacting the average performance of this security.

```
                  Mean        Median       Minimum       Maximum
Residual     -0.011339      0.026203       -4.6153        4.7090
Variance        1.3732       0.88071       0.51772        16.672

             Std. Dev.          C.V.      Skewness   Ex. kurtosis
Residual        1.1582        102.15      -0.23591        1.3303
```

```
Variance           1.5066         1.0972         5.0876         35.330

                 5% perc.       95% perc.       IQ range   Missing obs.
Residual          -1.8748         1.8701         1.3797              0
Variance          0.54209         3.6103        0.76726              0
```

Also, the returns are negatively skewed and leptokurtotic.

## 14.6 Script

```
open "@workdir\data\returns5.gdt"
set verbose off

# Example 14.1
setinfo allords --graph-name="Australia: All Ordinaries"
setinfo nasdaq --graph-name="United States: Nasdaq"
setinfo ftse --graph-name="United Kingdom: FTSE"
setinfo nikkei --graph-name="Japan: Nikkei"
scatters nasdaq allords ftse nikkei --output=display

freq nasdaq  --normal --plot=f1.png
freq allords --normal --plot=f2.png
freq ftse    --normal --plot=f3.png
freq nikkei  --normal --plot=f4.png

list indices = nasdaq allords ftse nikkei
summary indices

# Example 14.2
# Simulating returns
nulldata 200
setobs 1 1 --special-time-series

set seed 1010198
series e_arch = 0
series e = normal(0,1)
series e_arch= e*sqrt(1 + .8*(e_arch(-1))^2)

series y = e
series y_arch = e_arch
setinfo y --graph-name="Constant Variance: h(t)=1"
setinfo y_arch --graph-name="ARCH(1): h(t)=1 + 0.8 e^2(t-1) "
scatters y y_arch --output=display --with-lines

f5 <- freq y --normal --plot=display
f6 <- freq y_arch --normal --plot=display

```

```
garch 0 1; y
garch 0 1; y_arch

# Example 14.3
open "@workdir\data\byd.gdt"
setobs 1 1 --special-time-series

# Initial Plots and summary statistics
gnuplot r time --output=display --with-lines
freq r --normal --plot=display
summary r

# arch(1) LM test
m_ols <- ols r const
modtest 1 --arch

# LM test manually
ols r const --quiet
series ehat2 = $uhat^2
arch_test <- ols ehat2 const ehat2(-1)
printf "LM = %.4f with p-value = %.3f\n", $trsq, pvalue(X,1,$trsq)

# Example 14.4
# arch(1) Using built in command for arch
m1_arch <- arch 1 r const
# garch(0,1)=arch(1)
# set garch_vcv op          # OPG vcv: Uncomment to match POE5
set garch_vcv unset         # Resets the vcv to default
GARCH_11 <- garch 1 1 ; r const

# Example 14.5
m2_arch <- garch 0 1; r const
series ht = $h
g_arch <- gnuplot ht time --output=display --with-lines

# Example 14.6
garch 1 1 ; r const
series Return = $yhat
series Variance = $h
list vars = Return Variance
string title = "GARCH(1,1)"
string xname = "Time"
string yname = "Return"
string y2name = "Variance"
g3 <- plot vars
    options with-lines time-series
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
    printf "set y2label \"%s\"", y2name
```

```
end plot --output=display

gnuplot Variance time --with-lines --output=display
gnuplot Return time --with-lines --output=display

# threshold arch
open "@workdir\data\byd.gdt"
setobs 1 1 --special-time-series

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)    # Log-likelihood function
   series h = var(r)                # Initialization of variances
   series e = r - mu                # Model's residuals
   series e2 = e^2                  # Squared residuals
   series e2m = e2 * (e<0)          # Create the threshold
   series h = omega + alpha*e2(-1)\
       + delta*e2m(-1) + beta*h(-1) # TARCH variance
   params mu omega alpha delta beta # Parameters
end mle

# GIG GJR(1,1)  You may have to initialize this through the GUI
include gig.gfn
d=GUI_gig(r, 3, 1, 1, null, 1, 0, null, 0, 0, 1)   # contents to bundle

# garch-in-mean
open "@workdir\data\byd.gdt"
setobs 1 1 --special-time-series

# garch-in-mean -- means and variances

function matrix gim_filter(series y, \
                           scalar mu, scalar theta, scalar delta, scalar alpha, \
                           scalar gam, scalar beta, series *h)

    series lh = var(y)               # initialize the variance series
    series le = y - mu               # initialize the residual series
    scalar T = $nobs                 # Number of Observations
    loop for i=2..T --quiet
        scalar ilag = $i - 1
        scalar d = (le[ilag]<0)      # Create the negative threshold
        scalar e2lag = le[ilag]^2    # Square the residual
        lh[i] = delta + alpha*e2lag + gam*e2lag*d + beta*lh[ilag] # ht
        le[i] = le[i] - theta*lh[i]  # residual
    endloop

    series h = lh                    # Puts ht into series h (pointer in function)
    matrix matvar = { le, h}         # The  matrix return
    return matvar
end function


scalar mu = 0.8
scalar theta = 0.1
scalar delta = .5
scalar alpha = 0.4
scalar gam = .1
scalar beta = 0

series h

mle ll = -0.5*(log(2*$pi) + log(h) + (e^2)/h)
    matrix E = gim_filter(r, mu, theta, delta, alpha, gam, beta, &h)
    series e = E[,1]
   params mu theta delta alpha gam beta
end mle --robust

series Residual = E[,1]
series Variance = E[,2]
setinfo Residual -d "Unanticipated Return" -n "Unanticipated Return"
setinfo Variance -d "GARCH-in-mean variance" -n "GARCH-in-mean variance"
list vars = Residual Variance
string title = "GARCH in mean with threshold"
string xname = "Time"
string yname = "Unanticipated Return"
string y2name = "Variance"
g3 <- plot vars
    options with-lines time-series
    literal set linetype 1 lc rgb "black" pt 7
    printf "set title \"%s\"", title
    printf "set xlabel \"%s\"", xname
    printf "set ylabel \"%s\"", yname
    printf "set y2label \"%s\"", y2name
end plot --output=display

summary vars
```

# Chapter 15

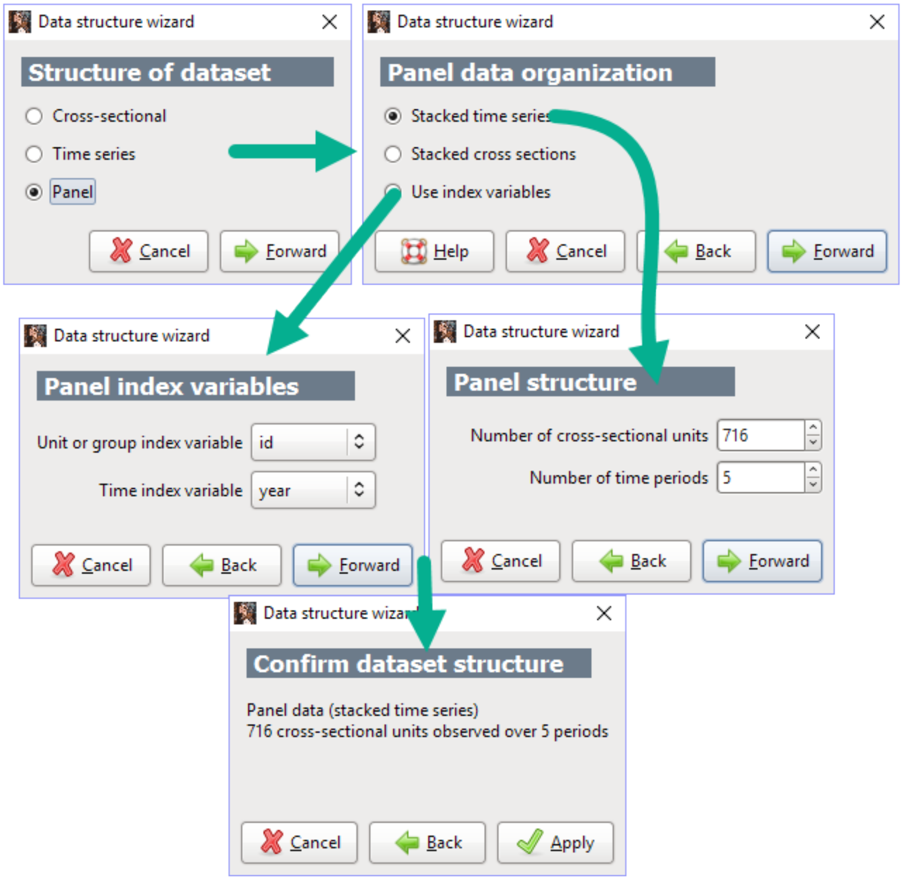
# Pooling Time-Series and Cross-Sectional Data

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. (2018) 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 must be 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 in Figure 15.1. To use this method, the data must 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 use unstructured data you'll want to assign the proper dataset structure to it so that all of the appropriate panel data procedures are available for use.

Gretl provides 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 attainment. These data can be installed using **File>Databases>On database server** from the menu bar as shown in Figure 15.2 below. From here, select a database you want. In Figure 15.3 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 3 icons at the top of the window

[1] Your computer must have access to the internet to use this.



Data structure wizard

**Structure of dataset**

○ Cross-sectional  
○ Time series  
● Panel

Data structure wizard

**Panel data organization**

● Stacked time series  
○ Stacked cross sections  
○ Use index variables

Data structure wizard

**Panel index variables**

Unit or group index variable: id  
Time index variable: year

Data structure wizard

**Panel structure**

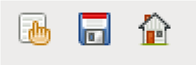
Number of cross-sectional units: 716  
Number of time periods: 5

Data structure wizard

**Confirm dataset structure**

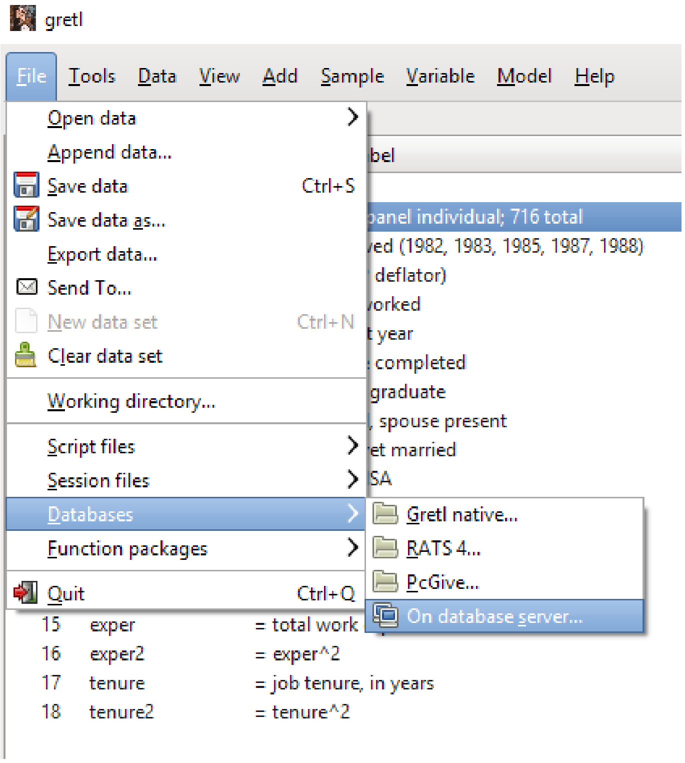
Panel data (stacked time series)  
716 cross-sectional units observed over 5 periods

Figure 15.1: The data structure wizard for panel data. If the dataset contains variables that identify time and individual, the best choice is to select *Use index variables*. Otherwise, select *Stacked time series* or *Stacked cross sections*, depending on how the data are arranged in the datafile.



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 15.4. 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.

Figure 15.4 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.



```
15   exper      = total work
16   exper2     = exper^2
17   tenure     = job tenure, in years
18   tenure2    = tenure^2
```

Figure 15.2: Accessing 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.

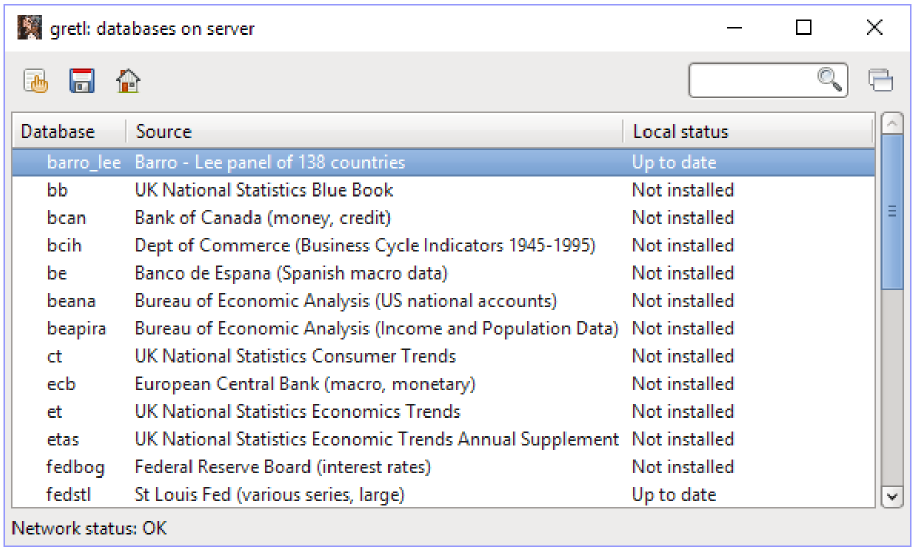
$$y_{it} = \beta_{1it} + \beta_{2it} x_{2it} + \beta_{3it} x_{3it} + e_{it} \tag{15.1}$$

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 balanced, the model contains 3 times as many parameters as observations (nT)! To be able to estimate the model, some assumptions must 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).

$$y_{it} = \beta_{1i} + \beta_2 x_{2it} + \beta_3 x_{3it} + e_{it} \tag{15.2}$$

This specification, which includes $n+2$ parameters, includes dummy variables that allow a separate intercept for each individual. Such a model implies that there are no substantive changes in the



gretl: databases on server

| Database | Source | Local status |
|---|---|---|
| barro_lee | Barro - Lee panel of 138 countries | Up to date |
| 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 |
| ct | UK National Statistics Consumer Trends | 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 |

Network status: OK

Figure 15.3: Installing a data from the database server via the pull-down menus

regression function over short time periods. Obviously, the longer the 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 the 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.

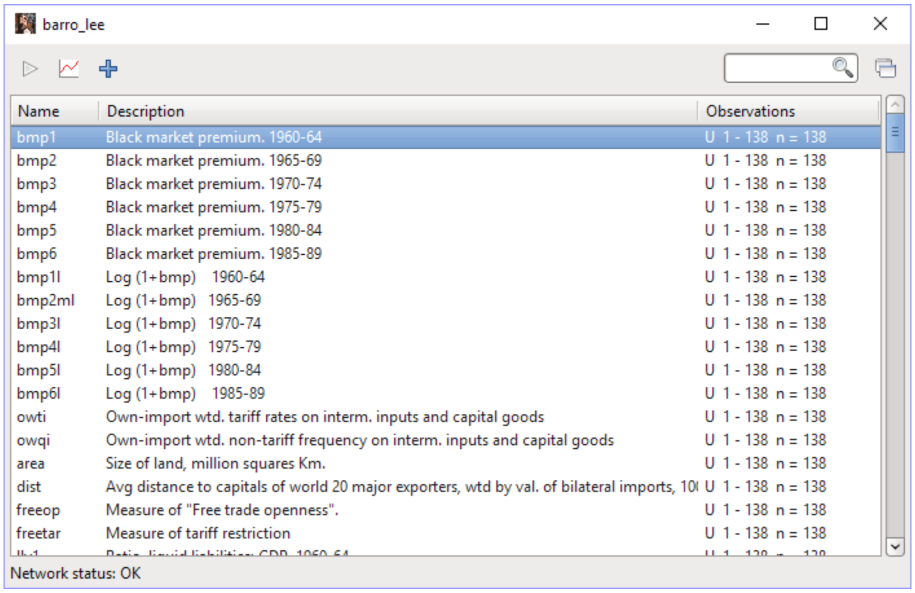
For longer panels where the regression function is shifting over time, $T-1$ time dummy variables can be added to the model. The model becomes

$$y_{it} = \beta_{1i} + \beta_{1t} + \beta_2 x_{2it} + \beta_3 x_{3it} + e_{it} \tag{15.3}$$

where either $\beta_{1i}$ or $\beta_{1t}$ must 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.

### Example 15.1 in *POE5*

This example lists observations on several variables in a microeconometric panel of individuals. The *nls_panel.gdt* dataset Hill et al. (2018) includes a subset of National Longitudinal Survey which is conducted by the U.S. Department of Labor. The database includes observations on women, who



barro_lee

| Name | Description | Observations |
|---|---|---|
| bmp1 | Black market premium. 1960-64 | U 1 - 138 n = 138 |
| bmp2 | Black market premium. 1965-69 | U 1 - 138 n = 138 |
| bmp3 | Black market premium. 1970-74 | U 1 - 138 n = 138 |
| bmp4 | Black market premium. 1975-79 | U 1 - 138 n = 138 |
| bmp5 | Black market premium. 1980-84 | U 1 - 138 n = 138 |
| bmp6 | Black market premium. 1985-89 | U 1 - 138 n = 138 |
| bmp1l | Log (1+bmp) 1960-64 | U 1 - 138 n = 138 |
| bmp2ml | Log (1+bmp) 1965-69 | U 1 - 138 n = 138 |
| bmp3l | Log (1+bmp) 1970-74 | U 1 - 138 n = 138 |
| bmp4l | Log (1+bmp) 1975-79 | U 1 - 138 n = 138 |
| bmp5l | Log (1+bmp) 1980-84 | U 1 - 138 n = 138 |
| bmp6l | Log (1+bmp) 1985-89 | U 1 - 138 n = 138 |
| owti | Own-import wtd. tariff rates on interm. inputs and capital goods | U 1 - 138 n = 138 |
| owqi | Own-import wtd. non-tariff frequency on interm. inputs and capital goods | U 1 - 138 n = 138 |
| area | Size of land, million squares Km. | U 1 - 138 n = 138 |
| dist | Avg distance to capitals of world 20 major exporters, wtd by val. of bilateral imports, 10( | U 1 - 138 n = 138 |
| freeop | Measure of "Free trade openness". | U 1 - 138 n = 138 |
| freetar | Measure of tariff restriction | U 1 - 138 n = 138 |

Network status: OK

Figure 15.4: This shows a portion of the series list window for the Barro and Lee data from the database server. Using the icons at the top of the window a series can be displayed, plotted, or added to the dataset. A right-click of the mouse will also reveal the available choices.

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. The 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.

```
open "@workdir\data\nls_panel.gdt"
list xvars = educ south black union exper exper2 tenure tenure2 const
print id year lwage xvars --byobs
```

This syntax shows that variable names and lists can be used together to produce the desired results. Also, the --byobs option prints the listed series by observation. If not used, the variables print all observations separately. For instance, the first series, id, would simply list the identification number for every observation. Once printed, it then prints every yearly observation and so on.

```
                 id          year         lwage          educ         south

  1:1             1            82      1.808289            12             0
  1:2             1            83      1.863417            12             0
  1:3             1            85      1.789367            12             0
  1:4             1            87      1.846530            12             0
  1:5             1            88      1.856449            12             0
  2:1             2            82      1.280933            17             0
```

```
2:2            2          83    1.515855          17           0
2:3            2          85    1.930170          17           0
2:4            2          87    1.919034          17           0
2:5            2          88    2.200974          17           0
3:1            3          82    1.814825          12           0
3:2            3          83    1.919913          12           0
3:3            3          85    1.958377          12           0
3:4            3          87    2.007068          12           0
3:5            3          88    2.089854          12           0
```

When data are arranged in this way, they are sorted as stacked time series. The first five observations contain all time periods for the first individual.

## 15.2 Estimation

### Example 15.2 in *POE5*

Estimation of models using panel data is relatively straightforward, though there are many variants one can consider. In fact, entire courses are devoted to the possibilities. In this chapter, a few estimators are discussed and data from two sets used in estimation and testing.

The first set of models is based on the presence of fixed effects in the regression function as shown in equation (15.2). When there are only two time periods, the data can be time-differenced and OLS used to estimate the slopes of all time-varying regressors. Time-invariant variables and the intercept drop out of the model upon differencing.

This is illustrated in Example 15.2 in *POE5*. The data are included in the *chemical2.gdt* dataset. This dataset contains sales, capital, and labor inputs for Chinese chemical firms. There are 3 years of observations on 200 firms. The model to be estimated is a log-log model of sales:

$$\ln(sales_{it}) = \beta_{1i} + \beta_2 \ln(capital_{it}) + \beta_3 \ln(labor_{it}) + e_{it}$$

The years considered are $T = 2005, 2006$. Taking the time difference yields:

$$\Delta \ln(sales_{it}) = \beta_2 \Delta \ln(capital_{it}) + \beta_3 \Delta \ln(labor_{it}) + \Delta e_{it}$$

where $\Delta \ln(x_{it}) = \ln(x_{i,2006}) - \ln(x_{i,2005})$, with $x = sales, capital, labor$.

The **gretl** script to estimate this model is:

```
open "@workdir\data\chemical2.gdt"
dummify year
smpl (Dyear_2005 ==1 || Dyear_2006 ==1) --restrict
```

```
list xvars = const lcapital llabor
diff xvars lsales
m1 <- ols d_lsales d_lcapital d_llabor
m2 <- ols lsales xvars
```

This is our second use of `dummify` (see page 247) and, though not strictly needed here, can be useful in creating time fixed effects. This command creates a set of indicator variables for the distinct values of the variables list. In this example, there are three years and three indicators will be created. There are options available for this command that allow one to drop either the first or last of the indicators. The `smpl` command is used to limit observations to the years 2005 and 2006. From there a `list` is created that contains the series and the `diff` command is used to generate the differences. The regression results are:

$$\widehat{\text{d\_lsales}} = \underset{(0.05072)}{0.03837}\,\text{d\_lcapital} + \underset{(0.07547)}{0.3097}\,\text{d\_llabor}$$
$$T = 200 \quad \bar{R}^2 = 0.0759 \quad F(2,198) = 8.6759 \quad \hat{\sigma} = 0.35299$$
$$\text{(standard errors in parentheses)}$$

The pooled model is also estimated which specifies a common intercept and slopes for the two years. This regression produces:

$$\widehat{\text{lsales}} = \underset{(0.2107)}{5.8745} + \underset{(0.03545)}{0.2536}\,\text{lcapital} + \underset{(0.05760)}{0.4264}\,\text{llabor}$$
$$T = 400 \quad \bar{R}^2 = 0.5017 \quad F(2,397) = 201.87 \quad \hat{\sigma} = 0.84831$$
$$\text{(standard errors in parentheses)}$$

The difference in estimates is quite pronounced. The elasticity of sales with respect to capital is 0.38 in the fixed effects model and nearly 10 times larger in the pooled regression.

### Example 15.2 in *POE5*

The difference estimator is used to estimate a simple wage regression based on the nls_panel.gdt data. The model is

$$\ln(wage_{it}) = \beta_{1i} + \beta_2 educ_i + \beta_3 exper_{it} + e_{it}$$

Taking a two period time difference causes the intercept and the time-invariant years of schooling to drop from the model. The script is:

```
open "@workdir\data\nls_panel.gdt"
smpl (year==87 || year==88) --restrict
```

```
diff lwage exper
ols d_lwage d_exper
```

The sample is restricted to 1987 and 1988 and the differenced regression yields:

$$\widehat{\text{d\_lwage}} = \underset{(0.007141)}{0.02184}\ \text{d\_exper}$$
$$T = 716 \quad \bar{R}^2 = 0.0129 \quad F(1,715) = 9.3546 \quad \hat{\sigma} = 0.25332$$
$$\text{(standard errors in parentheses)}$$

This matches the results from *POE5*.

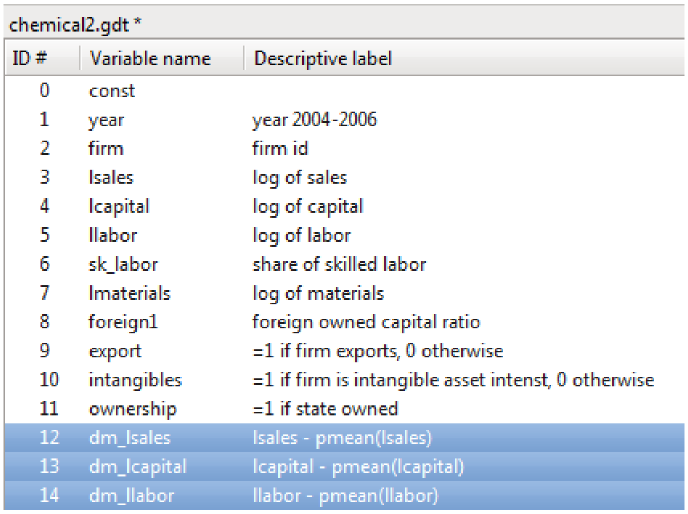
### Example 15.4 in *POE5*

In this example the within transformation is used to estimate a two-period fixed effects model of Chinese chemical firms. The data are loaded, a variable list created and the sample restricted to the years 2005 and 2006. The option `--replace` replaces the existing data and `--permanent` makes the changes permanent.[2]

```
1 open "@workdir\data\chemical2.gdt"
2 list xvar = lsales lcapital llabor
3 smpl year !=2004 --restrict --replace --permanent
4
5 loop foreach j xvar --quiet       # The within transformation
6     series dm_$j = $j - pmean($j)
7 endloop
```

The most interesting feature of this script starts in line 5. A `foreach` loop is initiated that will quietly demean each variable ($j = lsales, lcapital, llabor$) in `xvar`. The `pmean` function is a special panel operator that computes the time mean for each individual in the sample. The values are repeated for each period. The series created will have a `dm_` prefix as shown in variables 12-14 in the list below.

[2] Permanent at least in terms of this session. If the data are not saved to a file in this state, then reopening the dataset will repopulate the entire sample.



chemical2.gdt *

| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | year | year 2004-2006 |
| 2 | firm | firm id |
| 3 | lsales | log of sales |
| 4 | lcapital | log of capital |
| 5 | llabor | log of labor |
| 6 | sk_labor | share of skilled labor |
| 7 | lmaterials | log of materials |
| 8 | foreign1 | foreign owned capital ratio |
| 9 | export | =1 if firm exports, 0 otherwise |
| 10 | intangibles | =1 if firm is intangible asset intenst, 0 otherwise |
| 11 | ownership | =1 if state owned |
| 12 | dm_lsales | lsales - pmean(lsales) |
| 13 | dm_lcapital | lcapital - pmean(lcapital) |
| 14 | dm_llabor | llabor - pmean(llabor) |

Using these variables the within estimator is computed:

```
1  list allvar = dm_lsales dm_lcapital dm_llabor
2  scalar NT = $nobs
3  scalar N = rows(values(firm))
4  scalar k = nelem(allvar)-1
5
6  m4_within <- ols allvar
7  scalar correct = sqrt((NT-k)/(NT-N-k))
8  scalar correct_se_c = correct*$stderr(dm_lcapital)
9  scalar correct_se_l = correct*$stderr(dm_llabor)
```

The values function is used to create a vector that contains each of the unique elements of the series firm (and sorts them in ascending order). The number of rows will equal the number of firms, $n$, contained in the sample. The regression is estimated in line 6 and the results are:

$$\widehat{\text{dm\_lsales}} = \underset{(0.03577)}{0.03838}\,\text{dm\_lcapital} + \underset{(0.05323)}{0.3097}\,\text{dm\_llabor}$$
$$T = 400 \quad \bar{R}^2 = 0.0783 \quad F(2, 398) = 17.439 \quad \hat{\sigma} = 0.24897$$
$$\text{(standard errors in parentheses)}$$

The biggest problem here is that the standard errors are not computed with the correct degrees of freedom. Lines 7-9 of the script correct that.

```
Replaced scalar correct_se_c = 0.0507193
Replaced scalar correct_se_l = 0.0754665
```

which match the correct ones shown in *POE5* and from the difference estimation. The commands:

```
diff allvar
m4_diff <- ols d_dm_lsales d_dm_lcapital d_dm_llabor
```

produce:

$$\widehat{\text{d\_dm\_lsales}} = \underset{(0.05072)}{0.03838}\,\text{d\_dm\_lcapital} + \underset{(0.07547)}{0.3097}\,\text{d\_dm\_llabor}$$
$$T = 200 \quad \bar{R}^2 = 0.0759 \quad F(2,198) = 8.6759 \quad \hat{\sigma} = 0.35299$$
$$\text{(standard errors in parentheses)}$$

### Example 15.5 in *POE5*

In this example, the within transformation is used on the sample with $T = 3$ years worth of data. The data need to be reloaded so as to include all years available. The within transformation that uses `pmean` to compute and add the firm level means to the data must be computed with all three years of data.

```
open "@workdir\data\chemical2.gdt"
list allvar = lsales lcapital llabor

loop foreach j allvar --quiet
    series dm_$j = $j - pmean($j)
endloop

ols dm_lsales dm_lcapital dm_llabor
```

This yields:

$$\widehat{\text{dm\_lsales}} = \underset{(0.02709)}{0.08887}\,\text{dm\_lcapital} + \underset{(0.04134)}{0.3522}\,\text{dm\_llabor}$$
$$T = 600 \quad \bar{R}^2 = 0.1238 \quad F(1,598) = 85.615 \quad \hat{\sigma} = 0.24002$$
$$\text{(standard errors in parentheses)}$$

To correct the standard errors for degrees of freedom add:

```
scalar NT = $nobs
scalar N = rows(values(firm))
scalar k = nelem(allvar)-1
scalar correct = sqrt((NT-k)/(NT-N-k))
scalar correct_se_c = correct*$stderr(dm_lcapital)
scalar correct_se_l = correct*$stderr(dm_llabor)
```

which gives us:

```
Generated scalar correct_se_c = 0.0332076
Generated scalar correct_se_l = 0.05067
```

which is correct.

An equivalent way to estimate this model is using the least squares dummy variable estimator (LSDV). Here, an indicator variable is created for each individual in the sample. These are added to the model (dropping the intercept) and estimated by least squares.

There is a special operator that is used to generate indicators for each of the units in a panel. This function is `unitdum`, which must be used with `genr`.[3]

```
open "@workdir\data\chemical2.gdt"
genr unitdum
```

To estimate this fixed effects model with the LSDV estimator create a list that includes the wildcard and run a regression with this list added to the model.

```
list xvars = const lcapital llabor
list d = du_*
list d -= du_1
ols lsales xvars d
omit d --test-only
```

Notice that `unitdum` created an entire set of 200 indicators, one for each firm. To facilitate a hypothesis test that the fixed effects are equal, the model was reparameterized to include an overall intercept and one of the indicators was dropped to avoid the dummy variable trap. The `unitdum` function creates a prefix that is added to the variables that begins as `du_`. This enables one to use a wildcard `du_*` to include all of them in a list, `d`. The third line removes `du_1` from that list.

[3] There is also a `timedum` function that does the same thing for the time dimension variable. These were introduced in section 1.3.4.

Then, the model is estimated by OLS and the omit command is used to test the null hypothesis that all of the indicators parameters are equal zero.

There is a good reason why this formulation of the fixed effects model is not used more often. It produces a ton of output. Since there are 200 firms in the data, 200 lines of extra output will be sent to the screen (or table). Still, it works beautifully.

The abbreviated is:

```
Model 2: Pooled OLS, using 600 observations
Included 200 cross-sectional units
Time-series length = 3
Dependent variable: lsales

             coefficient    std. error     t-ratio      p-value
  ---------------------------------------------------------------
  const       8.06546       0.455529      17.71         3.74e-052 ***
  lcapital    0.0888719     0.0332076      2.676        0.0078    ***
  llabor      0.352244      0.0506700      6.952        1.49e-011 ***
  du_2       0.856350      0.252770      3.388       0.0008    ***
  du_3       1.19589       0.242080      4.940       1.15e-06  ***
  du_4       1.64249       0.240741      6.823       3.34e-011 ***
  du_5       1.03144       0.264892      3.894       0.0001    ***
  du_6       0.287830      0.243447      1.182       0.2378
  du_7       0.504913      0.258221      1.955       0.0512    *
  du_8       2.00044       0.248468      8.051       9.59e-015 ***
   .
   .
   .
   .
  du_199     0.513052      0.246341      2.083       0.0379    **
  du_200     0.914785      0.257840      3.548       0.0004    ***

Mean dependent var   9.868877   S.D. dependent var   1.191621
Sum squared resid    34.45147   S.E. of regression   0.294213
R-squared            0.959495   Adjusted R-squared   0.939040
F(201, 398)          46.90566   P-value(F)           2.1e-198
Log-likelihood       5.850284   Akaike criterion     392.2994
Schwarz criterion    1280.479   Hannan-Quinn         738.0500
rho                 0.301384   Durbin-Watson        1.551848
```

The joint test reveals:

```
 Test statistic: F(199, 398) = 22.7093, p-value 7.28428e-141
```

The hypothesis that the fixed effects are equal to one another is rejected at 5%.

The best way to estimate fixed effects models in **gretl** is using the panel command. This is the built-in command for estimating various types of panel models. The syntax for this important command is:

```
panel

Arguments:  depvar indepvars
Options:    --vcv (print covariance matrix)
            --fixed-effects (estimate with group fixed effects)
            --random-effects (random effects or GLS model)
            --nerlove (use the Nerlove transformation)
            --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 (compute Hausman test via matrix difference)
            --unbalanced=method (random effects only, see below)
            --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 fixed effects option (`--fixed-effects`) is used to estimate the Chinese chemical sales function:

```
open "@workdir\data\chemical2.gdt"
list xvars = lcapital llabor
p1 <- panel lsales xvars const --fixed-effects
```

This produces:

```
p1: Fixed-effects, using 600 observations
Included 200 cross-sectional units
Time-series length = 3
Dependent variable: lsales

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const        7.578        0.3523      21.51     1.22e-068 ***
  lcapital     0.08887      0.03321      2.676    0.0078    ***
  llabor       0.3522       0.05067      6.952    1.49e-011 ***

Mean dependent var   9.868877   S.D. dependent var   1.191621
Sum squared resid    34.45147   S.E. of regression   0.294213
LSDV R-squared       0.959495   Within R-squared     0.125239
LSDV F(201, 398)      46.90566   P-value(F)           2.1e-198
Log-likelihood        5.850284   Akaike criterion     392.2994
Schwarz criterion     1280.479   Hannan-Quinn         738.0500
rho                  0.301384   Durbin-Watson        1.551848

Joint test on named regressors -
  Test statistic: F(2, 398) = 28.4908
  with p-value = P(F(2, 398) > 28.4908) = 2.72879e-012

Test for differing group intercepts -
  Null hypothesis: The groups have a common intercept
  Test statistic: F(199, 398) = 22.7093
  with p-value = P(F(199, 398) > 22.7093) = 7.28428e-141
```

The slopes are equivalent to those from the LSDV model and printing the (199) fixed effects is suppressed. Furthermore, the joint test that the fixed effects are equal is automatically produced and a joint test that the slopes are equal zero is performed. The relationship between the within estimator and LSDV is no secret since several LSDV statistics are printed in the output. The LSDV $F$ tests the hypothesis that only a common intercept belongs in the model; all the 199 indicator coefficients and 2 slopes are jointly zero.

Before moving on, a pooled regression is estimated using panel robust standard errors. This model imposes the restriction that $\beta_{1i} = \beta_1$ for all individuals. All individuals share the same intercept. 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.

Using the data in *chemical3.gdt*, which contains data on 1000 Chinese chemical firms, a log-log sales model is proposed and estimated:

```
1 open "@workdir\data\chemical3.gdt"
2 list xvars = const lcapital llabor
3 OLS <- ols lsales xvars
4
5 set pcse off
6 Cluster <- ols lsales xvars --robust
```

First, notice is that the model is estimated by least squares with only a common intercept and two variables. In line 3, which is without a `robust` option, the usual OLS covariance is estimated. In line 6, the covariance of least squares is estimated using the `--robust` option. This uses the fact that errors for each firm are correlated with one another. Each firm is a cluster and these errors in each cluster are correlated with one another. On the other hand, clusters with each other. Firm 1 may have positive errors and firm 2 negative. But the errors of firms themselves are not correlated with other firms.[4] In summary, when panel data are loaded the `--robust` option defaults to an special version of the HAC covariance (see section 9.9.3) that is robust with respect to some forms of heteroskedasticity and autocorrelation due to the data's panel structure.

Setting `pcse` off ensures that the Arellano standard errors are computed. When this is on, **gretl** computes the Beck and Katz standard errors. It should be off by default. The cluster robust results appear below. Notice that **gretl** refers to the standard errors as HAC, but these are the Arellano cluster robust standard errors in parentheses.

Cluster: Pooled OLS, using 3000 observations
Included 1000 cross-sectional units
Time-series length = 3
Dependent variable: lsales
Robust (HAC) standard errors

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 5.541 | 0.1424 | 38.90 | 0.0000 |
| lcapital | 0.3202 | 0.02731 | 11.72 | 0.0000 |
| llabor | 0.3948 | 0.03903 | 10.12 | 0.0000 |

[4] For large n, knowing that the errors of firm 1 are negative and positively correlated with one another tells me nothing about whether the errors in firm 2 are negative or positive. They are independent.

To force **gretl** to produce the HCCME (inconsistent in this case) standard errors one must perform some trickery on the data. The data must be redefined as a cross-section in order for the HCCME standard errors to be computed for this OLS regression. That is done here:

```
setobs 1 1 --cross-section
Het_HC3 <- ols lsales xvars --robust
```

Be sure to restore the data to its proper format before estimating any more models with these data.

The results for OLS with 'usual,' HC3, and cluster standard errors is shown here:

Pooled OLS estimates
Dependent variable: lsales

| | OLS | Het_HC3 | Cluster |
|---|---|---|---|
| const | 5.5408** | 5.5408** | 5.5408** |
| | (0.0828) | (0.0890) | (0.1424) |
| lcapital | 0.3202** | 0.3202** | 0.3202** |
| | (0.0153) | (0.0179) | (0.0273) |
| llabor | 0.3948** | 0.3948** | 0.3948** |
| | (0.0225) | (0.0258) | (0.0390) |
| $n$ | 3000 | 3000 | 3000 |
| $\bar{R}^2$ | 0.5578 | 0.5578 | 0.5578 |
| $\ell$ | −3837 | −3837 | −3837 |

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

The cluster standard errors tend to be quite a bit larger than the inconsistent ones in columns (1) and (2). This is a typical result.

### Example 15.8 in *POE5*

Finally, cluster robust standard errors can also be used with the fixed effects estimator. The results:

Fixed-effects estimates
Dependent variable: lsales

| | FE | FE-Cluster |
|---|---|---|
| const | 7.9463** | 7.9463** |
| | (0.2143) | (0.3027) |
| lcapital | 0.1160** | 0.1160** |
| | (0.0195) | (0.0273) |
| llabor | 0.2689** | 0.2689** |
| | (0.0307) | (0.0458) |
| $n$ | 3000 | 3000 |
| $\bar{R}^2$ | 0.0582 | 0.0582 |
| $\ell$ | −681.9 | −681.9 |

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

Again, the cluster robust standard errors are 50% larger. Each of the coefficients remains significant though, so hypothesis tests are not being substantively impacted.

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.

## 15.3 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 $\beta_{1i}$ term in equation 15.3 is modeled:

$$\beta_{1i} = \bar{\beta}_1 + u_i \tag{15.4}$$

where the $u_i$ are random individual differences that are the same in each time period.

$$\begin{align} y_{it} &= \bar{\beta}_1 + \beta_2 x_{2it} + \beta_3 x_{3it} + (e_{it} + u_i) \tag{15.5} \\ &= \bar{\beta}_1 + \beta_2 x_{2it} + \beta_3 x_{3it} + v_{it} \tag{15.6} \end{align}$$

where the combined error is

$$v_{it} = u_i + e_{it}$$

the key property of the new error term is that it is homoskedastic

$$\sigma_v^2 = \text{var}\left(v_{it}\right) = \text{var}\left(u_i + e_{it}\right) = \sigma_u^2 + \sigma_e^2 \tag{15.7}$$

and serially correlated. For individual $i$, that covariance among his errors is

$$\text{cov}\,(v_{it}, v_{is}) = 0$$

for $t \neq s$. Also, 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 15.7 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 *POE5* 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

$$\theta = 1 - \frac{\sigma_e}{\sqrt{T\sigma_u^2 + \sigma_e^2}} \tag{15.8}$$

This parameter $\theta$ is estimated from the data and the transformation are

$$y_{it}^* = y_{it} - \theta\bar{y}_i, \quad x_{1it}^* = 1 - \theta, \quad x_{2it}^* = x_{2it} - \theta\bar{x}_{2i}, \quad x_{3it}^* = x_{3it} - \theta\bar{x}_{3i} \tag{15.9}$$

The bars over the variables indicate means for the $i^{th}$ individual taken over the available time periods. Gretl estimates $\theta$ and the variances.

### Example 15.9 in *POE5*

For the 1000 Chinese chemical firms the fixed effects, random effects, and random effects with cluster standard errors are estimated as:

```
open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
FE <- panel lsales xvars --fixed-effects
FGLS <- panel lsales xvars --random-effects
FGLS_cluster <- panel lsales xvars --random-effects --robust
```

The estimates are compared below:

Fixed-effects estimates
Dependent variable: lsales

|  | FE | FGLS | FGLS_cluster |
|---|---|---|---|
| const | 7.9463** | 6.1718** | 6.1718** |
|  | (0.2143) | (0.1142) | (0.1428) |
| lcapital | 0.1160** | 0.2393** | 0.2393** |
|  | (0.0195) | (0.0147) | (0.0221) |
| llabor | 0.2689** | 0.4140** | 0.4140** |
|  | (0.0307) | (0.0220) | (0.0327) |
| $n$ | 3000 | 3000 | 3000 |
| $\bar{R}^2$ | 0.0582 |  |  |
| $\ell$ | −681.9 | −3867 | −3867 |

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

The estimates match those from *POE5* nicely. The estimated value of $\alpha$ in the random effects estimator is 0.73529. Gretl calls this 'theta.'

### Example 15.10 in *POE5*

In this example we return to the wage model visited earlier and estimated using the *nls_panel.gdt* data. In this example the model is estimated using both fixed- and random-effects. To revisit a technique used earlier in this manual, the results are assembled into a model table using a script (as opposed to the GUI, which I've been using liberally).

The model considered is found in equation (15.3) below.

$$\begin{aligned} ln(wage)_{it} &= \beta_{1i} + \beta_2 educ_i + \beta_3 exper_{it} + \beta_4 exper^2_{it} + \beta_5 tenure_{it} \\ &+ \beta_6 tenure^2_{it} + \beta_7 south_{it} + \beta_8 union_{it} + \beta_9 black_i + e_{it} \end{aligned} \tag{15.10}$$

The **gretl** script used is:

```
open "@workdir\data\nls_panel.gdt"
list xvars = educ exper exper2 tenure tenure2 south union black
FE <- panel lwage xvars const --fixed-effects
modeltab add
RE <- panel lwage xvars const --random-effects
modeltab add
Pooled <- panel lwage xvars const --pooled --robust
modeltab add
modeltab show
modeltab free
```

Recall that `modeltab add` adds the results of the preceding regression to a model table. `modeltab show` displays the table, and `modeltab free` clears the table. The results from this set of regressions is shown below:

Dependent variable: lwage

| | Pooled | FE | RE |
|---|---|---|---|
| | Pooled OLS | Within | GLS |
| const | 0.4766** | 1.4500** | 0.5339** |
| | (0.0846) | (0.0401) | (0.0799) |
| educ | 0.0714** | | 0.0733** |
| | (0.0055) | | (0.0053) |
| exper | 0.0557** | 0.0411** | 0.0436** |
| | (0.0113) | (0.0066) | (0.0064) |
| exper2 | −0.0011** | −0.0004 | −0.0006** |
| | (0.0005) | (0.0003) | (0.0003) |
| tenure | 0.0150** | 0.0139** | 0.0142** |
| | (0.0071) | (0.0033) | (0.0032) |
| tenure2 | −0.0005 | −0.0009** | −0.0008** |
| | (0.0004) | (0.0002) | (0.0002) |
| south | −0.1060** | −0.0163 | −0.0818** |
| | (0.0271) | (0.0361) | (0.0224) |
| union | 0.1322** | 0.0637** | 0.0802** |
| | (0.0271) | (0.0143) | (0.0132) |
| black | −0.1167** | | −0.1167** |
| | (0.0281) | | (0.0302) |
| $n$ | 3580 | 3580 | 3580 |
| $\bar{R}^2$ | 0.3241 | 0.1430 | |
| $\ell$ | −1630 | 1174 | −1649 |

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

Notice that the time-invariant variables `educ` and `black` cannot be estimated using fixed effects. Also, the pooled OLS with cluster robust standard errors are reasonably close in magnitude to the

FGLS random effects estimates. As usual, the cluster standard errors are larger than the others. The similarity of the FE and RE estimates sometimes suggest that the unobserved components may not be correlated with regressors. This makes the RE estimator efficient. A test for this is explored below.

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.

## 15.4 Specification Tests

There are a couple of key specification tests one should 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.

### 15.4.1 Breusch-Pagan Test

The Breusch Pagan test statistic is based on a Lagrange multiplier and is computed

$$LM = \sqrt{\frac{nT}{2\,(T-1)}}\left\{\frac{\sum\limits_{i=1}^{n}\left(\sum\limits_{t=1}^{T}\hat{e}_{it}\right)^{2}}{\sum\limits_{i=1}^{n}\sum\limits_{t=1}^{T}\hat{e}_{it}^{2}} - 1\right\} \qquad (15.11)$$

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 $\chi^2(1)$ 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.

To compute equation (15.11) we use:

```
ols lsales xvars
series ehat = $uhat
scalar sse = $ess
scalar NT = $nobs
scalar N = rows(values(firm))
scalar T = rows(values(year))

series sm = psum(ehat)
matrix mti = pshrink(sm)
scalar Sum = sum(mti.^2)

scalar LM = sqrt(nT/(2*(T-1)))*((Sum/sse)-1)
printf "The LM test statistic = %.3f with p-value = %.3f\n", \
       LM, pvalue(z,LM)
```

This script uses the `values` function to identify the unique firms and years within the data. Counting the rows of these produces $n$ and $T$ for the computations. Also, two other panel functions are used to compute *LM*. The first is `psum`. This command computes the sum of the time series for each individual and places this value as a variable in the data. The next command, `pshrink`, shrinks this $nT$ vector into an $n$ vector that contains only a single sum for each observation. The sum-of-squares of these is used in the computation of LM. The advantage of this calculation is that it offers the possibility of conducting a one-sided test using the $N(0, 1)$ distribution.

For the Chinese chemical firms the result is

```
The LM test statistic = 44.064 with p-value = 0.000
```

```
'Between' variance = 0.612725
'Within' variance = 0.138509
theta used for quasi-demeaning = 0.73529
corr(y,yhat)^2 = 0.556365
```

These match the ones in *POE5* 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 15.6.

### 15.4.2 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 $\chi^2$ 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 (or **contrasts**) 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 $\chi^2$ 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.

### Example 15.12 in *POE5*

In this example a special case of the matrix difference approach is applied to the chemical firms data. In this example a single contrast is taken for the estimation of $\beta_2$, the coefficient on ln($capital$). The $t$-difference is

$$t = \frac{b_{FE,2} - b_{RE,2}}{[se(b_{FE,2})^2 - se(b_{RE,2})^2]^{1/2}}$$

The script to compute this is:

```
1  open "@workdir\data\chemical3.gdt"
2  list xvars = const lcapital llabor
3  RE <- panel lsales xvars --random-effects
4  scalar b_c = $coeff(lcapital)
5  scalar v_c = $stderr(lcapital)^2
6
7  FE <- panel lsales xvars --fixed-effects
8  scalar b_c_f = $coeff(lcapital)
9  scalar v_c_f = $stderr(lcapital)^2
10
11 scalar t_stat = (b_c_f-b_c)/sqrt(v_c_f-v_c)
12 printf "Hausman t-stat = %.3f with p-value = %.3f\n",\
13        t_stat, 2*pvalue(n,abs(t_stat))
```

The RE and FE estimators are computed and the coefficient on `lcapital` and its estimated standard error are saved as scalars. The contrast is computed in line 11 and the results printed to the screen. The outcome is:

```
Hausman t-stat = -9.555 with p-value = 0.000
```

It is significant at 5% and we reject the exogeneity of the random effect. The RE estimator is inconsistent and the FE estimator is preferred.

To perform the contrasts test based on the entire set of slopes, use the `hausman --matrix-diff` command after estimating the model by OLS. The data must be structured as a panel in order for this to work.

```
open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
ols lsales xvars
hausman --matrix-diff
```

The output shows the pooled, FE, and RE estimates as well as the results of the two-sided *LM* test and the Hausman test. The Hausman result (`--matrix-diff` option) is:

```
Hausman test statistic:
 H = 98.8166 with p-value = prob(chi-square(2) > 98.8166) = 3.48535e-022
(A low p-value counts against the null hypothesis that the random effects
model is consistent, in favor of the fixed effects model.)
```

The result matches the one reported in *POE5* and the exogeneity null is rejected at 5%.

Finally, it is worth noting that the regression based version of the Hausman test is printed by default whenever a RE regression is estimated.

### Example 15.13 in *POE5*

In this exercise the $t$-based contrast test is computed for each of the coefficients of the wage equation estimated using *nls_panel.gdt*. Since there are six coefficients that can be estimated by both FE and RE a loop is employed. A key to making this work is to order the variables so that the time-invariant ones (which cannot be estimated by FE) are listed after the others. The script follows:

```
open "@workdir\data\nls_panel.gdt"
list xvars = exper exper2 tenure tenure2 south union black educ
list TV_vars = exper exper2 tenure tenure2 south union
list TIV_vars = black educ
FE <- panel lwage TV_vars const --fixed-effects

matrix b_fe = $coeff
matrix var_fe = diag($vcv)

RE <- panel lwage TV_vars TIV_vars const --random-effects
matrix b = $coeff
matrix b_re = b[1:7]
matrix vars = diag($vcv)
matrix var_re = vars[1:7]

loop i=2..7
    scalar t_stat = (b_fe[i]-b_re[i])/sqrt(var_fe[i] - var_re[i])
    printf "\n Hausman t-stat = %.3f with p-value = %.3f\n",\
            t_stat, 2*pvalue(n,abs(t_stat))
endloop
```

This crude program appears to work. Several of the individual $t$-ratios are significant at 5%. According to the results these include `union`, `south`, `tenure`$^2$ and `exper`$^2$.

```
Hausman t-stat = -1.373 with p-value = 0.170
Hausman t-stat = 2.004 with p-value = 0.045
Hausman t-stat = -0.290 with p-value = 0.772
Hausman t-stat = -2.110 with p-value = 0.035
Hausman t-stat = 2.309 with p-value = 0.021
Hausman t-stat = -3.093 with p-value = 0.002
```

The statistics in iterations 2, 4, 5, and 6 fall within the rejection region. There is really no reason to do a series of these tests. The preferred procedure it to test endogeneity jointly using the $\chi^2(6)$ statistic obtained using the `hausman` command.

The joint Hausman test is also significant at 5%.

```
Hausman test statistic:
 H = 20.7252 with p-value = prob(chi-square(6) > 20.7252) = 0.00205521
(A low p-value counts against the null hypothesis that the random effects
model is consistent, in favor of the fixed effects model.)
```

## 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

$$\bar{y}_i = \bar{\beta}_1 + \beta_2 \bar{x}_{2i} + \beta_3 \bar{x}_{3i} + u_i + \bar{e}_i \tag{15.12}$$

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 an 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 "@workdir\data\nls_panel.gdt"
setobs id year --panel-vars
list xvars = educ exper exper2 tenure tenure2 union black south
panel lwage xvars --between
```

The results for each of the estimators, in tabular form, are in Table 15.6.

### 15.5.1 Mundlak Approach

Mundlak proposed that if unobservable heterogeneity is correlated with the explanatory variables then the random effects may be correlated with the time averages of the explanatory variables. To test for endogeneity, he adds the time averages to the model and tests their joint significance using an $F$-test. His test statistic is never negative (as the Hausman contrast test can be) and Mundlak's $F$ can be made robust with respect to autocorrelation and heteroskedasticity using cluster robust covariance estimation.

#### Example 15.14 in *POE5*

Mundlak's approach is used to determine whether the unobserved heterogeneity in the chemical plants is correlated with capital and labor. The pmean function is used to add the time means of $\ln(capital)$ and $\ln(labor)$ to the data. A regression is estimated and the coefficients on the time means are jointly tested for significance.

The code is:

```
open "@workdir\data\chemical3.gdt"
list allvar = lsales lcapital llabor

loop foreach j allvar --quiet
        series $j_bar = pmean($j)
endloop

OLS <- ols allvar const lcapital_bar llabor_bar --robust
omit lcapital_bar llabor_bar --chi-square
RE <- panel allvar const lcapital_bar llabor_bar --random-effects
omit lcapital_bar llabor_bar --chi-square
Cluster <- panel allvar const lcapital_bar llabor_bar\
        --random-effects --robust
omit lcapital_bar llabor_bar --chi-square
```

Dependent variable: lwage

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| | Within | FGLS | Between | 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 |
| $\ell$ | 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 15.6: Fixed Effects (Within), Random Effects (FGLS), Between, and Pooled OLS with robust standard errors.

Notice that in lines 4-6 a loop is used to add the time means of all variables to the dataset. The new variables are recognized as `varname_bar` in the main **gretl** window.

In the example the model is estimated by pooled least squares with cluster standard errors, by RE with conventional FGLS standard errors, and by RE with cluster standard errors. The results are shown below:

Dependent variable: lsales

| | (1) | (2) | (3) |
|---|---|---|---|
| | Pooled OLS | GLS | GLS-Cluster |
| const | 5.455$^{**}$ | 5.455$^{**}$ | 5.455$^{**}$ |
| | (0.1484) | (0.1371) | (0.1484) |
| lcapital | 0.1160$^{**}$ | 0.1160$^{**}$ | 0.1160$^{**}$ |
| | (0.02735) | (0.01955) | (0.02735) |
| llabor | 0.2689$^{**}$ | 0.2689$^{**}$ | 0.2689$^{**}$ |
| | (0.04582) | (0.03067) | (0.04582) |
| lcapital_bar | 0.2223$^{**}$ | 0.2223$^{**}$ | 0.2223$^{**}$ |
| | (0.04125) | (0.03338) | (0.04125) |
| llabor_bar | 0.1095$^{*}$ | 0.1095$^{**}$ | 0.1095$^{*}$ |
| | (0.06220) | (0.05010) | (0.06220) |
| $n$ | 3000 | 3000 | 3000 |
| $\bar{R}^2$ | 0.5614 | | |
| $\ell$ | −3824 | −3824 | −3824 |

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

It is interesting that the pooled OLS and GLS-RE coefficient estimates are equivalent. The cluster standard errors are larger than the conventional FGLS ones. The joint test of significance can be conducted using each regression. The `omit` statements do this. The results are:

```
  OLS with Robust
   Null hypothesis: the regression parameters are zero for the variables
     lcapital_bar, llabor_bar
   Wald test: Chi-square(2) = 56.5859, p-value 5.15854e-013
   (LR test: Chi-square(2) = 26.3213, p-value 1.92492e-006)

 GLS -- Conventional
   Null hypothesis: the regression parameters are zero for the variables
```

```
    lcapital_bar, llabor_bar
  Wald test: Chi-square(2) = 96.9967, p-value 8.65817e-022
  (LR test: Chi-square(2) = 86.266, p-value 1.85173e-019)

GLS -- Cluster
  Null hypothesis: the regression parameters are zero for the variables
    lcapital_bar, llabor_bar
  Wald test: Chi-square(2) = 56.5859, p-value 5.15854e-013
  (LR test: Chi-square(2) = 86.266, p-value 1.85173e-019)
```

The OLS and GLS cluster results are identical, matching those found in *POE5*.

### Example 15.16 in *POE5*

The Mundlak regressions and tests are repeated for the wage equation. The script is:

```
1  open "@workdir\data\nls_panel.gdt"
2  list xvars = exper exper2 tenure tenure2 south union black educ
3
4  loop foreach j xvars --quiet
5          series bar_$j = pmean($j)
6  endloop
7
8  list barvars = bar*
9  list barvars -= bar_black bar_educ
10 OLS <- ols lwage const xvars barvars --robust
11 omit barvars --chi-square --test-only
12 RE <- panel lwage const xvars barvars --random-effects
13 omit barvars --chi-square --test-only
14 Cluster <- panel lwage const xvars barvars --random-effects --robust
15 omit barvars --chi-square --test-only
16 FE <-  panel lwage const xvars --fixed-effects --robust
```

The regressors include experience, experience-squared, tenure, tenure-squared, south, black, union, and education. With so many variables to find time means for, a loop is most convenient. In this example I added the bar_ prefix so that I could unambiguously employ a wildcard to make a list. This appears in line 8 where the list barvars is constructed. In the following line two of the means are removed. These are time-invariant and their means are perfectly collinear with the constant. Otherwise the estimation is very similar to the preceding example. RE is used with FGLS standard errors, RE with cluster standard errors and fixed-effects results are added. I omit OLS with clusters since we've already seen that they are the same as RE with clusters. The results are:

Random-effects (GLS) estimates

Dependent variable: lwage

| | (1) | (2) | (3) |
|---|---|---|---|
| const | 0.4167$^{**}$ | 0.4167$^{**}$ | 1.450$^{**}$ |
| | (0.1358) | (0.1101) | (0.05503) |
| exper | 0.04108$^{**}$ | 0.04108$^{**}$ | 0.04108$^{**}$ |
| | (0.006620) | (0.008250) | (0.008240) |
| exper2 | −0.0004091 | −0.0004091 | −0.0004091 |
| | (0.0002733) | (0.0003303) | (0.0003299) |
| tenure | 0.01391$^{**}$ | 0.01391$^{**}$ | 0.01391$^{**}$ |
| | (0.003278) | (0.004220) | (0.004215) |
| tenure2 | −0.0008962$^{**}$ | −0.0008962$^{**}$ | −0.0008962$^{**}$ |
| | (0.0002059) | (0.0002498) | (0.0002495) |
| south | −0.01632 | −0.01632 | −0.01632 |
| | (0.03615) | (0.05855) | (0.05848) |
| union | 0.06370$^{**}$ | 0.06370$^{**}$ | 0.06370$^{**}$ |
| | (0.01425) | (0.01688) | (0.01686) |
| black | −0.1216$^{**}$ | −0.1216$^{**}$ | |
| | (0.03166) | (0.02842) | |
| educ | 0.07077$^{**}$ | 0.07077$^{**}$ | |
| | (0.005387) | (0.005565) | |
| bar_exper | 0.02511 | 0.02511 | |
| | (0.02437) | (0.02228) | |
| bar_exper2 | −0.001197 | −0.001197 | |
| | (0.001037) | (0.0009587) | |
| bar_tenure | 0.002649 | 0.002649 | |
| | (0.01263) | (0.01367) | |
| bar_tenure2 | 0.0004014 | 0.0004014 | |
| | (0.0007323) | (0.0007718) | |
| bar_south | −0.08899$^{*}$ | −0.08899 | |
| | (0.04641) | (0.06523) | |
| bar_union | 0.09204$^{**}$ | 0.09204$^{**}$ | |
| | (0.03822) | (0.04152) | |
| $n$ | 3580 | 3580 | 3580 |
| $\bar{R}^2$ | | | 0.1430 |
| $\ell$ | −1621 | −1621 | 1174 |

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

The Hausman tests reveal:

```
RE -- FGLS standard errors
  Null hypothesis: the regression parameters are zero for the variables
    bar_exper, bar_exper2, bar_tenure, bar_tenure2, bar_south, bar_union
  Wald test: Chi-square(6) = 20.4371, p-value 0.00231433
  (F-form: F(6, 3565) = 3.40618, p-value 0.00235946)

RE -- Cluster standard errors
  Null hypothesis: the regression parameters are zero for the variables
    bar_exper, bar_exper2, bar_tenure, bar_tenure2, bar_south, bar_union
  Wald test: Chi-square(6) = 17.2626, p-value 0.00836519
  (F-form: F(6, 715) = 2.8771, p-value 0.00890482)
```

The statistic falls within the rejection region of a 5% test in both versions. The $p$-value for cluster robust test is $.008 < .05$. The exogeneity of the random effects is rejected.

### 15.5.2 Hausman-Taylor

The Hausman-Taylor estimator is an instrumental variables estimator applied to a random effects model. The instrumental variables enable one to avoid inconsistency caused by correlation between random effects and some of the model's explanatory variables. The estimator requires one to separate the regressors into groups: time-varying exogenous, time-varying endogenous, time-invariant exogenous, and time-invariant endogenous. There must be at least as many time-varying exogenous regressors as time-invariant endogenous ones.

The routine to estimate this is somewhat complicated. It involves at least five steps and several regressions, including a FE regression, a 2SLS regression using augmented data, and another FGLS estimation. The standard error computation involves one more step. Given the complexity of this, I chose to use a beta version of a package written by Allin Cottrell. Allin was kind enough to let me use this provided that none of us hold him responsible for the results. That said, the package replicates the Baltagi example and the *POE5* example used below. Whatever the perceived deficiencies, they pale compared to what I would produce if I had to do this from scratch.

First, you must place the files included in the package, which includes the **gretl** function

```
hausman_taylor.gfn
```

into your **gretl** functions directory so that they will be found when by the include statement.[5] Then, open the dataset and define the lists of time-varying and time-invariant series. Finally, use the `hausman_taylor` function as shown, which will write output to a bundle called **b**. Be careful

[5] On my Windows machine this location is `C:\Users\leead\AppData\Roaming\gretl\functions`.

to order the lists in the function properly. First is time-varying exogenous, second is time-varying endogenous (both of these are in the regression function). Next come the time-varying exogenous and finally the time-invariant endogenous variable(s). The output from the bundle will appear as a session icon. Navigate to the icon view window and click on the folder labeled **b**.

```
include hausman_taylor.gfn

open nls_panel.gdt --quiet

# List definitions  TV=time-varying; TIV=time-invariant
list X1 = exper exper2 tenure tenure2 union   # TV exogenous
list Z1 = black             # Time-invariant exogenous
list X2 = south             # TV endogenous
list Z2 = educ              # TIV endogenous

bundle b = hausman_taylor(lwage, X1, X2, Z1, Z2)
```

The results are shown below.

```
Hausman-Taylor estimates for lwage
using 3580 observations (n = 716, T = 5)

             coefficient    std. error       z       p-value
  -----------------------------------------------------------
  const      -0.750769      0.586236      -1.281    0.2003
  exper       0.0399079     0.00647453     6.164    7.10e-010 ***
  exper2     -0.000391341   0.000267634   -1.462    0.1437
  tenure      0.0143257     0.00315970     4.534    5.79e-06  ***
  tenure2    -0.000852561   0.000197405   -4.319    1.57e-05  ***
  union       0.0719692     0.0134545      5.349    8.84e-08  ***
  south      -0.0317122     0.0348474     -0.9100   0.3628
  black      -0.0359136     0.0600681     -0.5979   0.5499
  educ        0.170508      0.0444628      3.835    0.0001    ***

  sigma_u = 0.44986996
  sigma_e = 0.19490590
  theta   = 0.80978255

Hausman test: chi-square(6) = 5.71023 [0.4564]
Sargan over-identification test: chi-square(4) = 7.33465 [0.1192]
```

These match what I get in Stata as well as the output in Table 15.8 or *POE5*. Thanks Allin!

## 15.6 Script

```
set echo off
open "@workdir\data\nls_panel.gdt"
# pooled least squares
list xvars = const educ south black union exper exper2 tenure tenure2

# Example 15.1
print id year lwage xvars --byobs

panel lwage xvars --pooled --robust

# Example 15.2
open "@workdir\data\chemical2.gdt"
dummify year
smpl (year == 2005 || year == 2006) --restrict
list xvars = const lcapital llabor
diff xvars lsales
ols lsales xvars
m1 <- ols d_lsales d_lcapital d_llabor
m2 <- ols lsales xvars
# Example 15.3
open "@workdir\data\nls_panel.gdt"
smpl (year==87 || year==88) --restrict
diff lwage exper
m3 <- ols d_lwage d_exper

# Example 15.4
open "@workdir\data\chemical2.gdt"
list xvar = lsales lcapital llabor
smpl year !=2004 --restrict --replace --permanent

loop foreach j xvar --quiet       # The within transformation
    series dm_$j = $j - pmean($j)
endloop

summary lsales --by=firm          # Produces lots of output

# smpl year !=2004 --restrict --replace
scalar NT = $nobs
scalar N = rows(values(firm))
scalar k = 2

list allvar = dm_lsales dm_lcapital dm_llabor
diff allvar
m4_diff <- ols d_dm_lsales d_dm_lcapital d_dm_llabor

# Within estimator
scalar NT = $nobs
scalar N = rows(values(firm))
scalar k = nelem(allvar)-1

m4_within <- ols allvar
scalar correct = sqrt((NT-k)/(NT-N-k))
scalar correct_se_c = correct*$stderr(dm_lcapital)
scalar correct_se_l = correct*$stderr(dm_llabor)

# Example 15.5
# The Within transformation
open "@workdir\data\chemical2.gdt"
list allvar = lsales lcapital llabor

loop foreach j allvar --quiet
    series dm_$j = $j - pmean($j)
endloop

m5_within <- ols dm_lsales dm_lcapital dm_llabor

scalar NT = $nobs
scalar N = rows(values(firm))
scalar k = nelem(allvar)-1
scalar correct = sqrt((NT-k)/(NT-N-k))
scalar correct_se_c = correct*$stderr(dm_lcapital)
scalar correct_se_l = correct*$stderr(dm_llabor)

# Example 15.6 Fixed Effects
open "@workdir\data\chemical2.gdt"
list xvars = lcapital llabor
p1 <- panel lsales xvars const --fixed-effects

# fixed effects and lsdv
genr unitdum
list xvars = const lcapital llabor
list d = du_*
list d -= du_1
ols lsales xvars d
omit d --test-only

panel lsales xvars --fixed-effects

# fe, re, between, and pooled comparison
open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
OLS <- ols lsales xvars
set pcse off
Cluster <- ols lsales xvars --robust

setobs 1 1 --cross-section
Het_HC3 <- ols lsales xvars --robust

# Example 15.8
```

```
setobs firm year --panel
p1 <- panel lsales xvars --fixed-effects
p2 <- panel lsales xvars --fixed-effects --robust

# Example 15.9
open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
FE <- panel lsales xvars --fixed-effects
FGLS <- panel lsales xvars --random-effects
FGLS_cluster <- panel lsales xvars --random-effects --robust

# Example 15.10
open "@workdir\data\nls_panel.gdt"
list xvars = educ exper exper2 tenure tenure2 south union black
FE <- panel lwage xvars const --fixed-effects
modeltab add
RE <- panel lwage xvars const --random-effects
modeltab add
Pooled <- panel lwage xvars const --pooled --robust
modeltab add
modeltab show
modeltab free

# Example 15.11
open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
ols lsales xvars
series ehat = $uhat
scalar sse = $ess
scalar NT = $nobs
scalar N = rows(values(firm))
scalar T = rows(values(year))

series sm = psum(ehat)
matrix mti = pshrink(sm)
scalar Sum = sum(mti.^2)

scalar LM = sqrt(NT/(2*(T-1)))*((Sum/sse)-1)
printf "The LM test statistic = %.3f with pvalue = %.3f\n",\
       LM, pvalue(z,LM)

# Between Estimator
open "@workdir\data\nls_panel.gdt"
setobs id year --panel-vars
list xvars = const educ exper exper2 tenure tenure2 union black south
panel lwage xvars --between

# Example 15.12
open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
RE <- panel lsales xvars --random-effects
```

```
scalar b_c = $coeff(lcapital)
scalar v_c = $stderr(lcapital)^2

FE <- panel lsales xvars --fixed-effects
scalar b_c_f = $coeff(lcapital)
scalar v_c_f = $stderr(lcapital)^2

scalar t_stat = (b_c_f-b_c)/sqrt(v_c_f-v_c)
printf "Hausman t-stat = %.3f with p-value = %.3f\n",\
       t_stat, 2*pvalue(n,abs(t_stat))

open "@workdir\data\chemical3.gdt"
list xvars = const lcapital llabor
ols lsales xvars
hausman --matrix-diff

# Example 15.13
open "@workdir\data\nls_panel.gdt"
list xvars = exper exper2 tenure tenure2 south union black educ
ols lwage xvars const
hausman --matrix-diff

list TV_vars = exper exper2 tenure tenure2 south union
list TIV_vars = black educ

FE <- panel lwage TV_vars const --fixed-effects
matrix b_fe = $coeff
matrix var_fe = diag($vcv)

RE <- panel lwage TV_vars TIV_vars const --random-effects
matrix b = $coeff
matrix b_re = b[1:7]
matrix vars = diag($vcv)
matrix var_re = vars[1:7]

loop i=2..7
    scalar t_stat = (b_fe[i]-b_re[i])/sqrt(var_fe[i] - var_re[i])
    printf "Hausman t-stat = %.3f with p-value = %.3f\n",\
            t_stat, 2*pvalue(n,abs(t_stat))
endloop
# Example 15.4
# Mundlak
open "@workdir\data\chemical3.gdt"
list allvar = lsales lcapital llabor

loop foreach j allvar --quiet
        series $j_bar = pmean($j)
endloop

OLS <- ols allvar const lcapital_bar llabor_bar --robust
omit lcapital_bar llabor_bar --chi-square
```

```
RE <- panel allvar const lcapital_bar llabor_bar --random-effects
omit lcapital_bar llabor_bar --chi-square
Cluster <- panel allvar const lcapital_bar llabor_bar\
     --random-effects --robust
omit lcapital_bar llabor_bar --chi-square

# Example 15.15
open "@workdir\data\nls_panel.gdt"
list xvars = exper exper2 tenure tenure2 south union black educ

loop foreach j xvars --quiet
        series bar_$j = pmean($j)
endloop

list barvars = bar*
list barvars -= bar_black bar_educ
OLS <- ols lwage const xvars barvars --robust
omit barvars --chi-square --test-only
RE <- panel lwage const xvars barvars --random-effects
omit barvars --chi-square --test-only
Cluster <- panel lwage const xvars barvars --random-effects --robust
omit barvars --chi-square --test-only
FE <-  panel lwage const xvars --fixed-effects --robust

# Example 15.16
include hausman_taylor.gfn

open nls_panel.gdt --quiet

# List definitions  TV=time-varying; TIV=time-invariant
list X1 = exper exper2 tenure tenure2 union   # TV exogenous
list Z1 = black             # Time-invariant exogenous
list X2 = south             # TV endogenous
list Z2 = educ              # TIV endogenous

bundle b = hausman_taylor(lwage, X1, X2, Z1, Z2)
```

# Chapter 16

# Qualitative and Limited Dependent Variable Models

## 16.1 Introduction

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.

Choices can be between two (binary) or more (multinomial) alternatives. Multinomial choices can be made from a hierarchy (ordered) or they may not. For instance, a choice from a satisfaction scale is ordered and the choice of whether to walk, drive, or ride the bus to work is not.

A limited dependent variable is continuous, but its range of values is constrained in some way. Some of the values of the dependent variable are unobserved or, if all are observed, some are constrained to the same value if the actual value exceeds (or falls below) some threshold. Simple versions of both types of model are considered below.

We start with binary decisions and then move to multinomial choice models. Models for count data are estimated and censored and truncated regressions are considered. When computing these estimators and related statistics there is a trade-off between using very generalized and complex programs, which require significant investments in time to write and debug, and using simpler single-purpose functions and programs that work for a particular example, but not necessarily for others. One-off programs are frequently used by actual econometricians who frequently work on different problems using different methods. These examples are targeted to this group.

That said, the principles used in the construction of these examples can provide a template for other examples that one may consider. For instance, obtaining standard errors for marginal effects using the delta method is fairly routine. The main unique input required is a function that computes the desired probability or nonlinear function for which a derivative is required.

Also, we will turn to some user written functions and programs that are available on **gretl**'s function package server. One of these was used in Chapter 14 to estimate a GARCH model.[1] In this chapter, the *HIP.gfn*[2] and Claudia Pigini and *lp-mfx.gfn*[3] packages are used is several examples.

## 16.2 Linear Probability

In a **binary choice** model, the decision to model has only two possible outcomes (see sections 7.7 and 8.6). An artificial number is assigned to each outcome before further empirical analysis can be done. 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, the dependent variable is

$$y_i = \begin{cases} 1 & \text{if individual i has the quality} \\ 0 & \text{if not.} \end{cases}$$

The linear probability model, first considered in section 7.3, models the probability that $y_i = 1$ as a linear function of the independent variables.

### Example 16.1 in *POE5*

In this example, which was also considered in section 7.3, a binary decision is made about whether to drive by automobile or to take public transportation.

$$auto_i = \begin{cases} 1 & \text{if individual i chooses auto} \\ 0 & \text{if public transportation is chosen} \end{cases} \tag{16.1}$$

This is estimated as a function of the commuting time differential between the two alternatives. That is $dtime = (bustime - autotime)/10$. In a linear probability model this becomes

$$auto_i = \beta_1 + \beta_2 dtime_i + e_i \tag{16.2}$$

The data are found in the *transport.gdt* dataset. These are loaded and simple summary statistics are computed.

```
open "@workdir\data\transport.gdt"
summary --simple
```

[1] The *gig.gfn* bundle written by Jack Lucchetti and Stefano Balietti.

[2] **H**eteroskedastic **I**nstrumental variables **P**robit written by Jack Lucchetti

[3] Marginal effects for various qualitative choice models written by Allin Cottrell.

which yields:

```
                   Mean      Median        S.D.         Min         Max
autotime          49.35       51.40       32.43      0.2000       99.10
bustime           48.12       38.00       34.63       1.600       91.50
dtime           -0.1224     -0.7000       5.691      -9.070       9.100
auto             0.4762      0.0000      0.5118      0.0000       1.000
```

The proportion of people who drive around 47.6% and the average time differential is around $-.12$, with average *bustime* being less than *autotime*.

The model is estimated by least squares using the `--robust` option since a binary dependent variable is heteroskedastic. A new series is computed that takes the value 1 if the predicted probability of taking auto is above 50%. Incorrect prediction is also measured when the model predicts auto and the individual takes the bus. The mean of this series measures the relative frequency of incorrect predictions.

```
1 m1 <- ols auto const dtime --robust
2 series y_pred = $yhat>0.5
3 series incorrect = abs(auto-y_pred)
4 summary incorrect --by=auto --simple
5
6 scalar correct = $nobs-sum(abs(auto-y_pred))
7 printf "The number correct predictions =\
8 %g out of %g commuters\n", correct, $nobs
9 t_interval_m($coeff,$vcv,$df,.95)
```

The estimated probit model from line 1 is:

m1: OLS, using observations 1–21
Dependent variable: auto
Heteroskedasticity-robust standard errors, variant HC1

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 0.484795 | 0.0712037 | 6.809 | 0.0000 |
| dtime | 0.0703099 | 0.00850764 | 8.264 | 0.0000 |

$R^2$ 0.611326 Adjusted $R^2$ 0.590869

The coefficient on *dtime* is positive (significantly so at 5%), which indicates that the larger the time differential, the more likely a person is to take a trip by automobile. The simple summary statistics are computed using the --by=auto option, which allows one to determine if incorrect predictions are similarly distributed across the choices.

```
auto = 0 (n = 11):
  Mean                        0.090909
  Minimum                      0.00000
  Maximum                       1.0000
  Standard deviation           0.30151
  Missing obs.                       0

auto = 1 (n = 10):
  Mean                         0.10000
  Minimum                      0.00000
  Maximum                       1.0000
  Standard deviation           0.31623
  Missing obs.                       0
```

From these we can determine that only 1 of 11 bus riders (1/11 = 0.090909) and 1 of 10 auto riders (1/10 = 0.10000) were incorrectly predicted.

The total number of correct predictions is computed in line 6 and equals 19/21. Finally, a 95% confidence interval for the coefficients is computed using a new user written function called t_interval_m, which is shown below.

```
The 95% confidence intervals (t-distribution)
        Lower  Estimate     Upper
b1     0.3358    0.4848    0.6338
b2     0.0525    0.0703    0.0881
```

The t_interval_m function uses accessors from an estimated model to construct confidence intervals for the parameters. It is a generalization of the t_interval function (see section 3.1) that has been heavily used in this manual. The function is:

```
function matrix t_interval_m (matrix b "Coefficients",
      matrix v "Variance-covariance matrix",
      int df "Degrees-of-freedom",
      scalar p "Coverage probability for CI")

    scalar alpha = (1-p)                 # Convert p to alpha
    matrix c = critical(t,df,alpha/2)   # alpha/2 critical value
    matrix se = sqrt(diag(v))           # standard errors
    matrix lb = b - c*se                # lower bound
    matrix ub = b + c* se               # upper bound
    matrix result = b ~ se ~ lb ~ ub    # put into matrix

    cnameset(result, "Estimate StdErr (Lower, Upper) ")
    rnameset(result, "b")
    printf "\nThe %2g%% confidence intervals\
(t-distribution)\n%10.4f\n", p*100, result
    return result
end function
```

It takes four arguments. The first is a matrix that contains the model's estimates. Next is a matrix containing the estimated variance-covariance. Then, the available degrees-of-freedom for the $t$-ratio, and finally, the desired coverage probability for the intervals. Output is printed to the screen and saved as a matrix for further manipulation, if desired. One possible improvement would be to use the actual variable names for the parameters, but I'll save this for later.

From the output, the 95% confidence interval for $\beta_2$ is $(0.0525, 0.0881)$. This function does not replace the univariate `t_interval` function entirely. There are still times when one has a single statistic for which a confidence interval is desired. This is where `t_interval` comes in handy.

Finally, the probability of driving is computed based on a 10 minute time differential.

```
scalar pr = $coeff(const)+$coeff(dtime)*1
printf "\n The predicted probability of auto travel if public\
transportation\n takes 10 minutes longer = %.4f \n", pr
```

The result is:

```
The predicted probability of auto travel if public transportation
takes 10 minutes longer = 0.5551
```

The goodness-of-fit in a linear model is measured by $R^2$.

```
printf "\n R2 = %.4f \n", $rsq
```

which in this model is 0.59.

## 16.3 Probit and Logit

In this section other binary choice models are examined and include probit and logit. Using examples the models are estimated and their margnial effects are considered. Also, routines for calculating the standard errors of the marginal effects are offered in a succession of increasing generality. The pentulitmate versions are based on a **gretl** function package called *lp-mfx* that is written by **gretl**'s own Allin Cottrell and available through the **gretl** function package server.

The **probit** statistical model expresses the probability $p$ that $y_i = 1$ as a function of the independent variables.

$$P[(y_i|x_{i2}, x_{i3}) = 1] = \Phi(\beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3}) \tag{16.3}$$

where $\Phi$ is the cumulative normal probability distribution (cdf). The argument inside $\Phi$ is linear in the parameters and is called the **index function**. $\Phi$ 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 uses the same format as for linear regression except the `probit` command replaces `ols`.

### Example 16.3 in *POE5*

This example is somewhat contrived to demonstrate algebraically how maximum likelihood works. A three observation dataset is created. The data have two variables, $y$ and $x$ and is shown below:

```
obs            y            x

1              1          1.5
2              1          0.6
3              0          0.7
```

To create this in **gretl** is simple. Create an empty dataset with three observations. Then initialize the two series as in line 2.

```
1 nulldata 3
2 series y x
```

Then, populate the observations using index commands as shown below:

```
3 series y[1]=1
4 series y[2]=1
5 series y[3]=0
6 series x[1]=1.5
7 series x[2]=.6
8 series x[3]=.7
```

Now estimate the parameters of

$$P[(y_i|x_i = 1] = \Phi(\beta_1 + \beta_2 x_i)$$

using `probit`. Be sure to include a constant.

```
probit y const x
```

The results are:

```
                Model 2: Probit, using observations 1–3
                        Dependent variable: y
                   Standard errors based on Hessian

            Coefficient   Std. Error        z        Slope*

   const     −1.15254      2.34506      −0.4915
   x          1.89162      2.91513       0.6489    0.625394

   Log-likelihood    −1.593971        *Evaluated at the mean

Number of cases ‘correctly predicted’ = 1 (33.3 percent)
Likelihood ratio test: χ2(1) = 0.631 [0.4269]
```

The MLE of $\beta_2 = 1.891$ which is not significantly different from zero at 5%. The log-likelihood is $-1.594$ and only one of the three cases was correctly predicted by the estimated model.

### Example 16.4 in *POE5*

In this example, maximum likelihood is used to estimate the parameters of a probit model of the decision to travel by car or bus. As in the LPM model in equation 16.2, 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 (*bustime* − *autotime*).

$$Pr[auto_i = 1] = \Phi(\beta_1 + \beta_2 dtime_i) \tag{16.4}$$

```
open "@workdir\data\transport.gdt"
list xvars = const dtime
m2 <- probit auto xvars
t_interval_m($coeff,$vcv,$df,.95)
```

The probit MLE is computed in line 3 using a variables list that includes the regressors (from line 2). The `t_interval_m` command is used to obtain 95% confidence intervals for both the constant, $\beta_1$, and $\beta_2$. The results appear below:

$$\widehat{\text{auto}} = \underset{(0.3992)}{-0.06443} + \underset{(0.1029)}{0.3000}\,\text{dtime}$$

$$T = 21 \quad \bar{R}^2 = 0.4381 \quad \hat{\sigma} = 0.32734$$

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

The confidence intervals are:

```
The 95% confidence intervals (t-distribution)
     Estimate    StdErr   (Lower,    Upper)
b1    -0.0644    0.3992   -0.9001    0.7712
b2     0.3000    0.1029    0.0847    0.5153
```

The interval for $\beta_2$, which is centered at 3.000 is $(0.0847, 0.5153)$ and does not include zero.

Next, the predicted value of the index when the time differential is 20 minutes is computed and the predicted probability of driving an auto is computed. Also, the marginal effect of increasing the time differential by 10 minutes (*dtime*=3) is computed, based on the assumption that *dtime* is continuous.[4] The `cnorm` function in line 5 computes the value of the cumulative standard normal distribution, $\Phi(\cdot)$, evaluated at its argument and the `dnorm` function in line 6 computes the pdf of the standard normal distribution, $\phi(\cdot)$, evaluated at its argument.

```
5   scalar p1=cnorm($coeff(const))
6   scalar i_20 = $coeff(const)+$coeff(dtime)*2
7   scalar d_20 = dnorm(i_20)*$coeff(dtime)
8   printf "\n The value of the index for dtime = 20 minutes is %6.4f\n\
9   The predicted probability of driving is = %6.4f\n\
10  The marginal effect on probability of driving is %6.4f \n",\
11      i_20, cnorm(i_20), d_20
```

which produces:

```
 The value of the index for dtime = 20 minutes is 0.5355
 The predicted probability of driving is = 0.7039
 The marginal effect on probability of driving is 0.1037
```

The probability of using an auto when its 20 minutes faster is estimated to be 0.704. The marginal effect on the probability of increasing that by 10 minutes, computed in line 3 of the script above, is 0.1037.

[4] $\partial p/\partial dtime = \phi(\beta_1 + \beta_2\, dtime) * \beta_2$

Finally, to compare the marginal effect computed based on calculus with one based on a discrete change, the probability of choosing auto if the time differential increases to 30 minutes is computed.

```
scalar i_30 = $coeff(const)+$coeff(dtime)*3
printf "\n The predicted probability of driving if dtime = 30\
minutes is %6.4f\n", cnorm(i_30)
printf "\n The difference in probability is %6.4f\n",\
       cnorm(i_30)-cnorm(i_20)
```

which produces:

```
The predicted probability of driving if dtime = 30 minutes is 0.7983
The difference in probability is 0.0944
```

The probability increased by $0.7983 - 0.704 = 0.0944$, which is slightly less than 0.1037 predicted under the assumption that *dtime* is continuous.

Of course, the probit MLE can be summoned from the pull-down menus using **Model>Limited dependent variable>Probit>Binary**. The dialog box (Figure 16.1) looks very similar to the one for linear regression, except it has more options, e.g., to view the details of the iterations. Fill in the boxes for the dependent and independent variables, select the desired options, and click OK.

### 16.3.1 Marginal Effects and Average Marginal Effects

#### Example 16.5 in *POE5*

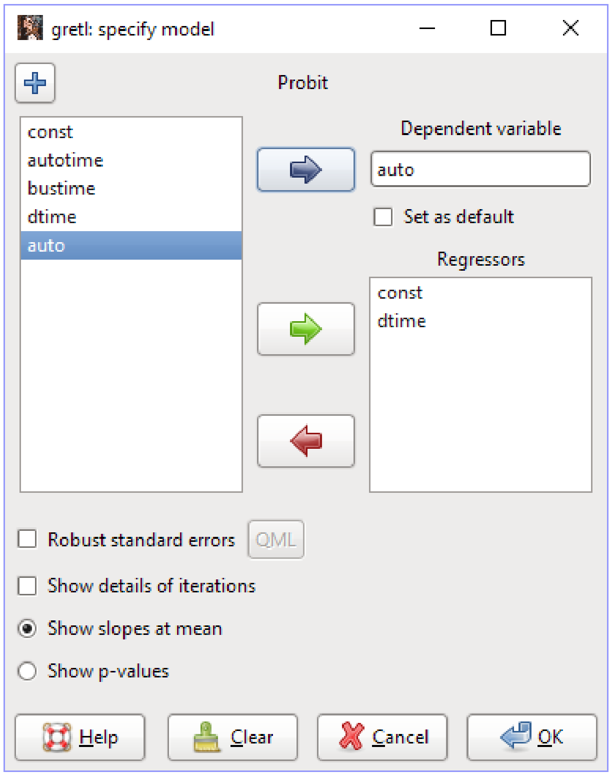
The marginal effect of a change in $x_{ij}$ on the probability of the choice, $P_i$, is

$$\frac{\partial P_i}{\partial x_{ij}} = \phi(\beta_1 + \beta_2 x_{i2} + \beta_3 x_{i3})\beta_j \tag{16.5}$$

where $\phi(\cdot)$ 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 from the preceding section the marginal effect of increasing public transportation time by one unit was computed. Given that travel via public transport currently takes 20 (*dtime*=2) minutes longer than auto, the estimated marginal effect was

$$\frac{\partial P_i}{\partial dtime_i} = \phi(\hat{\beta}_1 + \hat{\beta}_2 dtime_i) = \phi(-0.0644 + 0.3000 \times 2)(0.3000) = 0.1037 \tag{16.6}$$

Marginal effects for indicator variables require a different approach. For an indicator regressor, the probability is computed for each of its states (1 and 0), holding the values of the other variables constant at selected values. The other variables may be evaluated at their sample means or at representative points. More will be said about this shortly.



gretl: specify model

Probit

const
autotime
bustime
dtime
auto

Dependent variable

auto

□ Set as default

Regressors

const
dtime

□ Robust standard errors

□ Show details of iterations

● Show slopes at mean

○ Show p-values

Figure 16.1: Use **Model>Limited dependent variable>Probit>Binary** to open the Probit model's dialog box.

**Average Marginal Effects (AME)** A popular approach is to evaluate marginal effects at each sample point and to average them. These are referred to as **average marginal effects**. The average marginal effect of a change in $x_{ij}$ on $P_i$ is

$$\widehat{AME}_j = \frac{1}{N}\sum_{i=1}^{N}\phi(\hat{\beta}_1 + \hat{\beta}_2 x_{i2} + \hat{\beta}_3 x_{i3})\hat{\beta}_j \tag{16.7}$$

It is also common to evaluate the marginal effects at the means of the data. That would be

$$\widehat{ME}_j = \phi(\hat{\beta}_1 + \hat{\beta}_2 \bar{x}_2 + \hat{\beta}_3 \bar{x}_3)\hat{\beta}_j \tag{16.8}$$

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 the 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 get a good idea of the (average) marginal effects 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. To compute the AME for an increase in travel time:

```
1 open "@workdir\data\transport.gdt"
2 list x = const dtime
3 probit auto x
4 matrix b = $coeff
5 series me = dnorm(lincomb(x,b))*b[2]
6 scalar amf = mean(me)
7 printf "\n The average marginal effect for change in dtime =\
8 %6.4f\n", amf
9 summary me --simple
```

The data are loaded and a list of independent variables is created. The model is estimated via `probit`. Note, it is possible to add the `--robust` option to `probit`, but there is some debate about what this accomplishes. What it **does not do** is to make the MLE robust to heteroskedasticity, but *may* make is robust with respect to choice of likelihood function. This is referred to as **quasi-maximum likelihood** (QML).

A more general version of this technique is provided below. This simple one is used to illustrate what is being done to compute an AME. Line 5 is general in one sense. It generates marginal effects for $x_{i2}$ from any probit model that contains at least one variable other than a constant. The index 2 in `b[2]` refers to the element of `b` for which a marginal effect is desired. The average of these is computed in line 6 and the result printed from lines 7 and 8.

```
 The average marginal effect for change in dtime = 0.0484
```

Simple summary statistics reveal that the average of the marginal effects in the sample is 0.0484. The smallest is 0.0024738 and the largest 0.11526. That is a fairly large range, suggesting that our measurement is imprecise.

To facilitate the computation of the AMEs, I have written a function that will compute them for an arbitrary probit or logit model. The function is called `ame_binary` and it requires three inputs to compute the marginal effects. First, it needs the logit or probit parameter estimates. Then, it needs the list of explanatory variables from the model. Finally, it must include an indicator of whether the model was estimated by logit or probit. The `dist` argument is used to determine the latter. It can be read from the model's accessor using the `$command`. It should be equal to 2 for probit (line 5) and 1 for logit. 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. The obvious problem with this particular function is that a marginal effect is computed for the constant, which doesn't make sense. We address this below by using a more sophisticated function from the **gretl** function package server called *lp-mfx*.

```
1 function matrix ame_binary(matrix *b "parameter estimates",
2       list x "Variables list",
3       int dist[1:2:2] "distribution" )
```

```
 4
 5     matrix p = lincomb(x, b)           # The index function
 6     matrix d = (dist==1) ? exp(-p)./(1.+exp(-p)).^2 : dnorm(p)
 7     matrix ame_matrix = d*b'
 8     cnameset(ame_matrix, x)            # add column names
 9     matrix amfx = meanc(ame_matrix)    # find the means
10     cnameset(amfx, x)                  # add the column names to amfx
11     printf "\n Average Marginal Effects (AME):\
12       \n Variables: %s\n%12.4g \n", varname(x), amfx
13     return amfx
14 end function
```

The function is quite simple and make only four computations. However, it is made slightly more complicated by using a **pointer**. The asterisk in front of b, i.e., *b, identifies b as a pointer; it is not necessary to use a pointer in this case and removing it will have no effect on the results, provided the corresponding & used to retrieve its contents is removed as well. It is introduced here merely to illustrate its use. Pointers are often used to save precious computer memory or to make functions more modular.

Since a pointer identifies the parameter vector in the function (matrix *param), an ampersand (&) must be place in front of the parameter matrix being passed into the function, i.e., ame_binary(&b, x, dist). 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 retrieve the object using its memory address when called. Using pointers reduces the number of objects that are stored in memory, 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 programs more modular.[5] See section 13.4 of the Gretl Users Guide Cottrell and Lucchetti (2018) for more details.

Returning to the script, line 5 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 variables in the list. The result from lincomb can be a series, or in this case, a matrix. So for instance, suppose $X$ is $n \times k$ and contains variables and $\beta$ is a $k \times 1$ parameter vector. The linear combination $X\beta$ is $n \times 1$. Line 6 computes the matrix that contains all of the marginal effects. The meanc function in line 9 computes the column means of the matrix (AME), which gets printed in lines 11 and 12. 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 (or logit) model, and save the coefficients using matrix coef=$coeff. Line 4 uses the accessor $command to determine whether the preceding regression was estimated by probit or logit. Given the variable list and the parameter estimates, you can call the function as in line 6 of the script below.

[5] If you do not want what is being pointed at to change, you can declare it to be a constant using const.

```
1 open "@workdir\data\transport.gdt"
2 list x = const dtime
3 probit auto x --quiet
4 matrix b = $coeff
5 scalar dist = ($command == "logit")? 1 : 2
6 matrix me_probit = ame_binary(&b, x, dist)
```

The function could be further refined for error checking and to remove the constant from the output. Still, it serves our purpose in this manual.

The function `ame_binary(&coef, x, dist)` in line 6 prints the AME to the screen. To save the matrix output from the function, use:

```
matrix me_probit = ame_binary(&coef, x, dist)
```

and the result will be saved to `me_probit`. The result for the travel time example is:

```
Average Marginal Effects (AME):
Variables: const,dtime
      const       dtime
    -0.0104     0.04841
```

The average marginal effect of a 10 minute (*dtime* = 1) increase in travel time is 0.0484. The AME of the constant is not very meaningful. Conceptually, this is how much $\hat{\beta}_1$ would change if its variable were coded with a number infinitesimally larger than 1 instead of just 1. In a probit model, doubling the constant variable to 2 reduces $\hat{\beta}_1$ by half.

### 16.3.2 Standard Errors and Confidence Intervals for Marginal Effects

Obtaining confidence intervals for the marginal effects (and the AME) is straightforward. To estimate the standard error of the marginal effect, the delta method is used. This method to find the variance of functions of parameters was discussed in section 5.6.1. Take a look at this section again if a refresher is warranted (page 146).

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 or variance of the statistic. 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**. The delta method requires the partial derivatives of the function in question with respect its parameters. Not surprisingly, the fdjac function requires two arguments: a function and a vector of parameters.

The first step to use this method in **gretl** is to define the function to be differentiated. Then apply fdjac to that function. Adding lines 7-9 to the script produces the AMEs based on the function ame_binary is:

```
 7 matrix jac = fdjac(b, ame_binary(&b, x , dist))
 8 matrix variance = qform(jac, $vcv)
 9 matrix se = sqrt(diag(variance))
10
11 printf "\n The average marginal effects:\n%10.4f\
12 delta estimated standard errors: \n%10.4f \n", amfx, se'
13
14 # confidence interval for average mfx
15 t_interval_m(amfx',variance,$df,.95)
```

This produces:

```
 The average marginal effects:
     const      dtime
   -0.0104     0.0484
 delta estimated standard errors:
    0.0648     0.0034
```

The estimated standard error for the AME of *dtime* is 0.0034. The t_interval_m routine produces:

```
The 95% confidence intervals (t-distribution)
     Estimate     StdErr    (Lower,     Upper)
b1    -0.0104     0.0648   -0.1459     0.1251
b2     0.0484     0.0034    0.0413     0.0556
```

The first six lines of the script, found on page 552, are standard. The data are opened, the variable list created, the probit model is estimated using probit, and a matrix is used to hold the coefficients. Line 5 captures and evaluates the estimator used and the binary AMEs are created using our ame_binary function.

Given a function that computes the AMEs, the fdjac function is used to obtain numerical derivatives with respect to the parameters, b. Since we used pointers in the function, the ampersand needs to precede the coefficient and scalar inputs. The quadratic form used in the delta method is computed in line 8 using qform. qform(x,A) computes $xAx^T$, which is used to compute the variance expression in equation (5.15). The square roots of the diagonal are saved as standard errors, se. Thus, only three lines are used to compute standard errors for a nonlinear function like ame_binary.

This script produces:

```
The average marginal effects:
    const     dtime
  -0.0104    0.0484
delta estimated standard errors:
   0.0648    0.0034
```

The average marginal effect of a 1 unit change in *dtime* = 0.0484 with standard error 0.0034. The 95% confidence interval for the AME can be computed using our t_interval_m function:

```
t_interval_m(amfx',variance,$df,.95),
```

which yields:

```
The 95% confidence intervals (t-distribution)
     Estimate    StdErr   (Lower,    Upper)
b1    -0.0104    0.0648   -0.1459    0.1251
b2     0.0484    0.0034    0.0413    0.0556
```

The 95% confidence interval for $\beta_2$ is centered at 0.0484 and is (0.0413, 0.0556).

**Marginal effects at representative values (MER)** An alternative to the AME is to evaluate the marginal effect of a change in $x_j$ on the probability that $y = 1$ at as specific point. It can be a points of particular interest, the means of $x$, or quantiles of $x$. Of course, to make the MER more useful, standard errors should be computed.

As seen in the preceding paragraphs, what is needed is a function that computes the MER. One is provided in the script below:

```
function scalar me_at(matrix *param "parameter estimates",
      matrix xx "Representative Point",
      scalar q "Parameter of interest",
      int modl[1:2:2] "distribution" )
    # Marginal effects at a point -- continuous variables only
    scalar idx = xx*param
    scalar d = (modl==1)? (exp(-idx)./(1.+exp(-idx)).^2)*param[q] :\
      dnorm(idx)*param[q]
    return d
end function
```

This is a another simple function, consisting of only two computations. The inputs are 1) a matrix of parameter estimates from a logit or probit model 2) a matrix (a vector actually) that contains the representative point at which the marginal effect will be computed. It should have a dimension that matches that of b, 1 × $k$. 3) a scalar that identifies the number of the coefficient in `b` for which the marginal effect is desired and 4) a scalar that identifies whether `probit` is used to estimate the parameters. If `modl` = 1, then `b` is estimated by logit. If not, then the function assumes it was estimated by `probit`. The function returns a scalar, which is the MER at point `x`. The useful conditional assignment operator is used to determine whether to populate `d` with the probit or logit MER based on the value of `modl`.

Once this function has been run, the standard errors and confidence interval is computed using a function we call MER.

```
1  function void MER (matrix *b "parameter estimates",
2        matrix covmat "Covariance",
3        matrix x "Representative Point",
4        int q "Parameter of interest",
5        int df "Degrees of Freedom",
6        int modl[1:2:2] "distribution")
7      # Std errors for Marginal effects at a point -- continuous vars only
8      scalar p = me_at(&b, x, q, modl)
9      matrix jac = fdjac(b, me_at(&b, x , q, modl))
10     matrix variance = qform(jac,covmat)
11     matrix se = sqrt(diag(variance))
12     scalar crit = critical(t,df,0.025)
13     matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se
14     if modl == 1
15         printf "Logit:\n"
16     else
17         printf "Probit:\n"
18     endif
19     printf "95%% t(%.2g) confidence interval for b%.g at\n x =\
20     %9.2g \n", df, q, x
21     cnameset(results, " Lower ME Upper StdError" )
22     printf "%10.4f\n", results
23 end function
```

This function returns nothing (void) when called, but prints the confidence interval and the estimated standard error to the screen. It requires the same inputs as `me_at`, plus the estimated covariance matrix from the model (`matrix covmat`), which is needed to use the delta method, and the degrees of degrees-of-freedom for the $t$ critical value (`scalar df`) used to provide a suitable critical value. Line 8 computes the MER at `x` and line 9 computes its numerical derivative at the same point. The variance is computed in line 10 using `qform` and the standard errors obtained as the square roots of its diagonal elements in line 11.

Finally, the critical value from the $t$-distribution is obtained and used to accumulate the results in a 1×4 matrix in line 13. Results include the CI lower bound, the center of the interval, the upper bound and the standard error. Feel free to exchange positions for these if desired (an alternative format is considered below).

To use this function execute:

```
1 open "@workdir\data\transport.gdt"
2 list x = const dtime
3 m1 <- probit auto x --quiet
4 matrix bp = $coeff
5 matrix xi = { 1, 2 }
6 scalar dist = ($command == "logit")? 1 : 2
7 MER(&bp,$vcv,xi,2,$df,dist)
```

For a commute differential of 20 minutes, the representative point is $x_i = 1, 2$. The constant and *dtime* are hard coded in line 5. The coefficient input into MER is a pointer to the coefficient estimates saved as bp in line 4, so use the & prefix. Also, set q=2 in the fourth argument of MER since the desired marginal effect is for $\beta_2$, the coefficient for *dtime*.

The results from this are:

```
Probit:
95% t(19) confidence interval for b2 at
 x =        1.0000   2.0000

     Lower         ME     Upper  StdError
    0.0354     0.1037    0.1720    0.0326
```

For a time differential of 20 minutes the estimated marginal effect (extra 10 minutes) on the probability of choosing auto is 0.1037. The confidence interval is rather wide, (0.0354, 0.1720), but excludes zero.

**Marginal effects at the means** If particular values of interest are difficult to identify, it is common to use the sample means from the data to serve as the "representative point." This requires only that line 5be replaced by

```
5 matrix xi = { 1, mean(dtime) }
```

Calling the function as in line 7 produces:

```
Probit:
95% t(19) confidence interval for b2 at
```

```
x =        1.0000  -0.1224

    Lower         ME     Upper  StdError
   0.0333    0.1191    0.2049    0.0410
```

Notice that the average time differential is as found before, −0.1224, i.e., 1.2 minutes. The marginal effect is estimated to be 0.1191, with the 95% confidence interval of $(0.0333, 0.2049)$.

### 16.3.3 Using lp-mfx

There is also a very useful user written function that can be found on the **gretl** function package server called *lp-mfx.gfn*. This set of routines, written by **gretl**'s own Allin Cottrell, calculates marginal effects and associated statistics (standard errors, $z$-values and $p$-values) for logit and probit models. It includes facilities for binary logit and probit, ordered logit and probit, and multinomial logit.

The package provides a graphical interface that appears under the **Analysis** menu in **gretl**'s model window; the GUI is hard-wired to produce marginal effects at the sample means of all the regressors. However, the package includes a set of functions that can be used to calculate marginal effects at any vector of regressor values. Thus, it includes a function similar to `me_at` that can be coaxed into computing MER and their standard errors via out MER function. This will be demonstrated below within the context of multinomial logit.

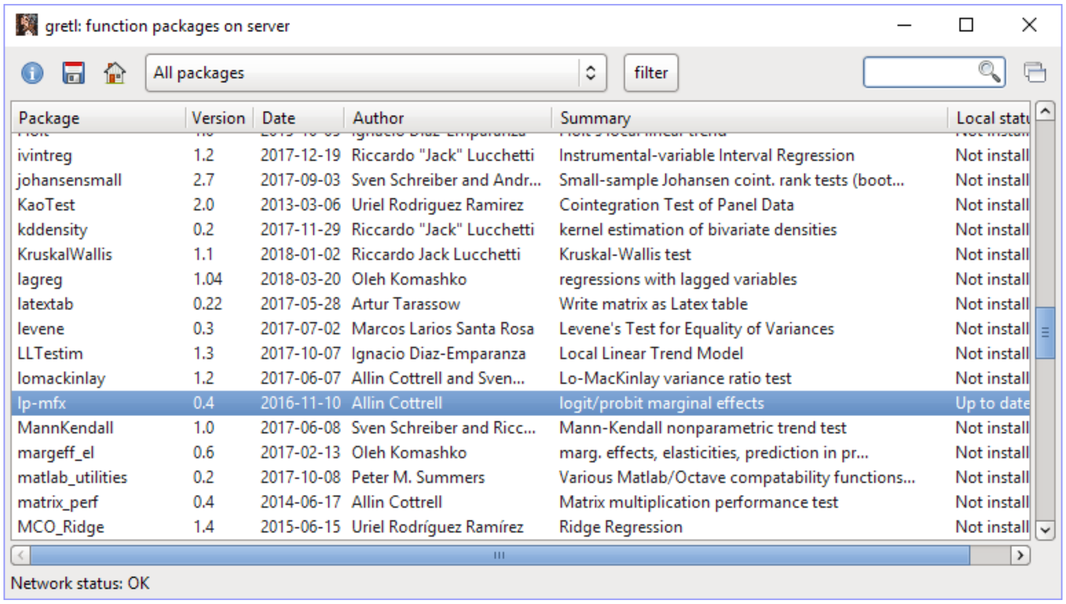
To illustrate the base output from this very useful set of functions, it is necessary to download and install the function package. This can be done, provided your system is connected to the internet, via the function package server. Choose **File>Function Packages>On server** to open the list of available packages shown in Figure 16.2. The *lp-mfx* package is highlighted. To install it, click on the diskette icon on the menu bar or right-click and choose **install**.

To estimate the marginal effects at the mean, use the following script:

```
include lp-mfx.gfn
m1 <- probit auto x
scalar dist = ($command == "logit")? 1 : 2
binary_mfx(auto, $xlist, $coeff, $vcv, $sample, dist)
```

The first line loads the package. The model is estimated and the `$command` accessor is used to determine which model is estimated. One of the benefits of this package over the ones I've written, is the inclusion of error checking in the routines. If you haven't estimated the preceding model by logit or probit, it will throw and error message telling you so.

Running this script yields:



| Package | Version | Date | Author | Summary | Local statu |
|---|---|---|---|---|---|
| ivintreg | 1.2 | 2017-12-19 | Riccardo "Jack" Lucchetti | Instrumental-variable Interval Regression | Not install |
| johansensmall | 2.7 | 2017-09-03 | Sven Schreiber and Andr... | Small-sample Johansen coint. rank tests (boot... | Not install |
| KaoTest | 2.0 | 2013-03-06 | Uriel Rodriguez Ramirez | Cointegration Test of Panel Data | Not install |
| kddensity | 0.2 | 2017-11-29 | Riccardo "Jack" Lucchetti | kernel estimation of bivariate densities | Not install |
| KruskalWallis | 1.1 | 2018-01-02 | Riccardo Jack Lucchetti | Kruskal-Wallis test | Not install |
| lagreg | 1.04 | 2018-03-20 | Oleh Komashko | regressions with lagged variables | Not install |
| latextab | 0.22 | 2017-05-28 | Artur Tarassow | Write matrix as Latex table | Not install |
| levene | 0.3 | 2017-07-02 | Marcos Larios Santa Rosa | Levene's Test for Equality of Variances | Not install |
| LLTestim | 1.3 | 2017-10-07 | Ignacio Diaz-Emparanza | Local Linear Trend Model | Not install |
| lomackinlay | 1.2 | 2017-06-07 | Allin Cottrell and Sven... | Lo-MacKinlay variance ratio test | Not install |
| lp-mfx | 0.4 | 2016-11-10 | Allin Cottrell | logit/probit marginal effects | Up to date |
| MannKendall | 1.0 | 2017-06-08 | Sven Schreiber and Ricc... | Mann-Kendall nonparametric trend test | Not install |
| margeff_el | 0.6 | 2017-02-13 | Oleh Komashko | marg. effects, elasticities, prediction in pr... | Not install |
| matlab_utilities | 0.2 | 2017-10-08 | Peter M. Summers | Various Matlab/Octave compatability functions... | Not install |
| matrix_perf | 0.4 | 2014-06-17 | Allin Cottrell | Matrix multiplication performance test | Not install |
| MCO_Ridge | 1.4 | 2015-06-15 | Uriel Rodríguez Ramírez | Ridge Regression | Not install |

Network status: OK

Figure 16.2: Many user written function packages are available on the **gretl** function package server.

```
Binary probit marginal effects
(evaluated at means of regressors)

auto = 1, Pr = 0.4597

             dp/dx        s.e.           z        pval        xbar
dtime      0.11907    0.040998      2.9042   0.0036817    -0.12238
```

The return for `binary_mfx` is a **bundle**. To save the bundle as an icon in the session window, one can use:

```
bundle b1 = binary_mfx(auto, $xlist, $coeff, $vcv, $sample, dist)
lp_mfx_print(&b1)
```

Notice the special `lp_mfx_print` command is used to print the results to the screen.

There are other advantages to using this function. 1) Notice that no marginal effect is estimated for the constant. Since it is not a meaningful statistic, this is desirable. 2) The underlying function that computes the marginal effect will detect whether the explanatory variable is discrete or continuous. This is an important advantage since it means that separate routines are not required to compute marginal effects (or standard errors). 3) The function computes MERs for all of the variables in the model. The value of this becomes obvious in Example 16.6 below, which includes more than one regressor.

To make use of these features the MER function will be revised to use the `lp-mfx` function that computes marginal effects at representative values. This function is called `MER_lpmfx` and it uses a function from *lp-mfx* that computes marginal effects for probit and logit models.

The `binary_dp_dx` function takes four arguments.

```
function matrix binary_dp_dx (matrix b "parameter estimates",
    list XL "list of regressors",
    matrix x "vector of x-values",
    int dist[1:2:1] "distribution" \
    {"logit", "probit"})
```

The main changes to `MER` occur in lines 2, 9 and 10 below. The regressor list is added to the inputs in line 2. In lines 9 and 10 the MERs are computed using `binary_dp_dx` and the Jacobian is computed. The printing statements that follow are modified as well, with the addition of row and column names for the results.

```
1  function void MER_lpmfx (matrix b "parameter estimates",
2        list  XL "list of regressors",
3        matrix covmat "Covariance matrix",
4        matrix x_at "Representative point",
5        int dist[1:2:1] "distribution",
6        int df "degrees-of-freedom")
7      # The MER function to be used with lp-mfx.gfn
8      # available from gretl’s function package server
9      matrix me = binary_dp_dx(b, XL, x_at, dist)
10     matrix jac = fdjac(b, binary_dp_dx(b, XL, x_at, dist))
11     matrix variance = qform(jac,covmat)
12     matrix se = sqrt(diag(variance))
13     matrix results = me’ ˜ se
14     if dist == 1
15         printf "Logit:\n"
16     else
17         printf "Probit:\n"
18     endif
19     scalar crit = critical(t,df,0.025)
20     matrix results = (me’-crit*se) ˜ me’ ˜ (me’+crit*se) ˜ se
21     cnameset(results, "Lower ME Upper StdErr")
22     rnameset(results, XL[2:nelem(XL)])
23     cnameset(x_at, XL )
24     printf "Representative Point\n%11.2g\n95%% CI for MER\n%10.4g\n",\
25     x_at, results
26 end function
```

The function call (line 6 below), including the model estimation is:

```
open "@workdir\data\transport.gdt"
list x = const dtime
probit auto x --quiet
scalar dist = ($command == "logit")? 1 : 2
matrix x_at = { 1 , 2}
MER_lpmfx($coeff,$xlist,$vcv,x_at,dist,$df)
```

The output shows us:

```
Probit:
Representative Point
      const      dtime
          1          2

95% CI for MER
           Lower         ME      Upper     StdErr
dtime    0.03537     0.1037      0.172    0.03264
```

The estimator is identified as being produced by probit and the representative point is given. The table containing the confidence intervals and standard errors now has a variable label as identifier. All-in-all this works quite well and the output looks good too.

**Marginal probabilities at representative values** 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, which is again used as a template.

First, a function that computes probabilities that $y_i = 1$ is required. That function is called `p_binary`. It requires three inputs: a vector of coefficient estimates, a point at which the probability will be evaluated and a scalar to indicate which distribution to use. The function is:

```
function scalar p_binary(matrix b "parameter estimates",
      matrix x "Representative Point",
      int dist[1:2:2] "distribution" )
    # Computes the probability of a binary choice: 1 = logit
    scalar p = x*b                      # The index function
    scalar d = (dist==1) ? 1./(1.+exp(-p)) : cnorm(p)
    return d
end function
```

Note, if *dist=1* then the logit probabilities are returned. A function called `Probs` is composed that uses `p_binary` to compute the delta method standard errors, confidence intervals, and to print a table of results.[6]

Probs is given below:

```
1  function void Probs (matrix b "parameter estimates",
2        matrix covmat "Covariance",
3        matrix x "Representative Point",
4        scalar df "Degrees of Freedom",
5        int dist[1:2:2] "distribution")
6      # Function computes std errors of binary predictions
7      # Requires p_binary
8      scalar p = p_binary(b, x, dist)
9      matrix jac = fdjac(b, p_binary(b, x , dist))
10     matrix variance = qform(jac,covmat)
11     matrix se = sqrt(diag(variance))
12     scalar crit = critical(t,df,0.025)
13     matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se
14
15     if dist == 1
16         printf "Logit:\n"
17     else
18         printf "Probit:\n"
19     endif
20
21     printf "95%% t(%.2g) confidence interval for probability at\n\
22     x = %8.4f\n", df, x
23     cnameset(results, " Lower ME Upper StdError" )
24     printf "%10.4f\n", results
25 end function
```

In line 8 the p_binary function is used to produce the probability at the representative point. The derivative is take in line 9 with respect to the parameters and combined in line 10 as prescribed by the delta method.

The function is used in this example to compute the marginal effect on the probability of driving when the time differential is 30 minutes (*dtime=3*). The model is estimated by probit.

```
probit auto x --quiet
matrix x_at = { 1 , 3}
scalar dist = ($command == "logit")? 1 : 2
Probs($coeff,$vcv,x_at,$df,dist)
```

The output is:

[6] Another thing to note is that the use of pointers has been abandoned in favor of simplicity.

```
Probit:
95% t(19) confidence interval for probability at
 x =   1.0000  3.0000

     Lower         ME     Upper  StdError
    0.5000     0.7983    1.0966    0.1425
```

The probability of driving when *dtime*=3 is 0.7983 with standard error 0.1425. The 95% confidence interval with *dtime*=3 is (0.5000, 1.0966). Obviously, the upper bound is not feasible since probabilities cannot exceed 1.

### 16.3.4 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 substantive outcomes from the logit estimation are usually very similar to those of probit. The probability that individual $i$ chooses the alternative is

$$P_i = F(z_i) = \Lambda(z_i) = \frac{1}{1+e^{-z_i}} \tag{16.9}$$

$$z_i = \sum_{j=1}^{k} x_{ij}\beta_j \tag{16.10}$$

In logit the probability is modeled using $\Lambda(z_i)$ rather than $\Phi(z_i)$ as in the probit model.

In **gretl**, the `logit` command syntax is the same as that for probit.

```
logit

  Arguments: depvar indepvars
  Options: --robust (robust standard errors)
                 --cluster=clustvar (clustered standard errors)
                 --multinomial (estimate multinomial logit)
                 --vcv (print covariance matrix)
                 --verbose (print details of iterations)
                 --p-values (show p-values instead of slopes)
```

In the next example estimators of probit, logit and the LPM are compared. The models are estimated and marginal effects are computed using the functions from the preceding subsection.

#### Example 16.6

The model used for this example is soft drink choice where the dependent variable is equal to one if the buyer purchases Coke and is zero otherwise. This is modeled as a function of the ratio of the Coke price to Pepsi price, the presence of a Coke display (1 = yes) and the presence of a Pepsi display (1 = yes). The model is:

$$\Pr(Coke_i = 1) = \Phi(\beta_1 + \beta_2 pratio + \beta_3 disp\_coke + \beta_4 disp\_pepsi) \tag{16.11}$$

First, the model in equation (16.11) is estimated using each of the binary choice estimators and the session window is used to create a model table. This is facilitated by assigning the output to the icons `m1`, `m2`, and `m3` in the session window:

```
open "@workdir\data\coke.gdt"
list x = const pratio disp_pepsi disp_coke
m1 <- probit coke x --quiet
m2 <- logit coke x --quiet
m3 <- ols coke x --robust
```

The model table, which is constructed by dragging each of the model icons onto the model table icon in the session window (see page 15), is:

Dependent variable: coke

| | (1)<br>OLS | (2)<br>Probit | (3)<br>Logit |
|---|---|---|---|
| const | 0.8902** | 1.108** | 1.923** |
| | (0.06530) | (0.1900) | (0.3258) |
| pratio | −0.4009** | −1.146** | −1.996** |
| | (0.06037) | (0.1809) | (0.3146) |
| disp_pepsi | −0.1657** | −0.4473** | −0.7310** |
| | (0.03436) | (0.1014) | (0.1678) |
| disp_coke | 0.07717** | 0.2172** | 0.3516** |
| | (0.03393) | (0.09661) | (0.1585) |
| $n$ | 1140 | 1140 | 1140 |
| $R^2$ | 0.1201 | 0.0930 | 0.0949 |
| $\ell$ | −748.1 | −710.9 | −709.4 |

Standard errors in parentheses
* indicates significance at the 10 percent level
** 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 estimators. In logit and probit, the coefficients, signs are consistent with the direction of the marginal effects, either positive or negative. 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.

$$\begin{aligned}\tilde{\gamma}_{Logit} &\cong 4\hat{\beta}_{LPM}\\ \tilde{\beta}_{Probit} &\cong 2.5\hat{\beta}_{LPM}\\ \tilde{\gamma}_{Logit} &\cong 1.6\hat{\beta}_{Probit}\end{aligned}$$

So, $4(-0.4009) = -1.6036$ is fairly close to the estimate $-1.996$ for the `pratio` coefficient in the logit column. More importantly, there are closer similarities between the marginal effects implied by logit and probit. Their averages (AME) are very close to the corresponding coefficient in the linear probability model. One can expect them to become closer as sample size increases.

The first set of statistics computed are the AME from each of the models. This is easy for the LPM since the marginal effects are the same no matter what the value of $x$. For probit and logit it requires the use of the delta method to obtain consistent estimators of standard errors.

The computation of AME and their standard errors has been consolidated to the following function, which uses the `ame_binary` function to estimate the average marginal effects from the sample.

```
function matrix ame_cov (matrix b "parameter estimates",
      matrix covmat "Covariance",
      list x "Variables list",
      int dist[1:2:2] "distribution" )
    # Computes std errs for AME probit/logit
    # Requires ame_binary
    matrix amfx = ame_binary(&b, x, dist)
    matrix jac = fdjac(b, ame_binary(&b, x , dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    matrix results = amfx’ ~ se
    rnameset(results, "b")
    cnameset(results, "AME StdErr")
    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    printf "%10.4f\n", results
    return amfx|variance
end function
```

The `ame_cov` function accumulates the AMEs and their delta method variance covariance matrix into a $k+1 \times k$ matrix. This matrix provides perfect input into the function that computes confidence intervals based on the $t$-distribution. The columns of the output from `ame_cov` are the parameters of the model. The first row contains the coefficient estimates and the remaining $k \times k$ matrix is the estimated variance covariance.

The model is estimated and the inputs required for these functions are assigned to matrices and a scalar.

```
m1 <- probit coke x --quiet
matrix bp = $coeff
matrix covmat = $vcv
scalar dist = ($command == "logit")? 1 : 2
```

To produce a set of confidence intervals, separate the coefficients from the covariance and use the `t_interval_m` function as shown below.

```
matrix c=ame_cov(bp,$vcv,x,dist)
t_interval_m(c[1,]',c[-1,],$df,.95)
```

The indexing prowess of **gretl** is in evidence here. The $k+1 \times k$ matrix produced by the function is `c`. `c[1,]` pulls out the first row, all columns of the matrix `c`. It is transposed to form a $k \times 1$ vector of coefficients. `c[-1,]` takes `c` and $-1$ in the row position removes the first row. What is left is the variance-covariance. The rest of the inputs are familiar by now.

The result is:

```
The 95% confidence intervals (t-distribution)
     Estimate    StdErr    (Lower,     Upper)
b1     0.3961    0.0652     0.2682     0.5241
b2    -0.4097    0.0616    -0.5306    -0.2887
b3    -0.1599    0.0353    -0.2292    -0.0906
b4     0.0776    0.0343     0.0103     0.1450
```

Logit and probit MERs are also compared. The representative point considered has the price of Coke 10% higher than Pepsi (pratio = 1.1) and neither is displayed. For probit:

```
matrix x_at = { 1, 1.1, 0, 0}
probit coke x
scalar dist = ($command == "logit")? 1 : 2
MER_lpmfx($coeff,$xlist,$vcv,x_at,dist,$df)
```

which produces:

```
Probit:
Representative Point
      const     pratio disp_pepsi  disp_coke
          1        1.1          0          0

95% CI for MER
                Lower         ME     Upper    StdErr
    pratio    -0.5898   -0.4519    -0.314   0.07028
disp_pepsi    -0.2346   -0.1651  -0.09554   0.03544
 disp_coke    0.01102   0.08639    0.1618   0.03842
```

Re-estimating the model using logit:

```
1 logit coke x
2 scalar dist = ($command == "logit")? 1 : 2
3 MER_lpmfx($coeff,$xlist,$vcv,x_at,dist,$df)
```

produces:

```
Logit:
Representative Point
      const     pratio disp_pepsi  disp_coke
          1        1.1          0          0

95% CI for MER
                Lower         ME     Upper    StdErr
    pratio    -0.6376   -0.4898    -0.342   0.07532
disp_pepsi    -0.2329    -0.164   -0.0952   0.03509
 disp_coke   0.009974   0.08747     0.165    0.0395
```

Do not forget to recreate the scalar dist in line 2 since it must be reinitiated each time a new estimator is used. Comparing the two sets of results, the MER for the *pratio* coefficient estimated by probit is $-0.4519$, and the 95% confidence interval is $(-0.5898, -0.314)$. From logit we get an estimate of $-0.4898$ and an interval of $(-0.6376, -0.342)$. The two sets of results are very similar.

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 $2 \times 2$ 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, the correlations between the predictions of the three estimators are high as shown below:

```
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
```

The correlations exceed 0.99 and are significant at 5%.

### 16.3.5 Hypothesis Tests

#### Example 16.7 in *POE5*

In this example several hypotheses are tested using Wald tests. In **gretl** these are done using the `restrict` block, possibly with the `--quiet` option. One-sided $t$-tests can be constructed manually using accessors and these are displayed below as well.

**One-sided *t*-test** Tests of significance were explored in section (5.4.3). Based on the soft drink model explored in equation (16.11), the hypothesis that the presence of a Coke display increases the probability of a Coke purchase. Parametrically, this is expressed as $H_0$: $\beta_3 \leq 0$ versus $H_1$: $\beta_3 > 0$. First, load the data and estimate the model by probit. Then form the $t$-ratio as a scalar and print the $t$-statistic and its one-sided $p$-value to the screen. The script is:

```
open "@workdir\data\coke.gdt"
list x = const pratio disp_pepsi disp_coke
probit coke x
```

```
# H1 Test of significance
scalar tv = $coeff(disp_coke)/$stderr(disp_coke)
printf "Ho: b3 = 0    Ha: b3>0\n \
  t  = %.4f\n \
  p-value = %.4f\n", \
  tv, pvalue(t,$df,tv)
```

Even though the example is rudimentary, the extra care taken to produce the output can pay off when your programs experience a long layoff between uses.

```
Ho: b3 = 0    Ha: b3>0
    t  = 2.2481
    p-value = 0.0124
```

Here, the $p$-value is less than 5% and we conclude that the display helps to sell more Coke.

**Two-sided *t*-test** For this hypothesis $H_0$: $\beta_3 = 0$ versus $H_1$: $\beta_3 \neq 0$. The same statistic from the one-sided test is used, but a different $p$-value is computed. In addition, the 5% critical value from the $t$-distribution is computed and displayed.

```
printf "Ho: b3 = 0    Ha: b3 != 0\n \
  t  = %.4f\n \
  p-value = %.4f\n", \
  tv, 2*pvalue(t,$df,abs(tv))
```

The results are:

```
Ho: b3 = 0    Ha: b3 != 0
    t  = 2.2481
    p-value = 0.0248

The 5% critical value from the t(1136) is 1.9621
```

Note that the $p$-value doubles, but $\beta_3$ is significantly different from zero at 5%. The $t$-ratio exceeds the 5% critical value from the $t(1136)$ distribution.

The same result can be obtained using a `restrict` block.

```
restrict --quiet
    b[disp_coke]=0
end restrict
```

The results are:

```
Restriction:
 b[disp_coke] = 0

Test statistic: chi^2(1) = 5.05403, with p-value = 0.0245687
```

Asymptotically, this is exactly equivalent to the $t$-test, since as $n$ extends to infinity, $t_n^2 \rightarrow \chi^2(1)$.

**Economic Hypothesis** The hypothesis that the Coke and Pepsi displays have an equal but opposite effect on the probability of buying Coke is to be tested. That is,

$$H_0\colon \beta_3 + \beta_4 = 0 \quad H_1\colon \beta_3 + \beta_4 \neq 0 \tag{16.12}$$

If a store has both displays, the net effect on Coke purchases is hypothesized to be zero.

As a two-sided alternative, the simplest thing to do is use the `restrict` statement as shown below:

```
probit coke x --quiet
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:

$$H_0\colon \beta_3 = 0 \text{ and } \beta_4 = 0 \quad H_1\colon \beta_3 \neq 0 \text{ or } \beta_4 \neq 0 \tag{16.13}$$

The **gretl** code is

```
probit coke x
restrict --quiet
    b[3]=0
    b[4]=0
end restrict
printf "The 5%% critical value from the chi-square(2) is %.4f\n",\
    critical(C,2,.05)
```

This statistic will have an $\chi^2(2)$ 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.

**Overall regression significance** For this hypothesis, the null is for all parameters other than the constant to be jointly zero.

```
probit coke x
restrict --quiet
    b[2]=0
    b[3]=0
    b[4]=0
end restrict
printf "The 5%% critical value from the chi-square(3) is %.4f\n",\
critical(C,3,.05)
```

This statistic will have an $\chi^2(2)$ distribution if the null hypothesis is true. The outcome in **gretl** is:

```
Restriction set
 1: b[pratio] = 0
 2: b[disp_pepsi] = 0
 3: b[disp_coke] = 0

Test statistic: chi^2(3) = 132.54, with p-value = 1.53304e-028
```

According to this result, the model is significant at 5%.

### Example 16.8 in *POE5*

Since probit and logit are estimated via maximum likelihood, you can also perform a **likelihood ratio test**. The likelihood ratio is

$$LR = 2(\ln L_U - \ln L_R) \sim \chi^2(J) \tag{16.14}$$

if the null is true. The parameter $J$ is the degrees of freedom for the $\chi^2$ and it equals the number of hypotheses you are testing jointly, 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 unrestricted and restricted models, respectively. The procedure is to estimate restricted and unrestricted models, collect the log-likelihood from each, compose the LR statistic, and compute its $p$-value.

**Parameter significance** For the first hypothesis, the restriction implies that $\beta_3 = 0$ under the null. The restricted model is:

$$P_{coke} = \Phi(\beta_1 + \beta_2 pratio + \beta_4 disp\_pepsi) \tag{16.15}$$

The script to estimate restricted and unrestricted models is:

```
1  open "@workdir\data\coke.gdt"
2  list x = const pratio disp_pepsi disp_coke
3
4  probit coke x --quiet
5  scalar llu = $lnl
6
7  probit coke const pratio disp_pepsi --quiet
8  scalar llr = $lnl
9
10 scalar lr = 2*(llu-llr)
11 printf "Ho: b3 = 0    Ha: b3 != 0\n \
12   LR   = %.4f\n \
13   p-value = %.4f\n", \
14   lr, pvalue(C,1,lr)
```

Since there are two models to estimate, the script looks more complicated than it is. The unrestricted model, which contains all variables, is estimated in line 4. The value of the log-likelihood is saved into a scalar `llu` using the accessor `$lnl`. In line 7 the restricted model is estimated and its log-likelihood is saved into a scalar `llr` in line 8, again using the accessor `$lnl`. The likelihood ratio is computed in line 10 and then `printf` is used to summarize things for us.

The result is:

```
Ho: b3 = 0    Ha: b3 != 0
    LR   = 5.0634
    p-value = 0.0244
```

This is nearly the same result obtained using the Wald test. For nonlinear estimators, these statistics will normally yield (slightly) different results.

**Economic hypothesis** Again, the hypothesis that the two types of display have equal and opposite effects on the probability of purchasing Coke (i.e., $\beta_3 = -\beta_4$). To estimate the restricted model, substitute the restriction into the model and collect parameters. This forms the new variable ($disp\_pepsi - disp\_coke$), which is used in estimating the restricted likelihood. The script to compute and evaluate the $LR$ is:

```
series c_p = disp_pepsi-disp_coke
probit coke x --quiet
scalar llu = $lnl
probit coke const pratio c_p --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b3+b4 = 0    Ha: b3+b4 != 0\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,1,lr)
```

The result is

```
Ho: b3+b4 = 0    Ha: b3+b4 != 0
    LR  = 5.4218
    p-value = 0.0199
```

The statistic is 5.42, which is very close to the value from the Wald test of this hypothesis.

The next hypothesis to consider is that the displays have no effect. The null and alternative hypotheses are:

$$H_0\colon \beta_3 = 0 \text{ and } \beta_4 = 0 \quad H_1\colon \beta_3 \neq 0 \text{ or } \beta_4 \neq 0 \tag{16.16}$$

The **gretl** code is

```
probit coke x --quiet
scalar llu = $lnl
probit coke const pratio --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b3 = b4 = 0  vs.  Ha: b3 != 0, b4 != 0\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,2,lr)
```

This statistic will have an $\chi^2(2)$ distribution if the null hypothesis is true. The outcome in **gretl** is:

```
Ho: b3 = b4 = 0  vs.  Ha: b3 != 0, b4 != 0
    LR  = 19.5515
    p-value = 0.0001
```

Again, this hypothesis is rejected at any reasonable level of significance.

**Overall regression significance** For this hypothesis, the null is for all parameters other than the constant to be jointly zero, i.e., $\beta_2 = \beta_3 = \beta_4 = 0$.

```
probit coke x --quiet
scalar llu = $lnl
probit coke const --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b2=b3=b4=0  vs.  Ha: not Ho\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,3,lr)
```

This statistic will have an $\chi^2(2)$ distribution if the null hypothesis is true. The outcome in **gretl** is:

```
Ho: b2=b3=b4=0  vs.  Ha: not Ho
    LR  = 145.8234
    p-value = 0.0000
```

According to this result, the model is significant at 5%.

## 16.4 Endogenous Regressors

With an endogenous, continuous regressor there are at least two approaches one can take to estimate the parameters of the model consistently. The first is to use linear two-stage least squares. This is the endogenous regressor counterpart to the linear probability model.

The other approach is to use an instrumental variable probit (or logit). This is NOT a two-stage estimator in the same sense as linear 2SLS. It requires some care in practice. For some computational hints on computing the AGLS estimator see Adkins (2009).

### Example 16.9 in *POE5*

In this example, the *mroz.gdt* data are used to estimate a model of female labor force participation (*LFP*). *LFP* is binary, taking the value 1 if a female is in the labor force and 0 otherwise. The linear probability model estimated is:

$$LFP = \beta_1 + \alpha_1 educ + \beta_2 exper + \beta_3 exper^2 + \beta_4 kidls6 + \beta_5 age + e$$

The woman's years of schooling, *educ*, is considered to be endogenous. For linear 2SLS we need an instrument. This is provided by her mother's education, *mothereduc*. Following the discussion in section 10.2.2 we estimate:

```
1 open "@workdir\data\mroz.gdt"
2 square exper
3 list x = const educ exper sq_exper kidsl6 age
4 list inst = const exper sq_exper kidsl6 age mothereduc
5 tsls lfp x ; inst --robust
```

Experience is squared in line 2, a list of regressors created in line 3 and the entire set of instruments saved to a list in line 4. The `tsls` estimator is used with the `--robust` option, which in this instance is robust with respect to the known heteroskedasticity of the binary dependent variable. The output is:

```
              LPM_IV: TSLS, using observations 1–753
                     Dependent variable: lfp
                       Instrumented: educ
       Instruments: const exper sq_exper kidsl6 age mothereduc
        Heteroskedasticity-robust standard errors, variant HC1

                Coefficient    Std. Error    t-ratio   p-value

   const          0.5919       0.2382          2.485   0.0132
   educ           0.03878      0.01649         2.352   0.0189
   exper          0.03938      0.005977        6.589   0.0000
   sq_exper      −0.0005715    0.0001944      −2.940   0.0034
   kidsl6        −0.2712       0.03212        −8.442   0.0000
   age           −0.01769      0.002279       −7.761   0.0000

Mean dependent var     0.568393   S.D. dependent var    0.495630
Sum squared resid      137.2405   S.E. of regression    0.428628
R^2                    0.257174   Adjusted R^2          0.252202
F(5, 747)              74.78407   P-value(F)            1.56e–63
Log-likelihood        −5568.220   Akaike criterion      11148.44
Schwarz criterion      11176.19   Hannan–Quinn          11159.13
```

Hausman test –
  Null hypothesis: OLS estimates are consistent
  Asymptotic test statistic: $\chi^2(1) = 0.211625$
  with p-value = 0.645497

Weak instrument test –
  First-stage $F(1, 747) = 144.4$

While the instrument appears to be strong ($F$=144.4), the Hausman test for the exogeneity of education is not rejected at 5%.

The first stage regression

```
ols educ inst
```

yields:

FirstStage: OLS, using observations 1–753
Dependent variable: educ

| | Coefficient | Std. Error | $t$-ratio | p-value |
|---|---|---|---|---|
| const | 8.995 | 0.5848 | 15.38 | 0.0000 |
| exper | 0.09396 | 0.02658 | 3.535 | 0.0004 |
| sq_exper | −0.002066 | 0.0008746 | −2.363 | 0.0184 |
| kidsl6 | 0.3540 | 0.1576 | 2.246 | 0.0250 |
| age | −0.002618 | 0.01108 | −0.2363 | 0.8133 |
| mothereduc | 0.2905 | 0.02261 | 12.85 | 0.0000 |

| | | | |
|---|---|---|---|
| Mean dependent var | 12.28685 | S.D. dependent var | 2.280246 |
| Sum squared resid | 3075.952 | S.E. of regression | 2.029222 |
| $R^2$ | 0.213320 | Adjusted $R^2$ | 0.208054 |
| $F(5, 747)$ | 40.51193 | P-value($F$) | 6.57e–37 |
| Log-likelihood | −1598.311 | Akaike criterion | 3208.622 |
| Schwarz criterion | 3236.366 | Hannan–Quinn | 3219.310 |

Notice that the squared value of the $t$-ratio on *mothereduc* is equal to the first-stage $F$ statistic for weak instruments.

The other possibility is to estimate an instrumental variables probit version of the model. This can be done using a package called *HIP*. *HIP* is written by Riccardo Lucchetti and Claudia Pigini and features a collection of scripts to estimate heteroskedastic probit models, that may include endogenous regressors. Estimation is by maximum likelihood assuming that the latent errors are assumed to be normally distributed, and hence estimated as probit. Below, we reestimate the model in Example 16.9 using *HIP*.

First, head to the **gretl** function package server and download and install *HIP.gfn*. The syntax used by *HIP* is a little different from that of `tsls`. In `tsls` two lists are composed: a list of regressors in the model and a full list of the exogenous variables, including the external instruments. *HIP* does this differently.

HIP can use four arguments.

1. the dependent variable (series)–`y`
2. the exogenous explanatory variables (normally as a list)–`Exog_x`
3. the endogenous explanatory variables (a list or, as in this case, a single variable name)–`Endog_x`
4. the external instruments (a list or, as in this case, a single variable name)–`External_IV`

So, the regressors are separated into exogenous and endogenous. The instrument list includes only the external instrument(s).

The syntax is:

```
HIP(y, Exog_x, Endog_x, External_IV)
```

HIP takes other arguments if a model of heteroskedasticity is used and to control the amount of output produced. Also, HIP is available from the GUI, which will be discussed presently.

For our example:

```
include HIP.gfn
list exog_vars = const exper sq_exper kidsl6 age
b=HIP(lfp, exog_vars, educ, mothereduc)
```

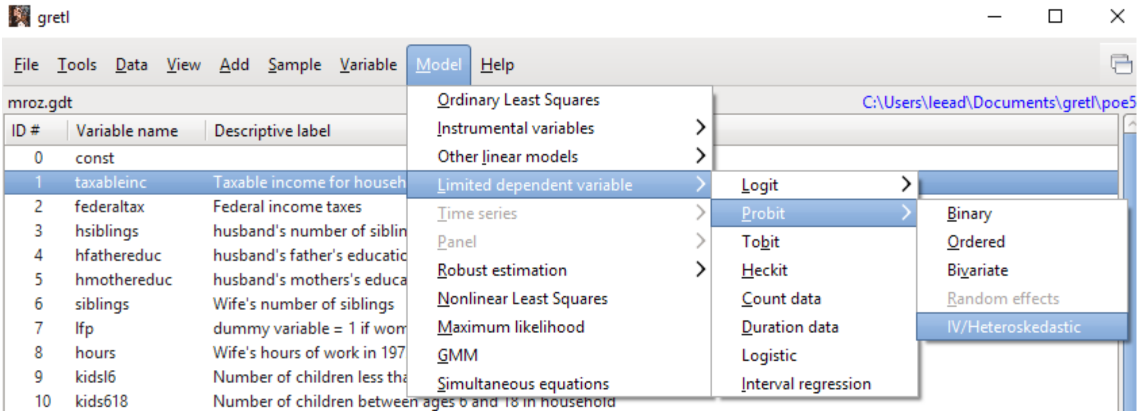
The exogenous regressors are placed into the list `exog_vars`. There is only 1 endogenous regressor, `educ`, and it is the next argument. It enters as a series. Finally, the external instrument is `mothereduc` and it also enters as a series. The output is printed to the screen and saved as a bundle, **b**, in the session window. The output is:

```
Probit model with endogenous regressors
ML, using observations 1-753
```

```
Dependent Variable: lfp
Instrumented: educ
Instruments: const, exper, sq_exper, kidsl6, age, mothereduc
Parameter covariance matrix: OPG

              coefficient   std. error       z       p-value
  ----------------------------------------------------------
  const        0.316430     0.767733       0.4122   0.6802
  exper        0.122673     0.0195898      6.262    3.80e-010 ***
  sq_exper    -0.00178989   0.000619681   -2.888    0.0039    ***
  kidsl6      -0.877123     0.119611      -7.333    2.25e-013 ***
  age         -0.0576838    0.00822293    -7.015    2.30e-012 ***
  educ         0.127417     0.0530207      2.403    0.0163    **

Log-likelihood       -2002.9255  Akaike criterion    4033.8511
Schwarz criterion     4098.5880  Hannan-Quinn        4058.7909
Conditional ll      -404.614712  Cragg-Donald stat.   166.205

Overall test (Wald) = 160.054 (5 df, p-value = 0.0000)
Endogeneity test (Wald) = 0.154795 (1 df, p-value = 0.6940)
```

The test results are quite similar to those of the linear probability IV estimator. Education is not found to be endogenous at 5%. The $t$-ratio on education was 2.35 in the LPM version and is 2.4 in the IV/probit version. Of course, computing marginal effects in the IV/probit is complicated by nonlinearity.

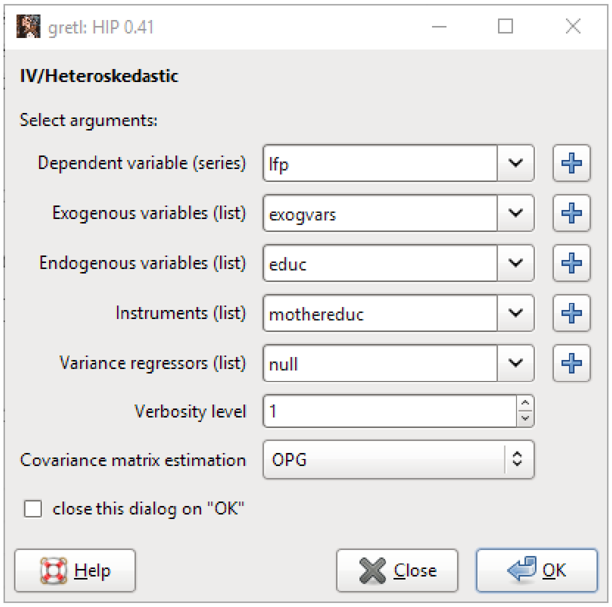
The GUI is easy to use. Once installed, use **Model** > **Limited dependent variable** > **Probit** > **IV/Heteroskedastic** to launch the dialog box shown in Figure 16.4 below.



| ID # | Variable name | Descriptive label |
|---|---|---|
| 0 | const | |
| 1 | taxableinc | Taxable income for househ |
| 2 | federaltax | Federal income taxes |
| 3 | hsiblings | husband's number of siblin |
| 4 | hfathereduc | husband's father's educatic |
| 5 | hmothereduc | husband's mothers's educa |
| 6 | siblings | Wife's number of siblings |
| 7 | lfp | dummy variable = 1 if wor |
| 8 | hours | Wife's hours of work in 197 |
| 9 | kidsl6 | Number of children less tha |
| 10 | kids618 | Number of children between ages 6 and 18 in household |

Figure 16.3: Choose **Model** > **Limited dependent variable** > **Probit** > **IV/Heteroskedastic** from the pull-down menu in **gretl**'s main window.

Click OK and estimates will be returned in a window. From there, the bundle can be saved, printed, or the output copied into memory for pasting into your editor (text) or word processor (RTF).



IV/Heteroskedastic

Select arguments:

Dependent variable (series) lfp

Exogenous variables (list) exogvars

Endogenous variables (list) educ

Instruments (list) mothereduc

Variance regressors (list) null

Verbosity level 1

Covariance matrix estimation OPG

□ close this dialog on "OK"

Figure 16.4: HIP dialog box. Fill in the dependent variable, and create lists for exogenous regressors, endogenous regressors, and external instruments. Single series can also be used as inputs as done here by typing in the variable's name.

## 16.5 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 16.5.1. The other methods demonstrate how to use **gretl**'s scripting language in conjunction with other software, in this case **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 "@workdir\data\nels_small.gdt"
list x = const grades
mnl <- logit psechoice x --multinomial
```

The `--multinomial` option is used when the choices are unordered. For **ordered logit**, it is omitted. Gretl analyzes the dependent variable, in this case `psechoice`, to determine that it is actually discrete. *psechoice* can takes three possible values (1, 2, or 3) and the `logit` function in **gretl** should handle this automatically.

MNL estimation yields the output shown below:

mnl: Multinomial Logit, using observations 1–1000
Dependent variable: psechoice
Standard errors based on Hessian

| | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | 2.50642 | 0.418385 | 5.991 | 0.0000 |
| grades | −0.308789 | 0.0522849 | −5.906 | 0.0000 |
| const | 5.76988 | 0.404323 | 14.27 | 0.0000 |
| grades | −0.706197 | 0.0529246 | −13.34 | 0.0000 |

| | | | |
|---|---|---|---|
| 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 percent)
Likelihood ratio test: $\chi^2(2) = 286.689$ [0.0000]

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.

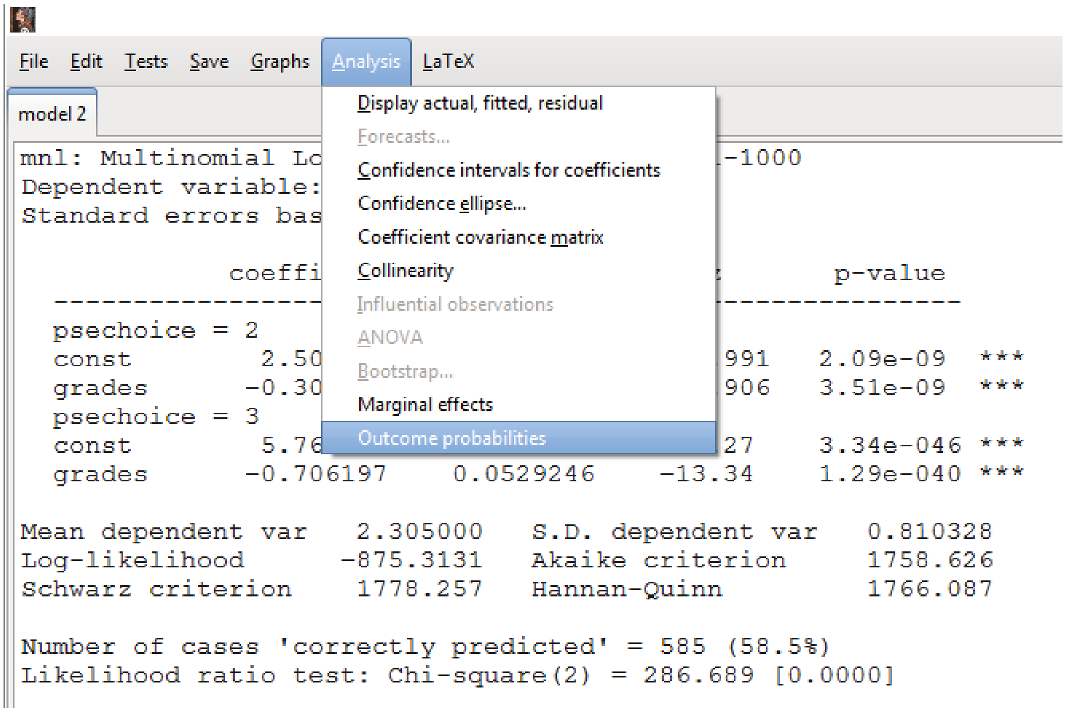
The probability of choosing an alternative in multinomial logit is

$$p_{i1} = \frac{1}{1+\sum_{j=2}^{J}\exp(\beta_{1j}+\beta_{2j}x_{i2}+\cdots+\beta_{kj}x_{ik})} \qquad j=1 \tag{16.17}$$

$$p_{ij} = \frac{\exp(\beta_{1j}+\beta_{2j}x_{i2}+\cdots+\beta_{kj}x_{ik})}{1+\sum_{j=2}^{J}\exp(\beta_{1j}+\beta_{2j}x_{i2}+\cdots+\beta_{kj}x_{ik})} \qquad j\neq 1 \tag{16.18}$$

Obtaining the probabilities is simple. Estimate the model via the GUI (**Model**>**Limited dependent variable** >**Logit**>**Multinomial**) or, as done above, by assigning the output to a model that appears in the session window. From the model window select **Analysis**>**Outcome probabilities** to produce the predicted probabilities for each case in the sample. This is shown in Figure 16.5. The first few probabilities are:

```
  Estimated outcome probabilities for psechoice
```



```
mnl: Multinomial Lo                          -1000
Dependent variable:
Standard errors bas

             coeffi                                   p-value
  ------------------                   ----------------
  psechoice = 2
  const        2.50                    991    2.09e-09  ***
  grades      -0.30                    906    3.51e-09  ***
  psechoice = 3
  const        5.76                    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 16.5: You can obtain the outcome probabilities from the multinomial logit model window. These are also available after estimation in the `$mnlprobs` accessor.

```
              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:

```
function list mlogitprob(series y "Dependent variable",
      list x "List of regressors",
      matrix theta "Coefficient vector")
    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)))
```

```
11
12         loop for i=1..j --quiet
13             if i == 1
14                 series p$i = 1/den
15             else
16                 scalar q = i - 1
17                 series num = exp(X[q,]*b[,q])
18                 series p$i=num/den
19             endif
20             list probs += p$i
21         endloop
22         return probs
23     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 5 the independent variables are converted into a matrix called X. Line 6 obtains the maximum category in the coding of the dependent variable. The variable *psechoice* takes values 1, 2, and 3 in the data so this will return the value 3. If the data are coded 0, 1, 2, as they sometimes are, the script must be altered 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 $(j)^{th}$ 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 $e^{index_j}$. The computed probabilities are added to the list probs using the operator (+=), which is an efficient way of appending new results to existing ones. 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.

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 "@workdir\data\nels_small.gdt"
list x = const grades
mnl <- logit psechoice x --multinomial
matrix theta = $coeff
list n = mlogitprob(psechoice, x, theta)
smpl 1 12
print n --byobs
smpl full
```

This can be made easy simply by using 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 this function marginal effects can be computed.

To get average marginal effects is a snap at this point. Add 1 to the value of *grades*, recompute the probabilities, and average the difference between the two. This requires renaming the predicted probabilities, but that is easily done using the `rename` function.

```
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 similar to the one used above, `mlogitprob`. 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 scalar or matrix. The new function is:

```
function matrix mlogitprob_at(series y "Dependent variable",
      matrix x "Representative point 1xk",
      matrix theta "Coefficient vector")
    # computes probabilities of each choice at a representative point
    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 easy to use and reproduces the results in *POE5* Table 16.3 using the script below.

```
open "@workdir\data\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 = %.3g are\n %8.4f\n" ,quantile(grades,.05), p05
printf "\nThe predicted probabilities for student\
grades = %.3g are\n %8.4f\n",quantile(grades,.50), p50
```

Output from this routine shows:[7]

```
The predicted probabilities for student grades = 2.63 are
   0.0177  0.0964  0.8859

The predicted probabilities for student grades = 6.64 are
   0.1810  0.2856  0.5334
```

To use the function to get marginal effects of 1 unit change in grades for median and $95^{th}$ percentile students create quantiles based on the series grades and use these in the function. Taking the difference in probabilities yields an approximate (discrete) marginal effect at the given quantiles.

[7] Notice that **gretl** computes the quantile using a weighted average of the 50th and 51st observations, i.e., .95*2.63+.05*2.64=2.3605 so the result differs slightly from the ones in *POE5*.

```
open "@workdir\data\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
rnameset(me,"MER")
cnameset(me,"NoColl 2Year 4Year ")
printf "\nThe marginal effect of grades for student\
grades=%5.2f\n\
%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
cnameset(me,"NoColl 2Year 4Year ")
rnameset(me,"MER")
printf "\nThe marginal effect of grades for student\
grades=%5.2f\n\
%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 one and the marginal effects should sum to zero. This script also uses a common trick. The one unit change is evaluated at ±0.5 of each quantile; then the discrete difference is taken. The results match those in *POE5* reasonably well.

```
The marginal effect of grades for student grades = 6.64
       NoColl   2Year   4Year
MER   0.0118  0.0335 -0.0452

The marginal effect of grades for student grades = 2.63
       NoColl   2Year   4Year
MER   0.0118  0.0335 -0.0452
```

We can also employ the *lp-mfx* package to compute the marginal effects and their standard errors. This uses the mlogit_dpj_dx function supplied in this function package. The syntax for is:

```
function matrix mlogit_dpj_dx (matrix b "parameter estimates",
    list XL "list of regressors",
    matrix x "vector of x-values",
    int j "1-based index of outcome",
    int m "number of possible outcomes")
```

The inputs are fairly clear. The 4th input, `j`, is the index number of the choice, so for instance for the two=year college choice, which is the second, this integer would be set to 2. The last integer, `m`, is total number of possible choices, in our case 3. So, the following will return the marginal impact on the probability of finishing two-year college with grade = 6.64.

```
matrix x1 = { 1 , 6.64 }
matrix c1 = mlogit_dpj_dx($coeff, $xlist, x1, 2, 3)
```

To obtain the marginal effects and to print results, we use a function call `mnl_se_lpfmx`. This function appears below:

```
function matrix mnl_se_lpfmx (matrix b "parameter estimates",
      matrix covmat "Covariance of MNL",
      list XL "list of regressors",
      matrix x "vector of x-values",
      int j "1-based index of outcome",
      int m "number of possible outcomes",
      int df "degrees of freedom for CI" )

    matrix p = mlogit_dpj_dx(b, XL, x, j, m)
    matrix jac = fdjac(b, mlogit_dpj_dx(b, XL, x, j, m))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

    cnameset(results, "Lower ME Upper StdErr")
    printf "95%% CI for MER\n%10.4f\n", results
    return results
end function
```

It takes seven inputs.

1. A vector of estimates from MNL
2. The estimated variance-covariance matrix
3. A list of regressors

4. A vector representing the point at which the derivative will be evaluated
5. The index number of the choice, j.
6. The total number of choices, m
7. An integer for degrees-of-freedom for the t-critical value used in the confidence interval

To replicate all of the results in Table 16.3 of *POE5* requires this to be executed a number of times using different values of `x`, `j`, and `m`. To demonstrate how this is used, the MER for the second of 3 possible choices with grades=6.64 use:

```
include lp-mfx.gfn
open "@workdir\data\nels_small.gdt"
list x = const grades
matrix x_at = {1, 6.64 }
logit psechoice x --multinomial
c = mnl_se_lpfmx( $coeff, $vcv, x, x_at, 2, 3, $df)
```

to produce:

```
95% CI for MER
     Lower         ME      Upper     StdErr
    0.0296     0.0446     0.0596     0.0076
```

This matches line 3 of Table 16.3 in *POE5* .

### 16.5.1 Using the `mle` Command for MNL

In this section the maximum likelihood estimator of the MNL model is estimated using **gretl**'s generic `mle` command.

Although versions of Gretl beyond 1.8 include a command for estimating MNL, it can be estimated with a little effort using the `mle` block commands (see Chapter 14 for other examples). To use the `mle` function, the user writes a script 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 `scalar` commands). If desired, the user may 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, then analytical ones may be required in order to make the program work.

What appears below is taken from the Gretl Users Guide. The example for MNL for *POE5* requires only a slight modification in order for the program to run with the *nels_small* dataset.

The multinomial logit function, which was found in the Gretl User's Guide (Cottrell and Lucchetti, 2011), is defined

```
function series mlogitlogprobs(series y "Dependent Variable",
      matrix X "Independent variables",
      matrix theta "Parameters")
    # This function computes the log probabilities for MLE
    # estimation of MNL
    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
    endloop
    return 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, the data must be of the right form in the right order for the function to work properly. After loading the data, determine if the dependent choice variable is in the correct format for the function. The function requires the choices to start at 0. Ours is not. So, the first step is to convert 1, 2, 3 into choices 0, 1, 2 by subtracting 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 "@workdir\data\nels_small.gdt"
series psechoice = psechoice -1
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 results from the program appear below. They match those in *POE5* produced by the `logit` command with the `--multinomial` option and are dirt simple to obtain.

```
                 ml: ML, using observations 1–1000
         loglik = mlogitlogprobs(psechoice, X, theta)
                  Standard errors based on Hessian

                  Estimate    Std. Error       z       p-value

      theta[1]     2.50643    0.418385       5.991    0.0000
      theta[2]   −0.308790    0.0522849     −5.906    0.0000
      theta[3]     5.76988    0.404323      14.27     0.0000
      theta[4]   −0.706197    0.0529246    −13.34     0.0000

  Log-likelihood      −875.3131   Akaike criterion   1758.626
  Schwarz criterion    1778.257   Hannan–Quinn       1766.087
```

Notice that contained in this output is a reference to the function used to specify the log-likelihood. Gretl stores estimates which can be accessed using `$coeff`. Other available accessors can be listed using the `varlist --type=accessor` command.

```
? varlist --type=accessor

model-related
 $T (scalar: 1000)
 $df (scalar: 996)
 $ncoeff (scalar: 4)
 $lnl (scalar: -875.313)
 $aic (scalar: 1758.63)
 $bic (scalar: 1778.26)
 $hqc (scalar: 1766.09)
 $sample (series)
 $coeff (matrix)
 $stderr (matrix)
 $vcv (matrix)
 $rho (matrix)
```

```
  other
   $nobs (scalar: 1000)
   $nvars (scalar: 9)
   $pd (scalar: 1)
   $t1 (scalar: 1)
   $t2 (scalar: 1000)
   $tmax (scalar: 1000)
   $datatype (scalar: 1)
   $windows (scalar: 1)
   $version (scalar: 20180)
   $error (scalar: 0)
   $seed (scalar: 1.53238e+009)
   $huge (scalar: 1e+100)
   $stopwatch (scalar: 6303.82)
```

This permits a user written likelihood to be used with the other functions in this chapter to produce the probabilities and marginal effects. This was the approach taken in the preceding section and the details will not be repeated here.

## 16.6 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 prices the alternatives. This kind of data differs from the data used in multinomial logit example because there 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 his choice of schooling. 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

$$p_{ij} = \frac{\exp(\beta_{1j} + \beta_2 price_{ij})}{\exp(\beta_{11} + \beta_2 price_{i1}) + \exp(\beta_{12} + \beta_2 price_{i2}) + \exp(\beta_{13} + \beta_2 price_{i3})} \tag{16.19}$$

Now there is only 1 parameter that relates to *price*, but there are J=3 constants. One of these cannot be 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 better suited for general use, but produces results that are quite similar to those 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 Probabilities

```
1  function scalar clprobs(list y "list of choices",
2        list x "list of independent variables",
3        matrix theta "parameters")
4      # computes the probabilities for Conditional Logit
5      # Used in user written MLE
6      matrix Y = { y }
7      matrix p = { x }
8      scalar n = $nobs
9      matrix P = {}
10     loop i=1..n --quiet
11         scalar i1 = exp(theta[1]+theta[3]*p[i,1])
12         scalar i2 = exp(theta[2]+theta[3]*p[i,2])
13         scalar i3 = exp(theta[3]*p[i,3])
14         scalar  d = i1+i2+i3
15         matrix pp = (Y[i,1]==1)*i1/d +\
16                     (Y[i,2]==1)*i2/d +\
17                     (Y[i,3]==1)* i3/d
18         matrix P = P | pp
19     endloop
20     return sumc(ln(P))
21 end function
```

Lines 6 and 7 convert the lists to matrices. The number of observations is counted in line 8 and an empty matrix is created to hold the result in 9. The loop that starts in line 10 just computes the probabilities for each observed choice. The scalars `i1`, `i2` and `i3` are added together for the denominator of equation (16.19); 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.

In order to use this function to work as intended the data must be arranged properly. Below is a script that rearranges the data contained in *cola.gdt* and outputs the new data into a file called *cola_mixed.gdt*.

```
open "@workdir\data\cola.gdt"
matrix ids = values(id)
matrix idx_pepsi  = seq(1, 5466, 3)
matrix idx_7up    = seq(2, 5466, 3)
matrix idx_coke   = seq(3, 5466, 3)
matrix Price      = { price }
matrix Choice     = { choice }
matrix cokePrice    = Price[idx_coke]
matrix pepsiPrice   = Price[idx_pepsi]
matrix sevenupPrice = Price[idx_7up]
matrix cokeChoice     = Choice[idx_coke]
matrix pepsiChoice    = Choice[idx_pepsi]
matrix sevenupChoice  = Choice[idx_7up]

nulldata 1822 --preserve
series coke = cokePrice
series pepsi = pepsiPrice
series sevenup = sevenupPrice
series d_coke = cokeChoice
series d_pepsi = pepsiChoice
series d_sevenup = sevenupChoice

setinfo d_pepsi -d "1 if Pepsi, 0 otherwise"
setinfo d_sevenup -d "1 if 7-Up, 0 otherwise"
setinfo d_coke -d "1 if Coke, 0 otherwise"
setinfo pepsi -d "Pepsi price"
setinfo sevenup -d "7-Up price"
setinfo coke -d "Coke price"
store cola_mixed
```

Warning, this is not pretty, but it works. The data from *cola.gdt* are arranged by a person's *id*. Each person has three choices and each choice is an observation. The choice is an indicator and arranged in order (Pepsi, 7UP, and Coke) for each id. The script pulls out each choice (every third observation) and puts it into a column vector, which is then converted to series and saved. The choices are recoded as 1=pepsi, 2=sevenup, and 3=coke. Prices are handled similarly. Each beverage has its own price and choice series. The original data are arranged:

```
obs 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
```

The clprobs function is written such that the data should be arranged as:

```
id  d_coke   d_pepsi d_sevenup  coke    pepsi   sevenup
 1    1        0          0     1.79    1.79     1.79
 2    1        0          0     1.79    1.79     0.89
 3    1        0          0     1.41    0.84     0.89
 4    1        0          0     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, any variables not being used are dropped.

Without further explanation, run the script which saves a new dataset called *cola_mixed.gdt* to the working directory. This is the file that will be loaded and used with the following example. The *cola_mixed.gdt* will also be included in the datasets distributed with this manual as well.

To estimate the model, load the data that you have created. Create a set of indicators based on the variable choice. Starting values for the three parameters in the MLE are given in line 4. Lines 6-8 contain the mle block. The log likelihood function uses a user written function clprobs and identifies the parameter vector as theta.

```
1 open "@workdir\cola_mixed.gdt"
2 list y = d_pepsi d_sevenup d_coke
3 list x =  pepsi sevenup coke
4 matrix theta = {-2, .3, .1}
5
6 mle lln = clprobs(y, x, theta)
7     params theta
8 end mle
```

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 *POE5*. 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.

More work is required in order to obtain marginal effects and their standard errors. This is tackled below.

### Example 16.13 in *POE5*

In this example there are three choices available to the consumer: Coke, Pepsi, and 7-Up. The only variable influencing the choice are the beverages' prices. The model is estimated using mle as before and the parameters and covariance are saved using accessors. The probabilities of choosing each drink is evaluated for a set of prices: $1 for Pepsi, $1.25 for 7-Up, and $1.10 for Coke. To do this, we use a function called clprobs_at, which is shown below:

```
function matrix clprobs_at(matrix x, matrix theta)
    scalar i1 = exp(theta[1]+theta[3]*x[1])
    scalar i2 = exp(theta[2]+theta[3]*x[2])
    scalar i3 = exp(theta[3]*x[3])
    scalar  d = i1+i2+i3
    matrix pp = i1/d ~ i2/d ~ i3/d
    return pp
end function
```

The function requires two inputs: a vector of variables at which to evaluate probabilities and the vector of coefficients from CL. Its form is relies heavily on the previously considered function, clprobs.

The clprobs_at function is used to calculate the probabilities and column names are added to its matrix output.

```
mle lln = clprobs(y, x, theta)
    params theta
end mle

```

```
matrix theta = $coeff
matrix covmat = $vcv
matrix x1 = {1.0, 1.25, 1.10}
matrix mm = clprobs_at(x1,theta)
cnameset(mm, " Pepsi 7-Up Coke")
print mm
```

The probabilities rendered by the `clprobs_at` function are:

```
     Pepsi         7-Up         Coke
   0.48319      0.22746      0.28934
```

At those prices, the probability of purchasing a Pepsi is 0.483. As we shall see, the `clprobs_at` function will play an important role in this example.

**Marginal Effects** Next, the own price marginal effect is computed using the output `clprobs_at`. As shown in *POE5*,

$$\frac{\partial p_{ij}}{\partial price_{ij}} = p_{ij}(1 - p_{ij})\beta_2$$

where $p_{ij}$ is the probability that individual $i$ purchases drink $j$.

For the cross-price marginal effect

$$\frac{\partial p_{ij}}{\partial price_{ik}} = -p_{ij}p_{ik}\beta_2.$$

This should have the opposite sign as the own price effect. To obtain these for the given point

```
scalar me_op_pepsi = mm[1]*(1-mm[1])*theta[3]  # own price pepsi
scalar me_cp_7up  = -mm[2]*mm[1]*theta[3]      # cross price 7up
scalar me_cp_coke = -mm[1]*mm[3]*theta[3]      # cross price coke
printf "\n Own-Price (Pepsi) marginal effect (1$) = %.3f\n\
Cross-Price (7-up) effect ($1) = %.3f\n\
Cross-Price (Coke) effect ($1)= %.3f\n",\
       me_op_pepsi, me_cp_7up, me_cp_coke
```

For $x_i = \{1.0, 1.25, 1.10\}$ we get

```
 Own-Price (Pepsi) marginal effect (1$) = -0.573
 Cross-Price (7-up) effect ($1) = 0.252
 Cross-Price (Coke) effect ($1)= 0.321
```

The marginal impact on probability appears to be huge since a price increase of 1 unit doubles the price of a Pepsi. A 10 cent price increase would diminish the probability by about 5.7%. The cross effects do in fact have opposite signs, indicating that reducing the probability of purchasing Pepsi will increase the probability of buying one of the others. The probabilities must add to one and the changes must add to zero.

To facilitate the computation of standard errors for the marginal effects, the computations above are put into another function that relies on `clprobs_at` that returns either the own-price marginal effect or a cross-effect that are controlled by the function's inputs.

```
function scalar clprobs_me(matrix *x "vector for the desired point",
      matrix *theta "parameters",
      int q "variable index for own price",
      int p "variable index for other price")
    matrix mm = clprobs_at(x, theta)
    if p == q
        scalar me = mm[q]*(1-mm[q])*theta[3]  # own price pepsi
    else
        scalar me = -mm[p]*mm[q]*theta[3]     # cross price 7up
    endif
    return me
end function
```

For instance, suppose you want to compute the marginal effect of a 1 unit change in Coke price on the probability of purchasing a Pepsi. The initial prices are $1 for Pepsi, $1.25 for 7-Up, and $1.10 for Coke. The `clprobs_me` function would be:

```
matrix x2 = {1.0, 1.25, 1.10}
scalar q = 1                         # Own price: 1 = pepsi
scalar p = 3                         # Other price: 2 = 7up, 3 = coke
scalar c = clprobs_me(&x2, &theta, q, p)
```

Own price parameter, q, is set to 1 for Pepsi, the other price parameter, p, is set to 3 for Coke. If $p = q$, then the own price effect is computed, otherwise it computes a cross-effect.

With this function, we can add the standard three lines to compute the delta standard errors and then use the output to print a confidence interval for the marginal effect.

```
matrix jac = fdjac(theta, clprobs_me(&x2, &theta, q, p))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)

t_interval(c,se,$df,.95)
```

This produces:

```
The 95% confidence interval centered at -0.573 is (-0.6421, -0.5048)
```

**Probabilities at specific points** The probability of making a choice at specific prices is easy to automate. We add the ability to compute delta standard errors in this section. Once again, the function `clprobs_at` is used and the mechanism to compute the delta standard errors is added to a script.

Suppose the prices are $1 for Pepsi, $1.25 for 7-Up, and $1.10 for Coke. We evaluate the probabilities and compute standard errors using:

```
matrix x2 = {1.0, 1.25, 1.10}
matrix m2 = clprobs_at(x2, theta)
matrix jac = fdjac(theta, clprobs_at(x2, theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = m2' ~ se
cnameset(results, " Probability std_error t-ratio" )
rnameset(results, "Pepsi 7UP Coke" )
print results
```

The result is shown below:

```
       Probability    std_error
Pepsi      0.48319     0.015356
  7UP      0.22746     0.011710
 Coke      0.28934     0.011491
```

As seen at the beginning of this example, the probability of purchasing a Pepsi at these prices is 0.483. This time, a standard error of 0.0153 is computed that could be used to produce a confidence interval for the ME.

Increasing the price of Pepsi by .10 and reestimating the probabilities produces:

```
       Probability    std_error
Pepsi      0.42632     0.013542
  7UP      0.25250     0.011546
 Coke      0.32119     0.012168
```

The probability of purchasing a Pepsi is falling to 0.426, while the probability of purchasing the others is increasing. In the next section the changes in probability are examined and standard errors for those effects are computed.

**Discrete changes in variables** Using `clprobs_at` one can also examine the effects of discrete changes on the probabilities. This is the idea behind the following function that computes probabilities at two different data points and then computes the change in probability induced by that.

```
function matrix clprobs_me_d(matrix ⋆x1,
      matrix ⋆x2,
      matrix ⋆theta)
    matrix mm = clprobs_at(x1, theta)
    matrix m2 = clprobs_at(x2, theta)
    mat = m2-mm
    return mat
end function
```

The advantage of this is that an entire matrix of effects (own-price and cross-price) are returned at once.

To compute marginal effects and standard errors for a discrete change in x where the price of Coke is increased from $1.10 to $1.25 use the script:

```
matrix x2 = {1.0, 1.25, 1.25}
matrix x1 = {1.0, 1.25, 1.10}
matrix c2 = clprobs_me_d(&x1, &x2, &theta)
matrix jac = fdjac(theta, clprobs_me_d(&x1, &x2, &theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = c2’ ˜ se ˜ c2’./se
cnameset(results, " m_effect std_error t-ratio" )
rnameset(results, "Pepsi 7UP Coke" )
print results
```

This produces:

```
           m_effect    std_error      t-ratio
Pepsi      0.044490    0.0033481       13.288
  7UP      0.020944    0.0011616       18.030
 Coke     -0.065434    0.0039051      -16.756
```

Again, the own-price effect is negative and the cross-price ones are positive.

This exercise is repeated for a change in the price of Pepsi from \$1 to \$1.10. The price of Coke is \$1.10.

```
matrix x2 = {1.1, 1.25, 1.10}
matrix x1 = {1.0, 1.25, 1.10}
matrix c2 = clprobs_me_d(&x1, &x2, &theta)
matrix jac = fdjac(theta, clprobs_me_d(&x1, &x2, &theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = c2’ ~ se ~ c2’./se
cnameset(results, " m_effect std_error t-ratio" )
rnameset(results, "Pepsi 7-UP Coke" )
print results
```

This indicates that the probability of purchasing a Pepsi falls by 0.057 to 0.4263. The estimated standard deviation of that difference is 0.0035.

```
            m_effect    std_error      t-ratio
Pepsi      -0.056875    0.0035459      -16.039
  7UP       0.025033    0.0014376       17.413
 Coke       0.031842    0.0025542       12.467
```

## 16.7 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.

The model is

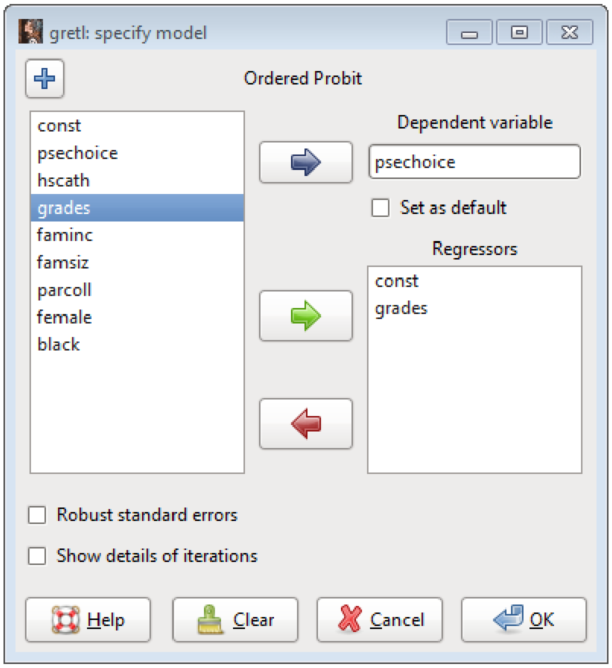
$$y_i^* = \beta grades_i + e_i \tag{16.20}$$

The variable $y_i^*$ is a latent variable, which means its value is unobserved. Instead, we observe categorical choices of college attendance:

$$y_i = \begin{cases} 3 & \text{four-year college;} \\ 2 & \text{two-year college;} \\ 1 & \text{no college.} \end{cases} \tag{16.21}$$

Gretl's `probit` command handles multinomial ordered probit models. Open the *nels_small.gdt* data. The set consists of 1000 observations collected as part of the National Education Longitudinal Study of 1988. The variable grades measures the average grade in math, English and social studies on 13 point scale with 1 being the highest.

The GUI provides access to a dialog box for ordered probit. It is opened from the main **gretl** window using the pull-down menu **Model>Limited dependent variable>Probit>Ordered**. The dialog box is shown in Figure 16.6. Choose a dependent variable and a set of regressors and click OK. This produces the result:



gretl: specify model

Ordered Probit

const
pseechoice
hscath
grades
faminc
famsiz
parcoll
female
black

Dependent variable: pseechoice

□ Set as default

Regressors:
const
grades

□ Robust standard errors

□ Show details of iterations

Figure 16.6: The ordered probit dialog box is opened from the pull-down menu using **Model>Limited dependent variable>Probit>Ordered**.

Model 3: Ordered Probit, using observations 1–1000
Dependent variable: pseechoice
Standard errors based on Hessian

|  | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| grades | −0.306624 | 0.0191735 | −15.99 | 0.0000 |
| cut1 | −2.94559 | 0.146828 | −20.06 | 0.0000 |
| cut2 | −2.08999 | 0.135768 | −15.39 | 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: χ2(1) = 285.672 [0.0000]

Test for normality of residual –
  Null hypothesis: error is normally distributed
  Test statistic: χ2(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 that the probability of being in the lowest category increases and of being in the highest category decreases. Whatever happens in the middle depends on net effects of people being pushed out of category 3 and pulled into category 1.

The other two parameters are estimates of 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 $\phi$ is the probability density function of a standard normal distribution. The parameters $\mu_1$ and $\mu_2$ are the thresholds (or cut-off points) and $\beta$ is the coefficient on `grades`. So, for example 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:

```
open "@workdir\data\nels_small.gdt"
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
```

Once again, the *lp-mfx* package can be used to improve upon this analysis. The function to use is `ordered_dpj_dx`, which works for both ordered probit or logit. The function syntax is:

```
function matrix ordered_dpj_dx (matrix theta "parameter estimates",
    list XL "list of regressors",
    matrix x "vector of regressors",
    int j "1-based index of outcome",
    int m "number of possible outcomes",
    int dist[1:2:1] "distribution" \
    {"logit", "probit"})
```

This should look familiar since it carries the same syntax as the `mlogit_dpj_dx` function from *lp-mfx*. It becomes the basis for a function similar to `mlogit_se_lpfmx`, which is shown below:

```
function matrix op_se_lpfmx (matrix b "parameter estimates",
      matrix covmat "Covariance of MNL",
      list XL "list of regressors",
      matrix x "vector of x-values",
      int j "1-based index of outcome",
      int m "number of possible outcomes",
      int df "degrees of freedom for CI",
      int dist[1:2:1] "distribution" )

    matrix p = ordered_dpj_dx(b, XL, x, j, m, dist)
    matrix jac = fdjac(b, ordered_dpj_dx(b, XL, x, j, m, dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

    cnameset(results, "Lower ME Upper StdErr")
    printf "95%% CI for MER\n%10.4f\n", results
    return results
end function
```

To use this function, make sure that *lp-mfx.gfn* is installed on your computer and has been loaded. Load the data, estimate ordered probit and define a value for grades at which to evaluate the marginal effect (line 4 x=6.64). Choose the remaining parameters. The outcome probability is for the 3rd choice, there are 3 choices total, include the accessor for degrees-of-freedom, and select `dist` using the `$command` accessor.

```
1 include lp-mfx.gfn
2 open "@workdir\data\nels_small.gdt"
3 probit psechoice grades
4 matrix x = {6.64}
5 scalar dist = ($command == "logit")? 1 : 2
6 op_se_lpfmx($coeff, $vcv, $xlist, x, 3, 3, $df, dist)
```

The result produced is:

```
95% CI for MER
     Lower        ME     Upper    StdErr
   -0.1371   -0.1221   -0.1072    0.0076
```

The marginal effect of an increase in grades on the probability of attending 4-year college is $-.1221$ and its 95% confidence interval is $(-0.1371, -0.1072)$.

## 16.8 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.

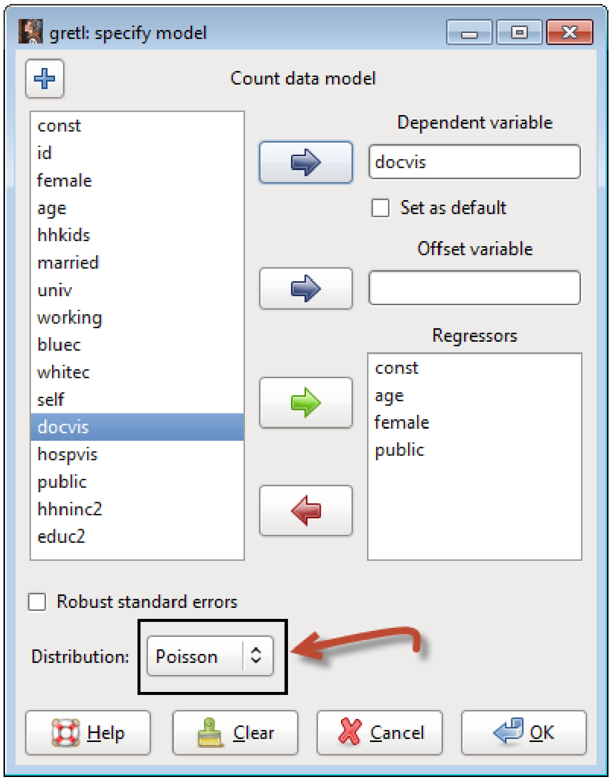
$$P(Y = y|x) = \frac{e^{-\lambda}\lambda^y}{y!} \quad y = 0, 1, 2, \ldots \tag{16.22}$$

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 similar to `ols`, but uses the `possion` command. This is shown in the following script which replicates the example 16.15 in *POE5*.

## Example 16.15 in *POE5*

The number of doctor visits in the past three years is modeled as a function of person's age, sex, and whether he or she has public or private insurance. The data are in *rwm88_small.gdt*, which are a subset from the German Socioeconomic Panel Survey for 1988. Once data are loaded, models for count data can be accessed via the menu system using **Model>Limited dependent variable>Count**. This opens the count data dialog box shown in Figure 16.7 below. Estimation of the model yields:



gretl: specify model

Count data model

const
id
female
age
hhkids
married
univ
working
bluec
whitec
self
docvis
hospvis
public
hhninc2
educ2

Dependent variable: docvis

□ Set as default

Offset variable:

Regressors:
const
age
female
public

□ Robust standard errors

Distribution: Poisson

Figure 16.7: Models for count data can be accessed via the menu system using **Model>Limited dependent variable>Count**. For Example 16.15, choose Poisson from the available distributions.

Model 2: Poisson, using observations 1–1200
Dependent variable: docvis

|  | Coefficient | Std. Error | $z$ | p-value |
|---|---|---|---|---|
| const | −0.00301416 | 0.0917868 | −0.03284 | 0.9738 |
| age | 0.0116388 | 0.00149144 | 7.804 | 0.0000 |
| female | 0.128273 | 0.0335225 | 3.826 | 0.0001 |
| public | 0.572624 | 0.0679804 | 8.423 | 0.0000 |

|  |  |  |  |
|---|---|---|---|
| Mean dependent var | 2.986667 | S.D. dependent var | 5.496059 |
| Sum squared resid | 35734.60 | S.E. of regression | 5.466116 |
| McFadden $R^2$ | 0.018933 | Adjusted $R^2$ | 0.018067 |
| Log-likelihood | −4532.404 | Akaike criterion | 9072.808 |
| Schwarz criterion | 9093.169 | Hannan–Quinn | 9080.478 |

Overdispersion test: $\chi^2(1) = 42.2804$ [0.0000]

The variables *age*, *female*, and *public* are all significantly different from zero and positive.

To obtain these results using a script:

```
open "@workdir\data\rwm88_small.gdt"
PR <- poisson docvis const age female public
```

The conditional mean of a Poisson regression is

$$E[y_0] = \exp(\beta_1 + \beta_2 age + \beta_3 female + \beta_4 public)$$

Choosing an appropriate point and estimating the $\beta$s by MLE is straightforward. For instance, a 29 year old female on public insurance is predicted to insure 2.816 doctor visits. Rounding to the nearest integer, we would predict 3 visits.

```
scalar p1 = exp(x1*b)
scalar p1_hat = round(p1)
printf "\nPoisson Regression\n\
\n x = %4.2g\n\
The predicted mean is %2.4g. This rounds to %2.4g\n",\
   x1, p1, p1_hat
```

The correlation between doctor visits and the number of (rounded) predictions is obtained by saving the model predictions using an accessor, rounding these, and taking correlations.

```
series yhat = $yhat
series round_yhat = round(yhat)
corr docvis yhat round_yhat
```

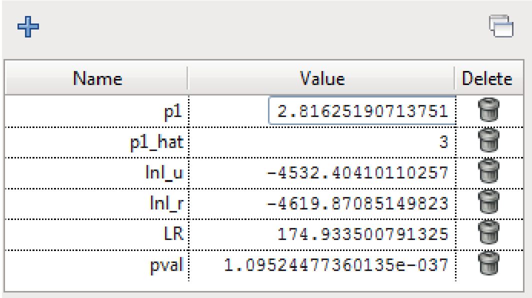
The correlation between *docvis* and rounded predictions is estimated to be 0.1179.

```
Correlation Coefficients, using the observations 1 - 1200
5% critical value (two-tailed) = 0.0566 for n = 1200

        docvis          yhat    round_yhat
        1.0000        0.1155        0.1179  docvis
                      1.0000        0.8996  yhat
                                    1.0000  round_yhat
```

The overall significance of the model can be determined using a likelihood ratio test. This involves estimating restricted and unrestricted models and computing the *LR* statistic.

```
Unrestricted <- poisson docvis const age female public
scalar lnl_u = $lnl
Restricted <- poisson docvis const
scalar lnl_r = $lnl
scalar LR = 2*(lnl_u-lnl_r)
scalar pval = pvalue(c,3,LR)
```

To view these, navigate to the session window and click on the **Scalars** icon. This opens a window that shows all currently defined scalars:



| Name | Value | Delete |
|---|---|---|
| p1 | 2.81625190713751 | |
| p1_hat | 3 | |
| lnl_u | -4532.40410110257 | |
| lnl_r | -4619.87085149823 | |
| LR | 174.933500791325 | |
| pval | 1.09524477360135e-037 | |

The *LR* statistic (174.93) and its *p*-value ($\approx 0$) are shown in the last two lines. The model is significant at the 5% level.

Marginal effect are computed as well. The marginal effect of a change in the $k^{th}$ (continuous) regressor for the $i^{th}$ individual is

$$\frac{\partial E(y_i|x)}{\partial x_{ij}} = \lambda_i \beta_j$$

where $\lambda_i = \exp(\beta_1 + \beta_2 x_{i2} + \cdots + \beta_k x_{ik})$ and $j = 1, 2, \cdots, k$. Based on this, take a 30 year old female on public insurance. The predicted marginal effect of being a year older is computed:

```
matrix x1 = { 1 , 30 , 1 , 1 }
scalar m1 = exp(x1*b)*b[2]
```

which equals 0.0331.

If the variable is discrete, then compute the discrete difference in probability functions for the two points. To determine the marginal effect of private insurance (vs public) for a 30-year-old woman:

```
matrix x1 = { 1 , 30 , 1 , 1 }
matrix x2 = { 1 , 30 , 1 , 0 }
scalar me_public_30=exp(x1*b)-exp(x2*b)
```

This is computed to be 1.242. Having public insurance increases the predicted number of doctor visits by 1.

The difference for a 70 year old woman

```
matrix x1 = { 1 , 70 , 1 , 1 }
matrix x2 = { 1 , 70 , 1 , 0 }
scalar me_public_70=exp(x1*b)-exp(x2*b)
```

The marginal impact is 1.98.

To obtain standard errors and confidence intervals for these, a proper function must be written. Below we have two functions. The first is used for continuous independent variables and the second is used if the variable is discrete.

```
# Poisson ME at point -- continuous variable
function scalar p_me_at(matrix b, matrix xx, scalar j)
    scalar me = exp(xx*b)*b[q]
    return me
end function
#-------------------------------------------------------
# Poisson ME at point -- indicator variable
function scalar p_me_at_d(matrix b, matrix x1, matrix x2)
```

```
    scalar me = exp(x1*b)-exp(x2*b)
    return me
end function
```

The first one, which is for continuous variables, is called p_me_at. It takes three arguments and returns a scalar. The first argument is the coefficients of the Poisson model. The second is a vector to evaluate the independent variables, and the last is an integer that locates the position in the parameter vector of the desired $\beta_j$.

The second function, p_me_at_d also has three arguments. The third argument, x2, is the second vector of x at which the second discrete point at which the ME is evaluated. So, for our 30-year-old female on public insurance we have:

```
m4 <- poisson docvis const age female public
matrix b = $coeff
matrix covmat = $vcv
matrix xx = { 1 , 30 , 1 , 1 }
scalar j = 2
matrix mfx = p_me_at(b, xx, j)
matrix jac = fdjac(b, p_me_at(b, xx, j))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)
```

which yields:

```
The 95% confidence interval centered at 0.033 is (0.0261, 0.0402)
```

For a 70-year-old female on public insurance we have:

```
matrix xx = { 1 , 70 , 1 , 1 }
scalar j = 2
matrix mfx = p_me_at(b, xx, j)
matrix jac = fdjac(b, p_me_at(b, xx, j))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)
```

which yields:

```
The 95% confidence interval centered at 0.053 is (0.0355, 0.0702)
```

For a discrete change, consider two 30-year-old women, one on public insurance and with private insurance. The script is:

```
matrix x1 = { 1 , 30 , 1 , 1 }
matrix x2 = { 1 , 30 , 1 , 0 }
matrix mfx = p_me_at_d(b, x1, x2)
matrix jac = fdjac(b, p_me_at_d(b, x1, x2))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)
```

which yields:

```
The 95% confidence interval centered at 1.242 is (1.0034, 1.4809)
```

For 70-year-old females on public vs private insurance:

```
matrix x1 = { 1 , 70 , 1 , 1 }
matrix x2 = { 1 , 70 , 1 , 0 }
matrix mfx = p_me_at_d(b, x1, x2)
matrix jac = fdjac(b, p_me_at_d(b, x1, x2))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)
```

which produces:

```
The 95% confidence interval centered at 1.979 is (1.5918, 2.3654)
```

The marginal impact of public insurance (more visits) now includes 2 in its estimated interval.

## 16.9 Tobit

The tobit model is 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, is recorded at that limit no matter what its 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, folks). 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.

### Example 16.16 in *POE5*

Hill et al. (2018) consider the following model of hours worked for a sample of women. equation (16.23).

$$hours_i = \beta_1 + \beta_2 educ_i + \beta_3 exper_i + \beta_4 age_i + \beta_5 kidsl6_i + e_i \qquad (16.23)$$

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)
            --cluster=clustvar (see logit for explanation)
            --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 *POE5*. 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 "@workdir\data\mroz.gdt"
list xvars = const educ exper age kidsl6
tobit hours xvars
```

The results from the 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

$$\frac{\partial E(hours_i)}{\partial educ_i} = \Phi(\widehat{hours}_i)\hat{\beta}_2, \tag{16.24}$$

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.
```

A slightly easier way to evaluate the index, $\widehat{hours}_0$, is to use matrices. In the alternative version the data are converted to a matrix and a vector of means is created. 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)<br>Tobit | (2)<br>OLS | (3)<br>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 |
| $\ell$ | $-3827$ | $-3426$ | $-6054$ |

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

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 women in the labor force, more schooling has no effect on hours worked. If the entire population of women is considered, more schooling increases hours worked, presumably by enticing more women into the labor force.

## 16.10 Selection Bias

### Example 16.17 in *POE5*

Selection bias occurs when for some observations we do not have data on the dependent for some reason. The statistical problems occur when the cause of the sample limitation is correlated with the dependent variable. Ignoring the correlation, the model might be estimated using least squares, tobit or truncated least squares. In any event, obtaining consistent estimates of the regression parameters is not possible when cause of the missing observations is correlated with the dependent variable of the regression model. 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

$$z_i^* = \gamma_1 + \gamma_2 w_i + u_i, \quad i = 1, \ldots, N \tag{16.25}$$

where $z_i^*$ is a **latent variable**, $\gamma_1$ and $\gamma_2$ 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

$$z_i = \begin{cases} 1 & z_i^* > 0 \\ 0 & \text{otherwise} \end{cases} \tag{16.26}$$

The second equation, called the **regression equation**, is the linear model of interest. It is

$$y_i = \beta_1 + \beta_2 x_i + e_i, \quad i = 1, \ldots, n, \quad N > n \tag{16.27}$$

where $y_i$ is an observable random variable, $\beta_1$ and $\beta_2$ 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

$$\begin{bmatrix} u_i \\ e_i \end{bmatrix} \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & \sigma_e^2 \end{pmatrix}\right] \tag{16.28}$$

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 $\beta$ in (16.27) 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:

$$E[y_i|z_i > 0] = \beta_1 + \beta_2 x_i + \beta_\lambda \lambda_i \tag{16.29}$$

where

$$\lambda_i = \frac{\phi(\gamma_1 + \gamma_2 w_i)}{\Phi(\gamma_1 + \gamma_2 w_i)} \tag{16.30}$$

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:

$$y_i = \beta_1 + \beta_2 x_i + \beta_\lambda \lambda_i + v_i \tag{16.31}$$

It can be shown that (16.31) is heteroskedastic and if $\lambda_i$ were known (and nonstochastic), then the selectivity corrected model (16.31) could be estimated by generalized least squares. Alternately, the heteroskedastic model (16.31) 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, $\lambda_i$ is not known and must be estimated using the sample. The stochastic nature of $\lambda_i$ in (16.31) 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 (16.30) to estimate the inverse Mill's ratio, $\hat{\lambda}_i$.

2. Add $\hat{\lambda}_i$ to the regression model as in equation (16.31) 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 *POE5* uses the *mroz.gdt* data. First, 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** generates an error when taking the natural logs. This can be safely ignored since **gretl** 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 "@workdir\data\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 there are 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 "@workdir\data\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 |

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)
            --two-step (perform two-step estimation)
            --vcv (print covariance matrix)
            --opg (OPG standard errors)
            --robust (QML standard errors)
            --cluster=clustvar (see logit for explanation)
            --verbose (print extra output)
Examples:   heckit y 0 x1 x2 ; ys 0 x3 x4
```

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
```

The `list` function is used to hold the variables of the regression and selection equations.

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 Mill's ratio is significant, which is evidence of selection bias.

To use the pull-down menus, select **Model>limited dependent variable>Heckit** from **gretl**'s main window. This will reveal the dialog shown in figure 16.8. 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 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.

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.

## 16.11 Simulation

### Appendix 16D

In this simulation **gretl** is used to show that least squares is biased when the sample is censored. The simulated data are generated

$$y_i^* = -9 + 1x_i + e_i \tag{16.32}$$

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. Line 7 uses the conditional assignment operator, `series yc = (y > 0) ? y : 0` is a logical statement that generates ‘y’ or ‘0’ depending on whether the statement in parentheses is true. 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.

```
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
```

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 $\sigma$. To test for unbiasedness of the tobit estimator of the slope ($H_o : b_2 = 1$ against the two-sided alternative) compute:

$$\sqrt{NMC}(\bar{b}_2 - 1)/\hat{\sigma} = \sqrt{1000}(1.00384 - 1)/0.0737160 = 1.647 \sim N(0,1) \tag{16.33}$$

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.

## 16.12 Script

First, here are all of the functions used in this chapter. They have been collected into a script called *functions_ch16.inp* that is included with the other script files distributed with this manual (also reproduced in section D.2). You will need to run these before using the second part of the script.

Functions used in Chapter 16

```
set echo off
# This function computes a t-dist confidence interval based on a statistic
function void t_interval (scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# This function computes t-dist confidence intervals after a model
function matrix t_interval_m (matrix b "Coefficients",
      matrix v "Variance-covariance matrix",
      int df "Degrees-of-freedom",
      scalar p "Coverage probability for CI")

    scalar alpha = (1-p)               # Convert p to alpha
    matrix c = critical(t,df,alpha/2)  # alpha/2 critical value
    matrix se = sqrt(diag(v))          # standard errors
    matrix lb = b - c*se               # lower bound
    matrix ub = b + c* se              # upper bound
    matrix result = b ~ se ~ lb ~ ub   # put into matrix

    cnameset(result, "Estimate StdErr (Lower, Upper) ")
    rnameset(result, "b")
    printf "\nThe %2g%% confidence intervals\
(t-distribution)\n%10.4f\n", p*100, result
    return result
end function

function matrix ame_binary(matrix *b "parameter estimates",
      list x "Variables list",
      int dist[1:2:2] "distribution" )
# Computes average marginal effects for probit or logit
    matrix p = lincomb(x, b)       # The index function
    matrix d = (dist==1) ? exp(-p)./(1.+exp(-p)).^2 : dnorm(p)
    matrix ame_matrix = d*b'
    cnameset(ame_matrix, x)            # add column names
    matrix amfx = meanc(ame_matrix)    # find the means
    cnameset(amfx, x)                  # add the column names to amfx
    printf "\n Average Marginal Effects (AME):\
      \n Variables: %s\n%12.4g \n", varname(x), amfx
    return amfx
end function

function matrix ame_cov (matrix b "parameter estimates",
      matrix covmat "Covariance",
      list x "Variables list",
      int dist[1:2:2] "distribution" )
    # Computes std errs for AME probit/logit
    # Requires ame_binary
    matrix amfx = ame_binary(&b, x, dist)
    matrix jac = fdjac(b, ame_binary(&b, x , dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    matrix results = amfx' ~ se
    rnameset(results, "b")
    cnameset(results, "AME StdErr")
    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    printf "%10.4f\n", results
    return amfx|variance
end function

function scalar p_binary(matrix b "parameter estimates",
      matrix x "Representative Point",
      int dist[1:2:2] "distribution" )
    # Computes the probability of a binary choice: 1 = logit
    scalar p = x*b                  # The index function
    scalar d = (dist==1) ? 1./(1.+exp(-p)) : cnorm(p)
    return d
end function

function void Probs (matrix b "parameter estimates",
      matrix covmat "Covariance",
      matrix x "Representative Point",
      scalar df "Degrees of Freedom",
      int dist[1:2:2] "distribution")
    # Function computes std errors of binary predictions
    # Requires p_binary
    scalar p = p_binary(b, x, dist)
    matrix jac = fdjac(b, p_binary(b, x , dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif

    printf "95%% t(%.2g) confidence interval for probability at\n\
    x = %8.4f\n", df, x
    cnameset(results, " Lower ME Upper StdError" )
    printf "%10.4f\n", results
end function

function scalar me_at(matrix *param "parameter estimates",
      matrix xx "Representative Point",
      scalar q "Parameter of interest",
      int modl[1:2:2] "distribution" )
    # Marginal effects at a point -- continuous variables only
    scalar idx = xx*param
    scalar d = (modl==1)? (exp(-idx)./(1.+exp(-idx)).^2)*param[q] :\
      dnorm(idx)*param[q]
    return d
end function

function void MER (matrix *b "parameter estimates",
      matrix covmat "Covariance",
      matrix x "Representative Point",
      int q "Parameter of interest",
      int df "Degrees of Freedom",
      int modl[1:2:2] "distribution")
    # Std errors for Marginal effects at a point -- continuous vars only
    scalar p = me_at(&b, x, q, modl)
    matrix jac = fdjac(b, me_at(&b, x , q, modl))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se
    if modl == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    printf "95%% t(%.2g) confidence interval for b%.g at\n x =\
    %9.2g \n", df, q, x
    cnameset(results, " Lower ME Upper StdError" )
    printf "%10.4f\n", results
end function

function void MER_lpmfx (matrix b "parameter estimates",
      list  XL "list of regressors",
      matrix covmat "Covariance matrix",
      matrix x_at "Representative point",
      int dist[1:2:1] "distribution",
      int df "degrees-of-freedom")
    # The MER function to be used with lp-mfx.gfn
    # available from gretl's function server
    matrix me = binary_dp_dx(b, XL, x_at, dist)
    matrix jac = fdjac(b, binary_dp_dx(b, XL, x_at, dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    matrix results = me' ~ se
    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    scalar crit = critical(t,df,0.025)
    matrix results = (me'-crit*se) ~ me' ~ (me'+crit*se) ~ se
    cnameset(results, "Lower ME Upper StdErr")
    rnameset(results, XL[2:nelem(XL)])
    cnameset(x_at, XL )
    printf "Representative Point\n%11.2g\n95%% CI for\
MER\n%10.4g\n",x_at, results
end function


# Poisson ME at point -- continuous variable
function scalar p_me_at(matrix b, matrix xx, scalar q)
    scalar me = exp(xx*b)*b[q]
    return me
end function

# Poisson ME at point -- indicator variable
function scalar p_me_at_d(matrix b, matrix x1, matrix x2)
    scalar me = exp(x1*b)-exp(x2*b)
    return me
end function

function list mlogitprob(series y "Dependent variable",
      list x "List of regressors",
      matrix theta "Coefficient vector")
    # computes probabilities of each choice for all data
    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 "Dependent variable",
   matrix x "Representative point 1xk",
   matrix theta "Coefficient vector")
  # computes probabilities of each choice at a representative point
  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 series mlogitlogprobs(series y "Dependent Variable",
   matrix X "Independent variables",
   matrix theta "Parameters")
  # This function computes the log probabilities for MLE
  # estimation of MNL
  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
  endloop
  return ret
end function

function matrix mnl_se_lpfmx (matrix b "parameter estimates",
   matrix covmat "Covariance of MNL",
   list XL "list of regressors",
   matrix x "vector of x-values",
   int j "1-based index of outcome",
   int m "number of possible outcomes",
   int df "degrees of freedom for CI" )
# Computes MER and std errors for MNL
# must install and use lp-mfx.gfn
  matrix p = mlogit_dpj_dx(b, XL, x, j, m)
  matrix jac = fdjac(b, mlogit_dpj_dx(b, XL, x, j, m))
  matrix variance = qform(jac,covmat)
  matrix se = sqrt(diag(variance))
  scalar crit = critical(t,df,0.025)
  matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

  cnameset(results, "Lower ME Upper StdErr")
  printf "95%% CI for MER\n%10.4f\n", results
  return results
end function

# Several Functions for conditional logit.
# These are NOT general
# clprobs       --Conditional logit probability scalar
# clprobs_mat   --Conditional logit probabilities matrix
# clprobs_at    --marginal effects at a point -> 1x3 vector
# cl_me         --marginal effects continuous w/std errors
# cl_me_d       --marginal effects discrete   w/std errors

function scalar clprobs(list y "list of choices",
   list x "list of independent variables",
   matrix theta "parameters")
  # computes the probabilities for Conditional Logit
  # Used in user written MLE
  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
  endloop
  return sumc(ln(P))
end function

function matrix clprobs_mat(list x, matrix theta)
  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 = i1/d ~ i2/d ~ i3/d
    matrix P = P | pp
  endloop
  return P
end function

function matrix clprobs_at(matrix x, matrix theta)
  scalar i1 = exp(theta[1]+theta[3]*x[1])
  scalar i2 = exp(theta[2]+theta[3]*x[2])
  scalar i3 = exp(theta[3]*x[3])
  scalar  d = i1+i2+i3
  matrix pp = i1/d ~ i2/d ~ i3/d
  return pp
end function

function scalar cl_me(matrix *x "vector for the desired point",
   matrix *theta "parameters",
   int q "variable index for own price",
   int p "variable index for other price")
  # Marginal effects for CL model -- continuous case
  # Function only works for 3 choice beverage model in poe
  # Inputs: x = point at which to evaluate
  # theta: Cond Logit MLE
  # q: own price index
  # p: other price index
  # op: 1 if own price, 0 otherwise
  matrix mm = clprobs_at(x, theta)
  if p == q
    scalar me = mm[q]*(1-mm[q])*theta[3]  # own price pepsi
  else
    scalar me = -mm[p]*mm[q]*theta[3]      # cross price 7up
  endif
  return me
end function

function matrix cl_me_d(matrix *x1,
   matrix *x2,
   matrix *theta)
  # Marginal effects for CL model -- discrete case
  matrix mm = clprobs_at(x1, theta)
  matrix m2 = clprobs_at(x2, theta)
  mat = m2-mm
  return mat
end function

function matrix op_se_lpfmx (matrix b "parameter estimates",
   matrix covmat "Covariance of MNL",
   list XL "list of regressors",
   matrix x "vector of x-values",
   int j "1-based index of outcome",
   int m "number of possible outcomes",
   int df "degrees of freedom for CI",
   int dist[1:2:1] "distribution" )
  # Computes marginal effects and std errors for ordered probit/logit
  # must install and use lp-mfx.gfn
  matrix p = ordered_dpj_dx(b, XL, x, j, m, dist)
  matrix jac = fdjac(b, ordered_dpj_dx(b, XL, x, j, m, dist))
  matrix variance = qform(jac,covmat)
  matrix se = sqrt(diag(variance))
  scalar crit = critical(t,df,0.025)
  matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

  cnameset(results, "Lower ME Upper StdErr")
  printf "95%% CI for MER\n%10.4f\n", results
  return results
end function
```

Once the functions have been run, the script below should produce all of the results in the chapter.

```
# run the functions_ch16.inp first!
include functions_ch16.inp

open "@workdir\data\transport.gdt"
set verbose off
summary --simple

# Example 16.1
m1 <- ols auto const dtime --robust
series y_pred = $yhat>0.5
series incorrect = abs(auto-y_pred)
summary incorrect --by=auto --simple

scalar correct = $nobs-sum(abs(auto-y_pred))
printf "The number correct predictions =\
%g out of %g commuters\n", correct, $nobs
t_interval_m($coeff,$vcv,$df,.95)

scalar pr = $coeff(const)+$coeff(dtime)*1
printf "\n The predicted probability of auto travel if public\
transportation\n takes 10 minutes longer = %.4f \n", pr

printf "\n R2 = %.4f \n", $rsq

# Example 16.3
nulldata 3
series y x
series y[1]=1
series y[2]=1
series y[3]=0
series x[1]=1.5
series x[2]=.6
series x[3]=.7
probit y const x

# Example 16.4
open "@workdir\data\transport.gdt"
list xvars = const dtime
m2 <- probit auto xvars
t_interval_m($coeff,$vcv,$df,.95)

scalar p1=cnorm($coeff(const))
scalar i_20 = $coeff(const)+$coeff(dtime)*2
scalar d_20 = dnorm(i_20)*$coeff(dtime)
printf "\n The value of the index for dtime = 20 minutes is %6.4f\n\
The predicted probability of driving is = %6.4f\n\
The marginal effect on probability of driving is %6.4f \n",\
   i_20, cnorm(i_20), d_20

scalar i_30 = $coeff(const)+$coeff(dtime)*3
printf "\n The predicted probability of driving if dtime = 30\
minutes is %6.4f\n", cnorm(i_30)
printf "\n The difference in probability is %6.4f\n",\
       cnorm(i_30)-cnorm(i_20)

# Example 16.5
# Estimated Probability of driving
set echo off
open "@workdir\data\transport.gdt"
list x = const dtime
probit auto x
matrix b = $coeff
series me = dnorm(lincomb(x,b))*b[2]
scalar amf = mean(me)
printf "\n The average marginal effect for change in dtime =\
%6.4f\n", amf
summary me --simple

# probit AME using function
list x = const dtime
probit auto x --quiet
matrix b = $coeff
scalar dist = ($command == "logit")? 1 : 2
matrix me_probit = ame_binary(&b, x, dist)

# using Delta method to get standard errors for AME
open "@workdir\data\transport.gdt"
list x = const dtime
probit auto x --quiet
matrix b = $coeff
matrix covmat = $vcv
scalar dist = ($command == "logit")? 1 : 2
matrix amfx = ame_binary(&b, x, dist)
matrix jac = fdjac(b, ame_binary(&b, x , dist))
matrix variance = qform(jac, $vcv)
matrix se = sqrt(diag(variance))
printf "\n The average marginal effects:\n%10.4f\
delta estimated standard errors: \n%10.4f \n", amfx, se'

# confidence interval for AME
t_interval_m(amfx',variance,$df,.95)

# MER: marginal effects and std errors at specific points
open "@workdir\data\transport.gdt"
list x = const dtime
m1 <- probit auto x --quiet
matrix bp = $coeff
matrix xi = { 1, 2 }
scalar dist = ($command == "logit")? 1 : 2
MER(&bp,$vcv,xi,2,$df,dist)

# MEM: Marginal effects at the means
matrix xi = { 1, mean(dtime) }
MER(&bp,$vcv,xi,2,$df,dist)

# MEM: using lp-mfx
include lp-mfx.gfn
open "@workdir\data\transport.gdt"
list x = const dtime
probit auto x --quiet
scalar dist = ($command == "logit")? 1 : 2
binary_mfx(auto, $xlist, $coeff, $vcv, $sample, dist)

bundle b1 = binary_mfx(auto, $xlist, $coeff, $vcv, $sample, dist)
lp_mfx_print(&b1)

# MER using lp-mfx
probit auto x --quiet
scalar dist = ($command == "logit")? 1 : 2
matrix x_at = { 1 , 2}
MER_lpmfx($coeff,$xlist,$vcv,x_at,dist,$df)

# predicted probability and its confidence interval
probit auto x --quiet
matrix x_at = { 1 , 3}
scalar dist = ($command == "logit")? 1 : 2
Probs($coeff,$vcv,x_at,$df,dist)

# Example 16.6
# comparing probit, logit, ols
set echo off
```

```
open "@workdir\data\coke.gdt"
list x = const pratio disp_pepsi disp_coke
m1 <- probit coke x --quiet
m2 <- logit coke x --quiet
m3 <- ols coke x --robust

# AME for probit/logit
m1 <- probit coke x --quiet
matrix bp = $coeff
matrix covmat = $vcv
scalar dist = ($command == "logit")? 1 : 2

matrix c=ame_cov(bp,$vcv,x,dist)
t_interval_m(c[1,]',c[-1,],$df,.95)

# Compute MER for probit and logit
matrix x_at = { 1, 1.1, 0, 0}
probit coke x
scalar dist = ($command == "logit")? 1 : 2
MER_lpmfx($coeff,$xlist,$vcv,x_at,dist,$df)

logit coke x
scalar dist = ($command == "logit")? 1 : 2
MER_lpmfx($coeff,$xlist,$vcv,x_at,dist,$df)

# Compute probabilities and std errors for probit/logit
matrix xi = { 1, 1.1, 0, 0}
probit coke x
matrix bp = $coeff
scalar dist = ($command == "logit")? 1 : 2
Probs(bp,$vcv,xi,$df,dist)

logit coke x
matrix lp = $coeff
scalar dist = ($command == "logit")? 1 : 2
Probs(lp,$vcv,xi,$df,dist)

# Compute probability and std errors for LPM
ols coke x --robust
matrix b_ols = $coeff
matrix pred = xi*b_ols
matrix v = (qform(xi,$vcv))
matrix se = sqrt(v)
printf "ME of OLS %.4f with std error %.4f\n", xi*b_ols, se

# Compute ME at representative value for probit/logit
matrix xi = { 1, 1.1, 0, 0}

probit coke x
matrix bp = $coeff
scalar dist = ($command == "logit")? 1 : 2
```

```
MER(&bp,$vcv,xi,2,$df,dist)

# 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

# Example 16.7
# test hypotheses with probit
open "@workdir\data\coke.gdt"
list x = const pratio disp_pepsi disp_coke
probit coke x

# H1 Test of significance
scalar tv = $coeff(disp_coke)/$stderr(disp_coke)
printf "Ho: b3 = 0    Ha: b3>0\n \
  t  = %.4f\n \
  p-value = %.4f\n", \
  tv, pvalue(t,$df,tv)

printf "Ho: b3 = 0    Ha: b3 != 0\n \
  t  = %.4f\n \
  p-value = %.4f\n", \
  tv, 2*pvalue(t,$df,abs(tv))
printf "The 5%% critical value from the t(%g) is %.4f\n",\
   $df, critical(t,$df,.025)

printf "The 5%% critical value from the chi-square(1) is %.4f\n",\
   critical(C,1,.05)

restrict --quiet
    b[disp_coke]=0
end restrict
# H2 Economic Hypothesis: b3=-b4

probit coke x --quiet
restrict
    b[3]+b[4]=0
end restrict

# H3 Joint significance: b3=b4=0
probit coke x
restrict --quiet
    b[3]=0
    b[4]=0
end restrict
printf "The 5%% critical value from the chi-square(2) is %.4f\n",\
    critical(C,2,.05)

# H4 Model significance: b2=b3=b4=0
probit coke x
restrict --quiet
    b[2]=0
    b[3]=0
    b[4]=0
end restrict
printf "The 5%% critical value from the chi-square(3) is %.4f\n",\
    critical(C,3,.05)

# Example 16.8
# LR tests
open "@workdir\data\coke.gdt"
list x = const pratio disp_pepsi disp_coke
# H1 Test of significance
probit coke x --quiet
scalar llu = $lnl
probit coke const pratio disp_pepsi --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b3 = 0    Ha: b3 != 0\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,1,lr)

# H2 Economic Hypothesis: b3=-b4
series c_p = disp_pepsi-disp_coke
probit coke x --quiet
scalar llu = $lnl
probit coke const pratio c_p --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b3+b4 = 0    Ha: b3+b4 != 0\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,1,lr)

# H3 Joint significance: b3=b4=0
probit coke x --quiet
scalar llu = $lnl
probit coke const pratio --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b3 = b4 = 0  vs.  Ha: b3 != 0, b4 != 0\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,2,lr)

# H4 Model significance: b2=b3=b4=0
```

```
probit coke x --quiet
scalar llu = $lnl
probit coke const --quiet
scalar llr = $lnl
scalar lr = 2*(llu-llr)
printf "Ho: b2=b3=b4=0  vs.  Ha: not Ho\n \
  LR  = %.4f\n \
  p-value = %.4f\n", \
  lr, pvalue(C,3,lr)

# Example 16.9
open "@workdir\data\mroz.gdt"
square exper
list x = const educ exper sq_exper kidsl6 age
list inst = const exper sq_exper kidsl6 age mothereduc
LPM_IV <- tsls lfp x ; inst --robust

FirstStage <- ols educ inst
# IV Probit
list exogvars = 0 exper sq_exper kidsl6 age
include HIP.gfn
b=HIP(lfp, exogvars, educ, mothereduc)

# Example 16.12
# Multinomial Logit
open "@workdir\data\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
printf "Average Marginal Effects"
summary d* --simple

# mnl predictions at points
open "@workdir\data\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 = %.3g are\n %8.4f\n" ,quantile(grades,.05), p05
printf "\nThe predicted probabilities for student\
grades = %.3g are\n %8.4f\n",quantile(grades,.50), p50

# mnl marginal effects at points
open "@workdir\data\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
rnameset(me,"MER")
cnameset(me,"NoColl 2Year 4Year ")
printf "\nThe marginal effect of grades for student\
grades =%5.2f\n\
%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
cnameset(me,"NoColl 2Year 4Year ")
rnameset(me,"MER")
printf "\nThe marginal effect of grades for student\
grades =%5.2f\n\
%8.4f\n", q05, me


# mnl logit with user written likelihood
open "@workdir\data\nels_small.gdt"
series psechoice = psechoice -1
list x = const grades
smpl full
matrix X = { x }
scalar k = cols(X)
matrix theta = zeros(2*k, 1)
ml <- mle loglik = mlogitlogprobs(psechoice, X, theta)
    params theta
```

```
end mle --hessian

varlist --type=accessor
# MNL marginal effects at the means and predicted probabilities
# using lp-mfx.gfn
include lp-mfx.gfn
open "@workdir\data\nels_small.gdt"
list x = const grades
logit psechoice x --multinomial
bundle b = mlogit_mfx(psechoice, $xlist, $coeff, $vcv, $sample)
lp_mfx_print(&b)

matrix x1 = { 1 , 6.64 }
matrix c1 = mlogit_dpj_dx($coeff, $xlist, x1, 2, 3)
matrix x2 = {1, 2.635 }
matrix c2 = mlogit_dpj_dx($coeff, $xlist, x2, 2, 3)
print c1 c2

# MER and std errors using lp_mfx.gfn
include lp-mfx.gfn
open "@workdir\data\nels_small.gdt"
list x = const grades
matrix x_at = {1, 6.64 }
logit psechoice x --multinomial
c = mnl_se_lpfmx( $coeff, $vcv, x, x_at, 2, 3, $df)

# Example 16.13
# conditional logit
open "@workdir\cola_mixed.gdt"
list y = d_Pepsi d_7Up d_Coke
list x =  pepsi sevenup coke
matrix theta = {-2, .3, .1}

mle lln = clprobs(y, x, theta)
    params theta
end mle

matrix theta = $coeff
matrix covmat = $vcv

# probabilities of purchasing drinks at the given price
matrix x1 = {1.0, 1.25, 1.10}
matrix x2 = {1.0, 1.25, 1.25}
matrix mm = clprobs_at(x1,theta)
cnameset(mm, " Pepsi 7-Up Coke")
print mm

# marginal effects of increase in pepsi price
scalar me_op_pepsi = mm[1]*(1-mm[1])*theta[3]  # own price pepsi
scalar me_cp_7up  = -mm[2]*mm[1]*theta[3]      # cross price 7up
scalar me_cp_coke = -mm[1]*mm[3]*theta[3]      # cross price coke
```

```
printf "\n Own-Price (Pepsi) marginal effect (1$) = %.3f\n\
Cross-Price (7up) effect ($1) = %.3f\n\
Cross-Price (Coke) effect ($1)= %.3f\n",\
   me_op_pepsi, me_cp_7up, me_cp_coke

# Confidence Intervals for CL
# Marginal effects and standard errors: calculus version
matrix x2 = {1.0, 1.25, 1.10}
scalar q = 1       # Own price:   1 = pepsi, 2 = 7up, 3 = coke
scalar p = 1       # Other price: 1 = pepsi, 2 = 7up, 3 = coke
scalar c = cl_me(&x2, &theta, q, p)
matrix jac = fdjac(theta, cl_me(&x2, &theta, q, p))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(c,se,$df,.95)

# Confidence Intervals for CL
# Marginal effects and standard errors: discrete change version
matrix x2 = {1.0, 1.25, 1.25}
matrix x1 = {1.0, 1.25, 1.10}
matrix c2 = cl_me_d(&x1, &x2, &theta)
matrix jac = fdjac(theta, cl_me_d(&x1, &x2, &theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = c2’ ˜ se ˜ c2’./se
cnameset(results, " m_effect std_error t-ratio" )
rnameset(results, "Pepsi 7UP Coke" )
print results

# Marginal effects and standard errors: discrete change version
# increase pepsi price to 1.10
matrix x2 = {1.1, 1.25, 1.10}
matrix x1 = {1.0, 1.25, 1.10}
matrix c2 = cl_me_d(&x1, &x2, &theta)
matrix jac = fdjac(theta, cl_me_d(&x1, &x2, &theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = c2’ ˜ se ˜ c2’./se
cnameset(results, " m_effect std_error t-ratio" )
rnameset(results, "Pepsi 7UP Coke" )
print results

# probabilities and ME of an
# increase coke price to 1.25
matrix x1 = {1.0, 1.25, 1.10}
matrix m1 = clprobs_at(x1,theta)

matrix x3 = {1.0, 1.25, 1.25}
matrix m3 = clprobs_at(x3,theta)
cnameset(m3, " pepsi 7up coke")
print m3
```

```
mat = m3-m1
cnameset(mat, " pepsi 7up coke")
print mat

# Probability of pepsi purchase
# Probabilities and delta std errors
matrix x2 = {1.0, 1.25, 1.10}
matrix m2 = clprobs_at(x2, theta)
matrix jac = fdjac(theta, clprobs_at(x2, theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = m2’ ~ se
cnameset(results, " Probability std_error" )
rnameset(results, "Pepsi 7UP Coke" )
print results

# Probability of pepsi purchase
# Pepsi price up by .1
# Probabilities and delta std errors
matrix x3 = {1.10, 1.25, 1.10}
matrix m3 = clprobs_at(x3, theta)
matrix jac = fdjac(theta, clprobs_at(x3, theta))
matrix variance = qform(jac,covmat)
matrix se = sqrt(diag(variance))
matrix results = m3’ ~ se
cnameset(results, " Probability std_error" )
rnameset(results, "Pepsi 7UP Coke" )
print results


# increase coke price to 1.25
matrix x1 = {1.0, 1.25, 1.10}
matrix m1 = clprobs_at(x1,theta)

matrix x4 = {1.0, 1.25, 1.25}
matrix m4 = clprobs_at(x4,theta)
cnameset(m4, " pepsi 7up coke")
print m4
mat = m4-m1
cnameset(mat, " pepsi 7up coke")
print mat

# Example 16.14
#Ordered Probit
open "@workdir\data\nels_small.gdt"
op <- 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

include lp-mfx.gfn
probit psechoice grades
matrix theta = $coeff
matrix x = {6.64}
scalar dist = ($command == "logit")? 1 : 2
op_se_lpfmx($coeff, $vcv, $xlist, x, 3, 3, $df, dist)

# Example 16.15
# Poisson Regression -- means and marginal effect
open "@workdir\data\rwm88_small.gdt"
poisson docvis const age female public
matrix b = $coeff
matrix x1 = { 1 , 29 , 1 , 1 }
scalar p1 = exp(x1*b)
scalar p1_hat = round(p1)
printf "\nPoisson Regression\n\
\n x = %4.2g\n\
The predicted mean is %2.4g. This rounds to %2.4g\n",\
   x1, p1, p1_hat

series yhat = $yhat
series round_yhat = round(yhat)
corr docvis yhat round_yhat

Unrestricted <- poisson docvis const age female public
scalar lnl_u = $lnl
Restricted <- poisson docvis const
scalar lnl_r = $lnl
scalar LR = 2*(lnl_u-lnl_r)
scalar pval = pvalue(c,3,LR)

matrix x1 = { 1 , 30 , 1 , 1 }
scalar m1 = exp(x1*b)*b[2]

matrix x2 = { 1 , 30 , 1 , 0 }
scalar me_public_30=exp(x1*b)-exp(x2*b)
```

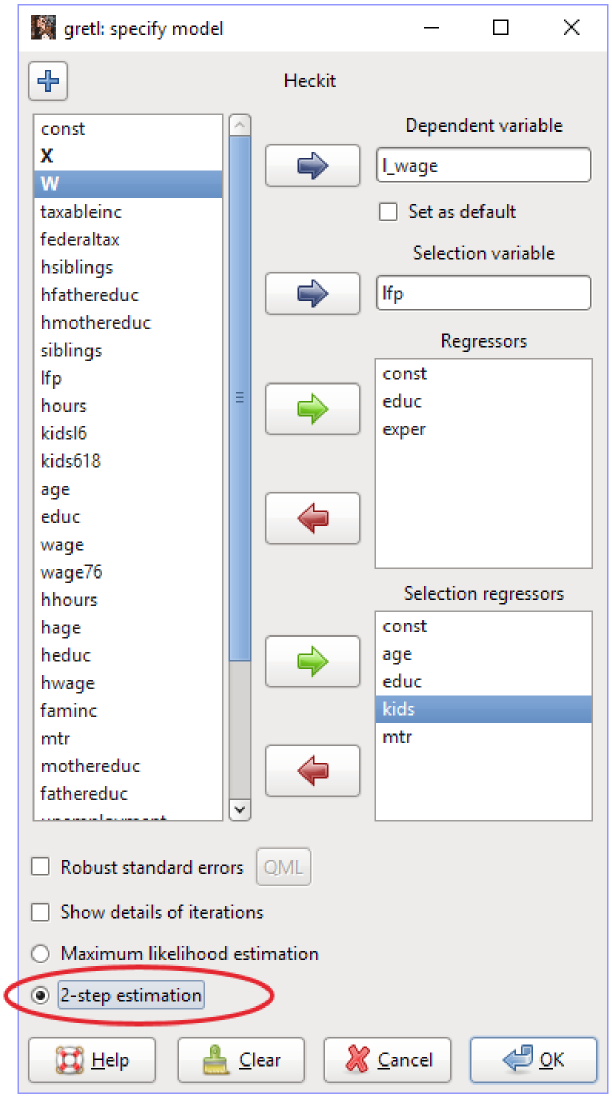
```

matrix x1 = { 1 , 70 , 1 , 1 }
matrix x2 = { 1 , 70 , 1 , 0 }
scalar me_public_70=exp(x1*b)-exp(x2*b)

m4 <- poisson docvis const age female public
matrix b = $coeff
matrix covmat = $vcv
matrix xx = { 1 , 30 , 1 , 1 }
scalar j = 2
matrix mfx = p_me_at(b, xx, j)
matrix jac = fdjac(b, p_me_at(b, xx, j))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)

matrix xx = { 1 , 70 , 1 , 1 }
scalar j = 2
matrix mfx = p_me_at(b, xx, j)
matrix jac = fdjac(b, p_me_at(b, xx, j))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)

# ME indicator
matrix x1 = { 1 , 30 , 1 , 1 }
matrix x2 = { 1 , 30 , 1 , 0 }
matrix mfx = p_me_at_d(b, x1, x2)
matrix jac = fdjac(b, p_me_at_d(b, x1, x2))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)

matrix x1 = { 1 , 70 , 1 , 1 }
matrix x2 = { 1 , 70 , 1 , 0 }
matrix mfx = p_me_at_d(b, x1, x2)
matrix jac = fdjac(b, p_me_at_d(b, x1, x2))
matrix variance = qform(jac,covmat)
matrix se = sqrt(variance)
t_interval(mfx,se,$df,.95)

# Example 16.16
#Tobit
open "@workdir\data\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

# Example 16.17
#Heckit
open "@workdir\data\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

# Example 16.17
#Heckit
open "@workdir\data\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
```

```

# Example 16.18
# 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
```

Finally, this is the script used to rearrange the data for conditional logit.

```
# Generates data for Conditional Logit by id
open "@workdir\data\cola.gdt"
matrix ids = values(id)
matrix idx_pepsi  = seq(1, 5466, 3)
matrix idx_7up    = seq(2, 5466, 3)
matrix idx_coke   = seq(3, 5466, 3)
matrix Price      = { price }
matrix Choice     = { choice }
matrix cokePrice    = Price[idx_coke]
matrix pepsiPrice   = Price[idx_pepsi]
matrix sevenupPrice = Price[idx_7up]
matrix cokeChoice     = Choice[idx_coke]
matrix pepsiChoice    = Choice[idx_pepsi]
matrix sevenupChoice  = Choice[idx_7up]

nulldata 1822 --preserve
series coke = cokePrice
series pepsi = pepsiPrice
series sevenup = sevenupPrice
series d_coke = cokeChoice
series d_pepsi = pepsiChoice
series d_sevenup = sevenupChoice

setinfo d_pepsi -d "1 if Pepsi, 0 otherwise"
setinfo d_sevenup -d "1 if 7-Up, 0 otherwise"
setinfo d_coke -d "1 if Coke, 0 otherwise"
setinfo pepsi -d "Pepsi price"
setinfo sevenup -d "7-Up price"
setinfo coke -d "Coke price"
# store cola_mixed
```



gretl: specify model

Heckit

Dependent variable: l_wage

□ Set as default

Selection variable: lfp

Regressors: const, educ, exper

Selection regressors: const, age, educ, kids, mtr

□ Robust standard errors

□ Show details of iterations

○ Maximum likelihood estimation

● 2-step estimation

Figure 16.8: Choose **Model>Limited dependent variable>Heckit** from **gretl**'s main window to reveal the dialog box for Heckit.

# 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 |
| `midasreg` | OLS and NLS of mixed data sampling | | |

## 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 |
| `genr` | Generate a new variable | `info` | Information on data set |
| `join` | Add data from a file | `labels` | Print labels for variables |
| `info` | Information on data set | `markers` | Write obs markers to file |
| `nulldata` | Creating a blank dataset | `open` | Open a data file |
| `setinfo` | Edit attributes of variable | `rename` | Rename variables |
| `scalar` | Generate a scalar | `setobs` | Set frequency and starting observation |
| `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 |
| `hfplot` | MIDAS plot | `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 | | |

## Prediction

| | |
|---|---|
| `fcast` | Generate forecasts |

## Utilities

| | | | |
|---|---|---|---|
| `critical` | Computes critical values | `eval` | Evaluate expression |
| `help` | Help on commands | `pkg` | |
| `modeltab` | The model table | `pvalue` | Compute $p$-values |
| `quit` | Exit the program | `shell` | Execute shell commands |

## 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 |

# 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 $\mu$, 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, $\mu$ 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 $\mu$ 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 quite 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 $\chi^2$ 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 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`.

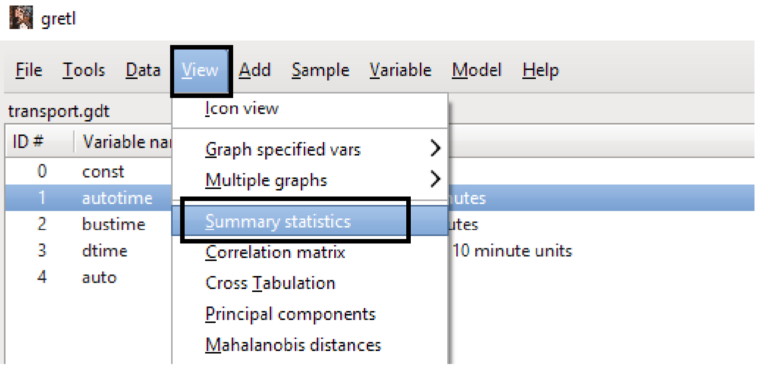
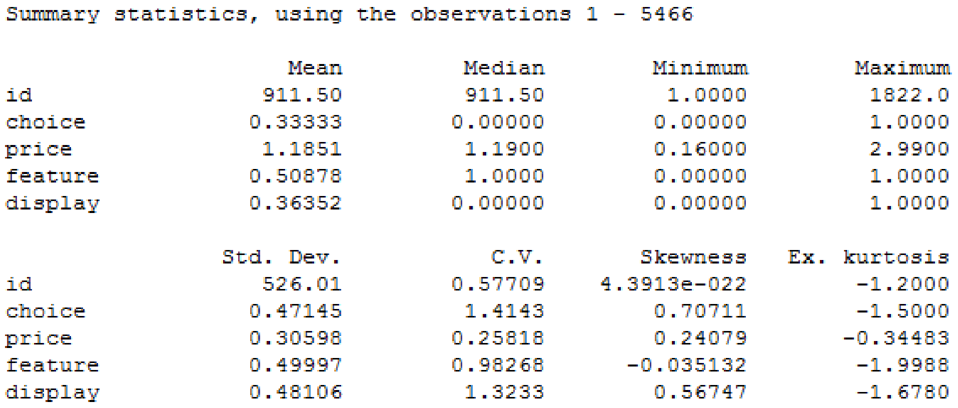


Figure B.1: Choosing summary statistics from the pull-down menu

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 distribution) extends over a relatively larger range of the variable than the right



```
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.

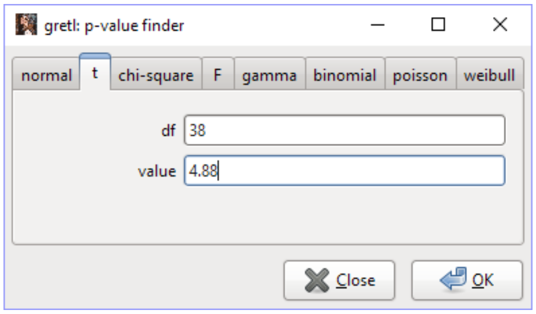
tail, the distribution is negatively skewed. If the right tail covers a larger range of values then it is positively skewed. Normal and $t$-distributions are symmetric and have zero skewness. The $\chi^2(n)$ is positively skewed. Excess kurtosis refers to the fourth moment about the mean of the distribution. '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.

| Statistic | Formula |
|---|---|
| Mean | $\sum x_i/n = \bar{x}$ |
| Variance | $\sum (x_i - \bar{x})^2/n = s_x^2$ |
| Standard Deviation | $s_x = \sqrt{s_x^2}$ |
| Coefficient of Variation | $s_x/\bar{x}$ |
| Skewness | $n^{-1}\sum\left((x_i - \bar{x})/s_x\right)^3$ |
| Excess Kurtosis | $n^{-1}\sum\left((x_i - \bar{x})/s_x\right)^4 - 3$ |

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

$$P[X \geq 4] = P[Z \geq (4-3)/\sqrt{9}] = P[Z \geq 0.334] \doteq 0.3694 \tag{B.2}$$

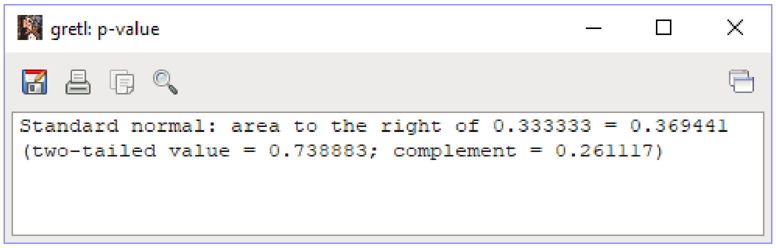
To obtain this probability, you can use the **Tools**>**P-value finder** from the pull-down menu. Then, give **gretl** the value of X, the mean of the distribution and its standard deviation using



df 38

value 4.88

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.

the dialog box shown in Figure B.3. The result appears in Figure B.4. Gretl is using the mean 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:

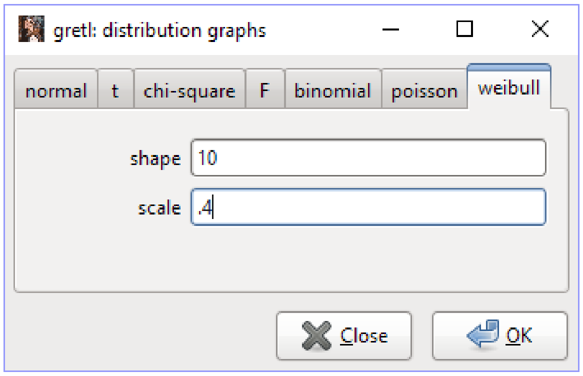
$$(1 - 0.1587) - (1 - 0.3694) = (0.3694 - 0.1587) = 0.2107 \tag{B.4}$$

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 $P[Z \le .334]$ was truncated to $P[Z \le .33]$; 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 $P[Z \le \frac{1}{3}]$. 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
```

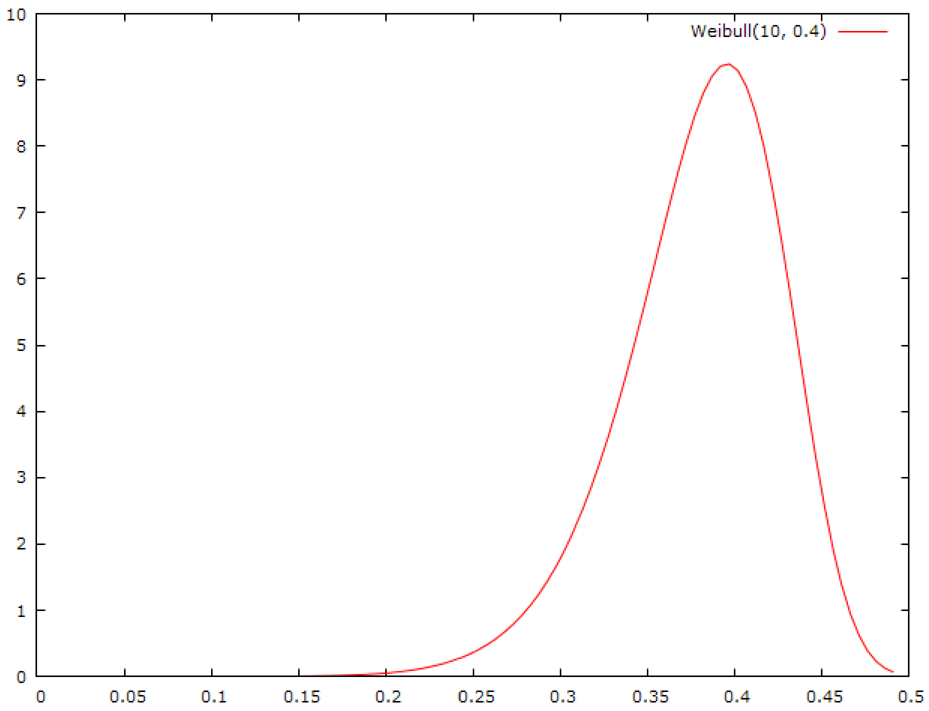
Gretl has a handy new feature that allows you to plot probability distributions. If you've ever wondered what a Weibull(10,0.4) looks like then this is the utility you have waited for. From the main menu choose **Tools>Distribution graphs** from the main menu. The following dialog will appear:



shape 10

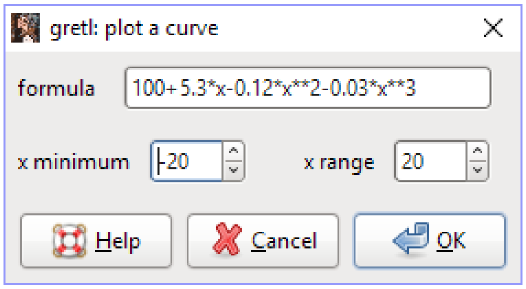
scale .4

which produces:



You can plot normal, $t$, $\chi^2$, $F$, 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.

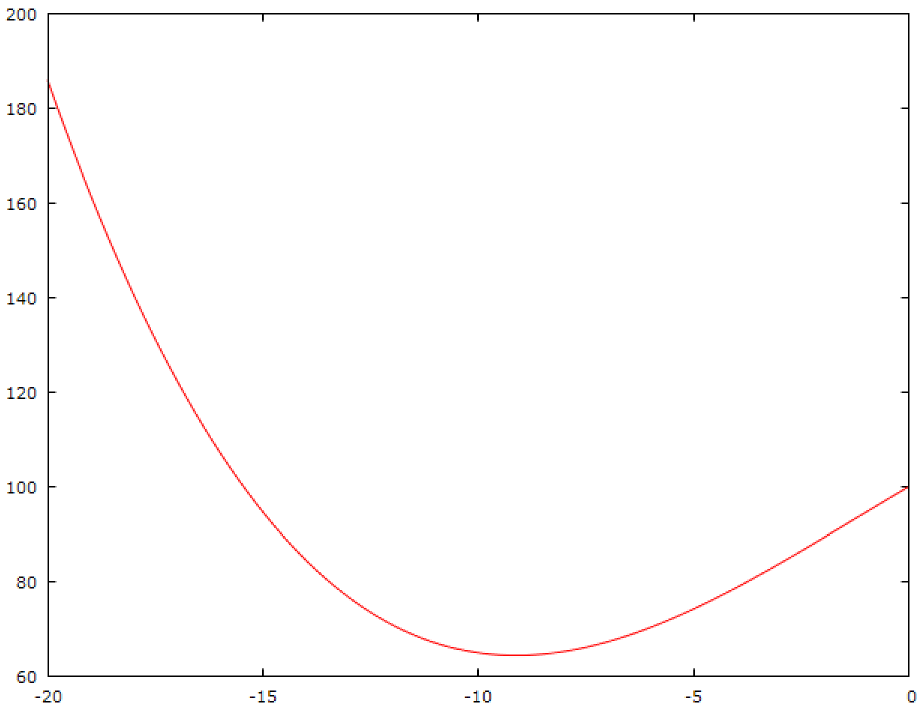


formula 100+5.3*x-0.12*x**2-0.03*x**3

x minimum -20

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 $\star\star$ for exponentiation (raising to a power). The curve plotted is:



### Example B.15

In this example, the inversion method is used to generate random numbers from a triangular distribution.

$$f(y) = 2y \quad 0 < y < 1$$

and hence $p = F(y) = y^2$. By inversion, $p = y^2$ and $y = \sqrt{p}$. The probability, p, is modeled using a pseudo-random uniform. In this example you could either generate a set of uniform variates or use the ones contained in the dataset *uniform1.gdt*.

```
1 open "@workdir\data\uniform1.gdt"
2 freq u1 --plot=display
3 series y1 = sqrt(u1)
4 freq y1 --plot=display
```

The uniforms are plotted in line 2 and the inversion in line 3.

The extreme value cdf is $F(\nu) = \exp(-\exp(-\nu))$ which inverted produces $\nu = -\ln(-\ln(u))$. The script is:

```
open "@workdir\data\uniform1.gdt"
series ev1 = -log(-log(u1))
freq ev1 --plot=display --nbins=20
```

### Example B.16

In this example, uniform random variables are generated using the linear congruential generator. This method uses the modulus (*mod*)

$$X_n = (aX_{n-1} + c) \quad \mathrm{mod}\ m$$

where $a$, $c$, and $m$ are constants chosen by the user. The constant, $m$, determines the maximum period of the recursively generated values. It is limited by the architecture of your 32-bit or 64-bit computing system. The beginning number of the sequence, $X_0$, is the seed.

Let $X_0 = 1234567$, $a = 1664525$ and $b = 1013904223$. The max periodicity of my machine is $2^{32}$. The uniforms are then generated:

```
nulldata 10001
series u1 = 1234567      # Seed 1
series u2 = 987654321    # Seed 2
scalar a = 1664525
scalar c = 1013904223
scalar m = 2^32
series u1 = (a*u1(-1)+c) - m*ceil((a*u1(-1)+c)/m) + m
series u1=u1/m

series u2 = (a*u2(-1)+c) - m*ceil((a*u2(-1)+c)/m) + m
series u2=u2/m

setinfo u1 -d "uniform random number using seed = 1234567"
setinfo u2 -d "uniform random number using seed = 987654321"

freq u1 --plot=display --nbins=20
freq u2 --plot=display --nbins=20
summary u1 u2
```

This produces Figures B.5 and B.6: The summary statistics indicate that the generator is pretty good. The mean is nearly .5 and the $95^{th}$ and $5^{th}$ percentiles are very close the their theoretical levels.

```
                           Mean         Median        Minimum        Maximum
u1                      0.49867        0.50084    3.2682e-005        0.99984
u2                      0.50087        0.50252    1.2647e-006        0.99980
```

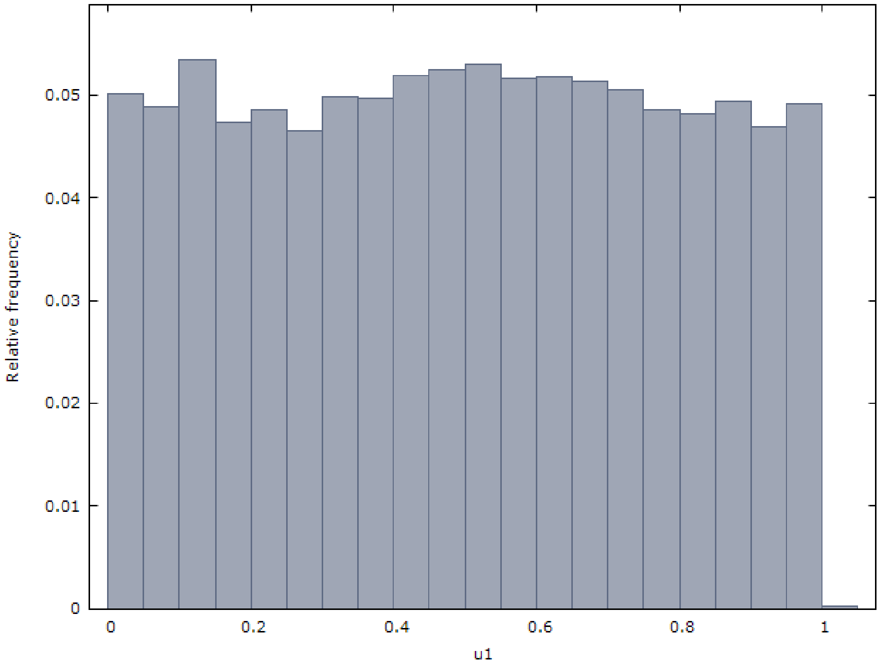


Figure B.5: Linear Congruential Generator: seed=1234567

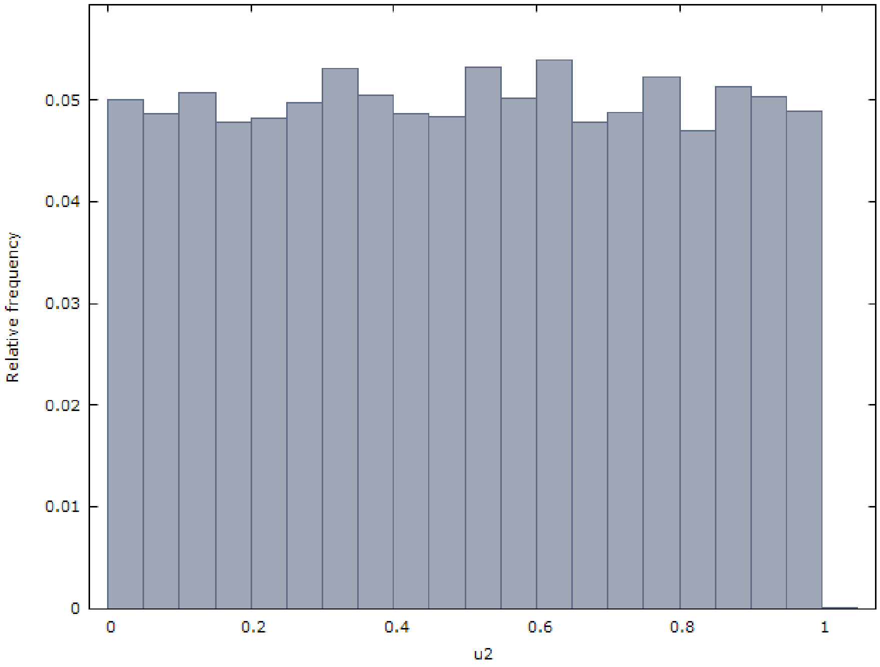


Figure B.6: Linear Congruential Generator: seed=987654321

```
                Std. Dev.           C.V.       Skewness   Ex. kurtosis
u1                0.28652        0.57456     -0.0039760        -1.1732
u2                0.28772        0.57445     -0.0056054        -1.1906

                 5% perc.      95% perc.       IQ range   Missing obs.
u1               0.049937        0.94909        0.49041              0
u2               0.049805        0.94847        0.49466              0
```

# Appendix C

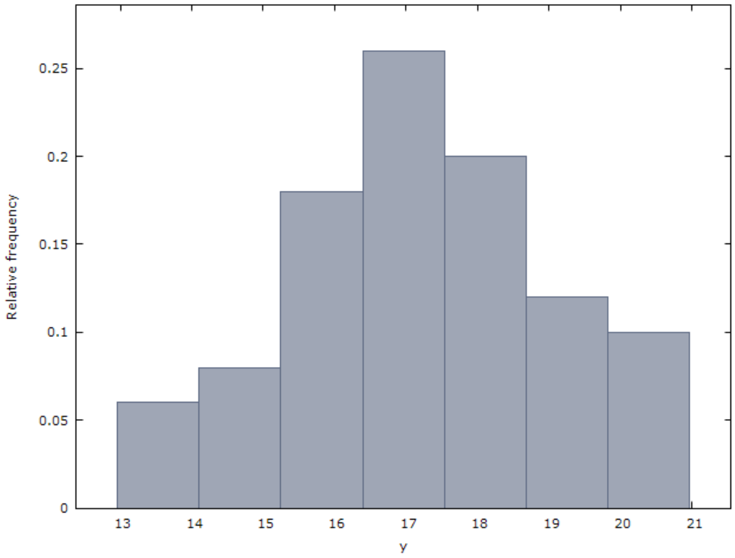
# Some Statistical Concepts

The hip data are used to illustrate computations for some simple statistics in your text.

## Example C.1 in *POE5*

In this example a simple histogram is plotted using the `freq` command. This command can be used to plot histograms of a series or of a vector. The plot can be sent to the display or to a file. In this example, the hip data are loaded and a simple frequency plot is graphed to the display:

```
open "@workdir\data\hip.gdt"
freq y --plot=display
```



## C.1 Summary Statistics

### Example C.2

In this example the sample mean of the variable y is computed using the summary command. summary computes summary statistics of the variables listed. It can also work on matrices if the --matrix= option is specified.

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, minimum, maximum, standard deviation, coefficient of variation, skewness and excess kurtosis of your variable(s).

```
                 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

### Example C.3 in *POE5*

In this example a set of 10 normal variates are created and placed into a matrix. Each sample has 50 observations. The mean of each is set to 17 and the standard deviation set to 1.8. Once created, simple summary statistics are created.

```
matrix Y = 17 + 1.8*mnormal(50,10)
summary --matrix=Y --simple
```

The mnormal command has two arguments. The first is the row dimension and the second the column dimension of the resulting matrix or set of series. The matrix Y in this case will be $50 \times 10$, and each column will have a mean of 17 and standard deviation of 1.8 (in the population).

The summary statistics are:

```
                 Mean      Median       S.D.        Min        Max
col1            16.88       16.99      2.087      11.15      20.30
col2            16.89       16.94      1.520      13.62      20.19
col3            17.06       17.14      1.604      13.61      20.58
col4            17.25       17.57      1.984      12.39      21.44
col5            16.88       16.94      1.578      12.97      20.16
col6            16.87       17.15      1.729      11.98      19.74
col7            17.34       17.36      2.011      13.01      21.41
col8            16.69       16.83      1.683      13.04      19.97
col9            17.33       17.39      1.470      14.14      20.78
col10           17.25       17.15      1.982      11.13      21.71
```

## C.2 Central Limit Theorem

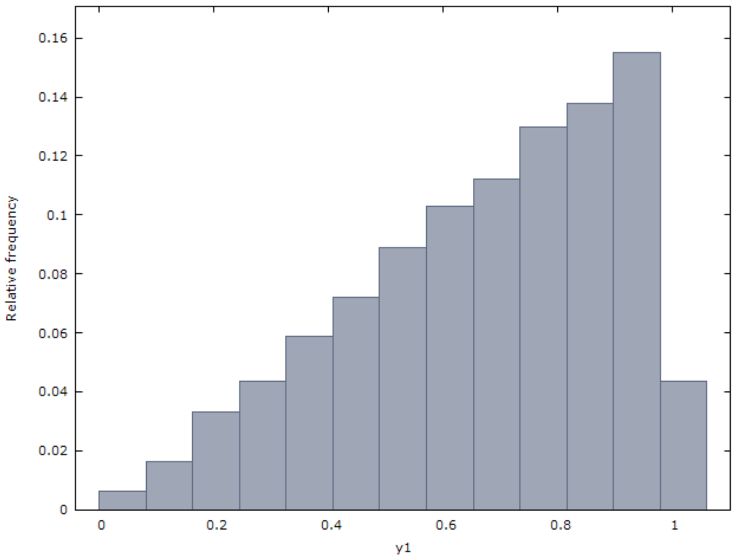
### Example C.5 in *POE5*

This example is based on a simple simulation where random variables are generated from a triangular distribution. Means of 10,000 samples are computed for various sample sizes (3, 10, and 30). Histograms of the means are compared to plots from a normal distribution. What we find is that even with a sample as small as 30, the behaviour of the mean is approaching normality.

First, 10000 observations from a triangular distribution are created (using the inverse method) and plotted.

```
nulldata 10000
set seed 123456
series y1 = sqrt(uniform(0,1))
freq y1 --plot=display --nbins=12
```

This produces the plot:



Obviously, these are not normally distributed.

Then, an empty, three observation dataset is created. A `--progressive` loop is initiated and the triangular series is generated and its mean computed. The mean is stored to a dataset *c5_3.gdt*. This is repeated for samples of size 10 and 30.

```
nulldata 3
loop 1000 --progressive --quiet
    series yt = sqrt(uniform())
    scalar ybar = mean(yt)
    store c5_3.gdt ybar
    endloop

nulldata 10
loop 1000 --progressive --quiet
    series yt = sqrt(uniform())
    scalar ybar = mean(yt)
    store c5_10.gdt ybar
endloop

nulldata 30
loop 1000 --progressive --quiet
    series yt = sqrt(uniform())
    scalar ybar = mean(yt)
    store c5_30.gdt ybar
endloop
```

Now there are three datasets *c5_3.gdt*, *c5_10.gdt*, and *c5_30.gdt*. Open each one and plot the frequency distribution of ybar as in:

```
open c5_3.gdt
freq ybar --plot=display --normal

open c5_10.gdt
freq ybar --plot=display --normal

open c5_30.gdt
freq ybar --plot=display --normal
```

This yields three graphs: Notice that the test statistic for normality shrinks as the sample size used to compute the mean of y increases. This is the essence of the central limit theorem (CLT).

## C.3 Sample Moments

### Example C.6 in *POE5*

In this example, several sample moments of the hip data are computed. The sample mean and standard deviation, the standard deviation of the mean, and the third and fourth moments of the hip data are computed.

Once the data are loaded, you can use **gretl**'s language to generate these. 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)ˆ2)/($nobs-1)`. The script below can be used to compute other summary statistics as discussed in your text.

```
open "@workdir\data\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, $K = \tilde{\mu}^4/\tilde{\sigma}^4 - 3$:

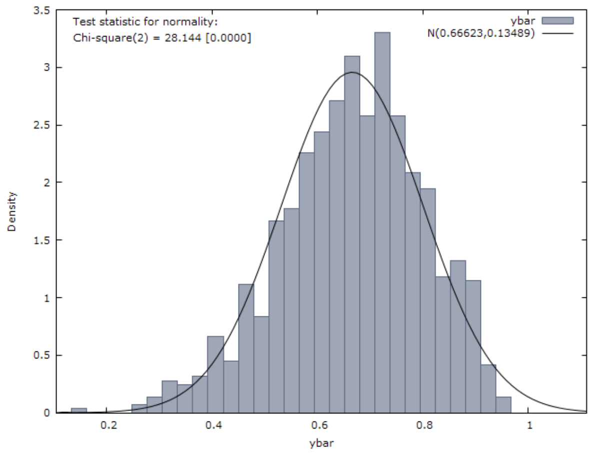


Figure C.1: Histogram of the triangular mean: N=3

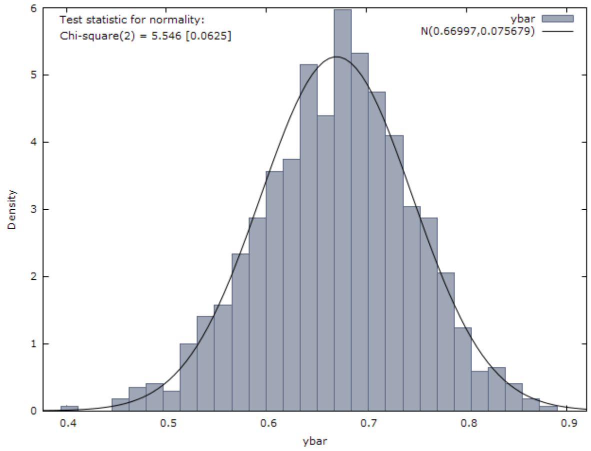


Figure C.2: Histogram of the triangular mean: N=10

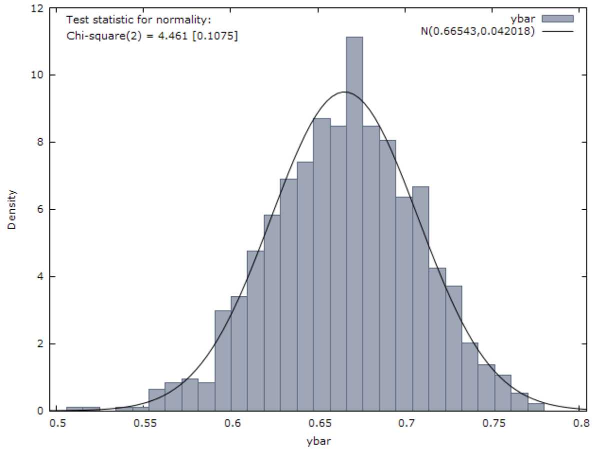


Figure C.3: Histogram of the triangular mean: N=30

```
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 = $\tilde{\mu}^4/\tilde{\sigma}^4 - 3$

### Example C.7 in *POE5*

If hip size in inches is normally distributed, $Y \sim N(\mu, \sigma^2)$. Based on the 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 $\mu$ and $\sigma$ by their estimates yields

```
scalar zs = (18 - mean(y))/sd(y)
pvalue z zs
```

The last line 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

$$P(Y \le y^*) = \frac{y^* - \bar{y}}{\hat{\sigma}} \le \frac{y^* - 17.1582}{1.8070} = 0.95 \tag{C.2}$$

To find the value of $Z = (y^* - \bar{y})/\hat{\sigma}$ that satisfies the probability one could use the `invcdf` function. 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.4 Interval Estimation

### Example C.8 in *POE5*

In this example pseudo-random numbers are used to generate 30 observations from a N(10,10) distribution. The data are found in *table_c3.gdt*. The sample mean is computed and a 95% confidence interval is computed for y.

$$\bar{y} \pm 1.96 \times \sqrt{10//30}$$

This is computed using:

```
open "@workdir\data\table_c3.gdt"
scalar ybar = mean(y)
scalar std = sqrt(10/$nobs)
scalar lb = ybar - critical(z,.025)*std
scalar ub = ybar + critical(z,.025)*std
printf "\nThe 95%% confidence interval is (%5.4f, %6.4f)\n",lb, ub

t_interval(ybar,std,10e10,.95)
```

to produce:

```
The 95% confidence interval is (9.0743, 11.3375)
```

In the last line, the `t_interval` program from Chapter 3.1 is used as well to produce a similar result. The `t_interval` command uses critical values from the *t*-distribution rather than the normal. The normal critical value can be approximated simply by using a very large number as the degrees-of-freedom parameter as done here.

```
The 95% confidence interval centered at 10.206 is (9.0743, 11.3375)
```

In the second part of this example 10 samples of N(10,10) variables are opened from the *table_4c.gdt* dataset. The script is:

```
open "@workdir\data\table_c4.gdt"
summary y* --simple

scalar std = sqrt(10/30)
scalar crit = critical(z,.025)
matrix result
list yall = y*
```

```
8  loop foreach i yall
9   scalar lb = mean($i) - crit*std
10  scalar ub = mean($i) + crit*std
11  matrix result = result | (lb ~ ub)
12 endloop
13 cnameset(result,"LowerBound UpperBound")
14 print result
```

After loading the data, summary statistics are summoned in line 2. Note that a wildcard is used for a variable list, `y*`. The variables are *y1*, *y2*, and so on and `y*` collects all series that begin with `y`.

The standard deviation of 30 observations from a N(10,10) are computed in line 4. The critical value from the normal is obtained in line 5. A matrix called `result` is initialized in line 6. This matrix will hold the lower and upper bounds of the computed intervals for each sample's mean.

A `foreach` loop is started in line 8 that will loop through each element of the list `yall`. The interval is computed and added as a row to the matrix results. The loop ends and column names are assigned to the matrix and printed.

The result:

```
LowerBound   UpperBound
    9.0743       11.338
    8.6963       10.959
    10.063       12.326
    7.6903       9.9535
    9.3025       11.566
    7.7231       9.9863
    9.3798       11.643
    8.0802       10.343
    9.3329       11.596
    9.0100       11.273
```

which matches the results from *POE5*.

### Example C.9 in *POE5*

In this continuation of the hip data simulation from Example C.8, the loop is modified to use the computed standard error of the mean from each sample and the $t$ critical value.

Since the true variance, $\sigma^2$, 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,$nobs-1,.025)` returns the 0.025 critical value from the $t_{49}$ distribution.

```
1  open "@workdir\data\table_c4.gdt"
2  
3  scalar crit = critical(t,$nobs-1,.025)
4  matrix result
5  list yall = y*
6  loop foreach i yall
7   scalar lb = mean($i) - crit*sd($i)/sqrt($nobs)
8   scalar ub = mean($i) + crit*sd($i)/sqrt($nobs)
9   matrix result = result | (lb ~ ub)
10 endloop
11 cnameset(result,"LowerBound UpperBound")
12 print result
```

The first change occurs in line 3 where the critical value is selected from the $t_{49}$ distribution rather than the normal. Then, the calculation of the lower and upper bounds in lines 7 and 8 uses the computed standard deviation of each series (`sd`(·)) and the number of observations stored in the accessor `$nobs`. Otherwise, it is the same and in the previous example.

The results are:

```
LowerBound   UpperBound
    9.0734       11.338
    8.8487       10.807
    9.9940       12.394
    7.6489       9.9948
    9.3789       11.489
    7.9227       9.7868
    9.5003       11.523
    7.7808       10.643
    9.2595       11.669
    8.5711       11.712
```

which matches the results from table C.5 in *POE5*.

### Example C.10 in *POE5*

Finally, these procedures are used on the original hip.gdt sample to obtain 95% confidence interval for the population mean.

```
open "@workdir\data\hip.gdt"
scalar y_sd = sd(y)
scalar ybar_sd = y_sd/sqrt($nobs)
scalar lb = mean(y) - critical(t,$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
```

which produces:

```
The 95% confidence interval is (16.6447, 17.6717)
```

## C.5 Hypothesis Tests

### Examples C.11 and C. 13 in *POE5*

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 "@workdir\data\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)
printf "\n The test statistic = %.3f\n\
One-sided critical value = %.3f\n\
Two-sided critical value = %.3f\n",\
   tstat, critical(t,df,.05), critical(t,df,0.025)

printf "\n The p-value for the one-sided test = %.4f\n",\
 pvalue(t,df,tstat)
```

This yields:

```
 The test statistic = 2.576
 One-sided critical value = 1.677
 Two-sided critical value = 2.010

 The p-value for the one-sided test = 0.0065
```

### Examples C.12 and C.14 in *POE5*

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 $\pm 2.01$. 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)
printf "\n The test statistic = %.3f\n\
Two-sided critical value = +/-%.3f\n",\
  abs(tstat), critical(t,df,.025)

printf "\n The p-value for the one-sided test = %.4f\n",\
  2*pvalue(t,df,abs(tstat))
```

which produces the results:

```
The test statistic = 0.619
Two-sided critical value = +/-2.010

The p-value for the one-sided test = 0.5387
```

## C.6 Testing for Normality

*POE5* 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 manually, one must obtain the summary statistics from your data series.

From **gretl** script

```
open "@workdir\data\hip.gdt"
summary
```

The summary statistics can be obtained from the user interface as well. From the main **gretl** window, highlight the hip series and right-click to bring up a list of options. Choose summary statistics. Or, highlight the desired series in the main window and choose `View>Summary statistics` 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.7 Maximum Likelihood

In this section some basic computations based on likelihood estimation are demonstrated. The premise is demonstrated using a game where there are two wheels that have shaded regions on their surface. Spinning the wheel and having it stop on the shaded area is considered a win. If the wheel stops on a non-shaded area, the contestant loses.

There are two different wheels in this game. Wheel A has a 25% chance of winning (P(A=win)=.25), wheel B has P(B=win)=.75. Someone spins three times and produces the following result: win, win, lose. Which wheel did she spin?

The likelihood is

$$L(p) = P_1 \times P_2 \times P_3 = p^2(1-p)$$

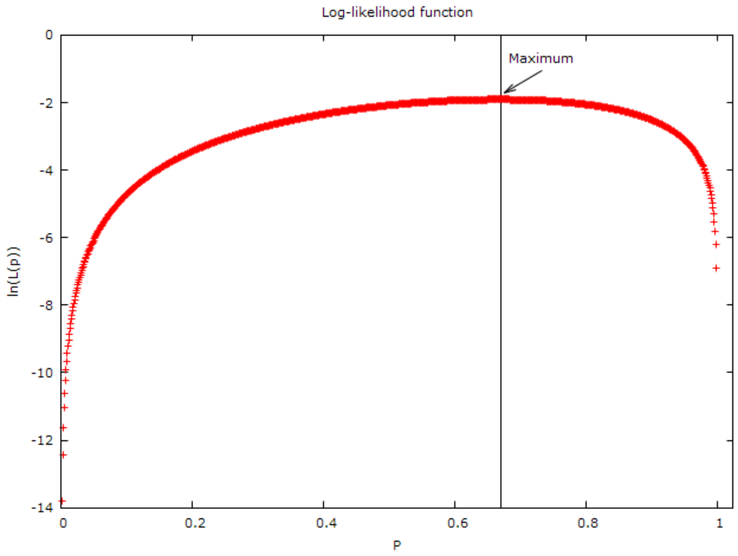
The maximum likelihood estimator is the value of p that maximizes this. For computational reasons, the likelihood is seldom used. Instead, users take its natural logarithm, which in this case is:

$$\ln L(p) = 2\ln(p) + \ln(1-p)$$

In this example a sequence of $p$ is created based on a sequence of numbers from .001 to 1 by .001. One way to do this is to create an empty dataset with 1000 observations. Then multiply the internal variable `index`, which by default in **gretl** identifies an observation's number by .001. Then formulate the log-likelihood series as in:

```
1 nulldata 1000
2
3 series p=.001 * index
4 series lnl=2*ln(p)+ln(1-p)
5 setinfo p -d "Probability" -n "P"
6 setinfo lnl -d "ln(L(p))" -n "ln(L(p))"
7 gnuplot  lnl p --fit=none --output=display
```

The setinfo commands add meaningful labels for the plot, which is carried out in line 7 by **gnuplot**. The result appears below:



The maximum occurs at 0.6667.

### Example 6.19 in *POE5*

A marketer wants to know whether potential customers prefer a blue box or a green one for their cereal. Two hundred are sampled at random and asked to express a preference. Seventy-five prefer the blue one. What is the estimated population proportion of those who prefer the blue?

Gretl has a new function called `eval` that is useful in this circumstance. It acts as a calculator in either a console window or in a script.

```
eval(75/200)
```

which yields:

```
? eval(75/200)
0.375
```

Our estimate of the population proportion is 37.5% prefer blue.

### Example C.20 in *POE5*

Following the previous example, suppose the CEO guesses that 40% prefer blue. To test this construct the null hypothesis $H_0$: $p = 0.4$ against the alternative $H_1$: $p \neq 0.4$.

As in the preceding examples, this random variable is binomial. The sample proportion is known to be

$$\hat{p} \overset{a}{\sim} N\left(p, \frac{p(1-p)}{N}\right) \tag{C.10}$$

The estimated variance substitutes $\hat{p}$ for $p$. The script to compute p and its estimated standard error is:

```
1 nulldata 200
2 scalar p = 75/200
3 scalar se_p = sqrt((p*(1-p))/$nobs)
4 scalar t = (p-0.4)/se_p
5 scalar pval = pvalue(t,$nobs-1,t)
6 scalar crit = critical(t,$nobs-1,.025)
7 print t pval crit
8
9 t_interval(p,se_p,$nobs-1,.95)
```

The empty dataset is created so that `$nobs` has the appropriate value. In line 3 the estimated standard error is computed and in 4 the $t$-ratio. The $p$-value and critical values are obtained and printed.

Finally, the `t_interval` function is used to produce a 95% confidence interval for $p$. The results are:

```
              t =  -0.73029674
           pval =   0.76696616
           crit =   1.9719565

The 95% confidence interval centered at 0.375 is (0.3075, 0.4425)
```

The $t$-ratio is $-0.73$, which is not in the rejection region of a 5% test (critical value is ±1.97). The $p$-value is .77, which is greater than 5%. The 95% confidence interval is $(0.3075, 0.4425)$, which of course does not include zero.

### C.7.1 Other Hypothesis Tests

### Example C.21 in *POE5*

The likelihood ratio test compares the log-likelihood functions evaluated at two-sets of estimates. One set is the values of the parameters under the null hypothesis (restricted) and the other is a set evaluated under the alternative hypothesis (unrestricted).

$$LR = 2(\ln(L(p_u) - \ln(L(p_r)) \sim \chi^2(J)$$

if the null hypothesis is true. The parameter, $J$, is the number of joint hypotheses specified in the null. The parameters, $p_u$ and $p_r$ are the unrestricted and restricted maximum likelihood estimates, respectively.

In this example, $p_u = 75/200 = 0.375$ and it is hypothesized to be equal to .4 under the null so $p_r = .4$. Evaluating the log-likelihood under these values and computing LR is accomplished using:

```
scalar p = 75/200
scalar lnL_u = 75*ln(p)+(200-75)*ln(1-p)
scalar lnL_r = 75*ln(0.4)+(200-75)*ln(1-0.4)

scalar LR = 2*(lnL_u-lnL_r)
scalar pval = pvalue(C,1,LR)
scalar crit = critical(C,1,.05)
print lnL_u lnL_r LR pval crit
```

which produces:

```
lnL_u = -132.31265
lnL_r = -132.57501
   LR =  0.52472046
 pval =  0.46883499
 crit =  3.8414588
```

The LR is equal to .5247 and has a p-value of .4688. It is not significant at 5% and the null hypothesis that 40% prefer blue cannot be rejected.

### Example C.22 in *POE5*

For this test, a Wald statistic is computed. The Wald statistic is based on quadratic forms of normally distributed random variables. Thus, if

$$X \sim N(\mu, \Sigma)$$

then

$$W = (x - \mu)^T \Sigma^{-1} (x - \mu) \sim \chi^2(k)$$

where $k$ is the dimension of $x$. Based on equation (C.10) we have

$$W = (\hat{p} - 0.4) \left( \frac{\hat{p}(1-\hat{p})}{N} \right)^{-1} (\hat{p} - 0.4) \sim \chi^2(1)$$

if $H_0$ is true.

The script to estimate this is:

```
matrix V = p*(1-p)/$nobs
matrix Wald = qform(p-0.4,inv(V))
scalar pval = pvalue(C,1,Wald)
scalar crit = critical(C,1,.05)
printf "The Wald statistic is: %.3g\n\

The 5%% critical value is:  %.3f\n\
and the p-value is: %.3f\n", Wald, crit, pval
```

The `qform` command is used to compute the quadratic form. The first argument is a vector to be tested. The second argument is the inverse of the variance. The output follows:

```
The Wald statistic is: 0.533
 The 5% critical value is:  3.841
 and the p-value is: 0.469
```

These results are very similar to those from the LR test.

### Example C.23 in *POE5*

The Lagrange multiplier version of the hypothesis test is considered here. The *LM* test considers the value of the score at the values hypothesised under the null (restricted). The score is the slope of the log-likelihood:

$$s(p) = \frac{\partial ln(L(p))}{\partial p}$$

The textitLM statistic is formulated as a quadratic form much like the Wald test. In this case,

$$LM = s(p)^T \Sigma(p) s(p) \sim \chi^2(1)$$

if $p = 0.4$.

The script:

```
scalar c = 0.4
scalar score = 75/c - (200-75)/(1-c)
scalar LM = qform(score,(c*(1-c))/$nobs)
scalar pval = pvalue(C,1,LM)
scalar crit = critical(C,1,.05)

printf "The LM statistic is: %.3g\n\
The 5%% critical value is:  %.3f\n\
and the p-value is: %.3f\n", LM, crit, pval
```

which produces:

```
The LM statistic is: 0.521
 The 5% critical value is:  3.841
 and the p-value is: 0.470
```

Again, this is very similar to LR and Wald tests. It should be. They are asymptotically equivalent.

### Example C.24 in *POE5*

In this example, the mean of the hip population is estimated by minimizing the sum of squared errors.

$$hip_i = \mu + e_i$$

This is a least squares problem, hence:

```
open "@workdir\data\hip.gdt"
summary y
ols y const
```

which produces:

```
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
```

and for the regression:

```
Model 1: OLS, using observations 1-50
Dependent variable: y

             coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const        17.1582      0.255550     67.14    6.74e-050 ***

Mean dependent var   17.15820   S.D. dependent var   1.807013
```

It should be clear that the least squares estimator of this model is simply the sample mean. Summary statistics and a regression using only a constant produce identical results.

## C.8 Kernel Density

Gretl includes a function that estimates the shape of a distribution based on a sample of observations. The approach in non-parametric since it does not rely on a specific functional form to generate an estimate. Instead, smoothing functions called kernels are used to "fit" the shape of the distribution using the data. The idea of a kernel was first use here in the discussion of HAC standard errors (see page 325). The function that computes these in **gretl** is called `kdensity`.

```
kdensity

Output:  matrix
Arguments: x  (series or vector)
scale  (scalar, optional)
control  (boolean, optional)
```

The function computes a kernel density estimate for a series or a vector. It returns a matrix having two columns, the first holding a set of evenly spaced abscissae and the second the estimated density at each of these points.

The optional scale parameter can be used to adjust the degree of smoothing relative to the default of 1.0 (higher values produce a smoother result). Is used to choose the specific kernel to use for smoothing. Set to zero to use a Gaussian kernel and something else to switch to the Epanechnikov kernel.

A plot of the results may be obtained using the gnuplot. Here are a couple of examples using the supplied dataset kernel.gdt.

```
open "@workdir\data\kernel.gdt"

matrix d = kdensity(y,1)
gnuplot 2 1 --matrix=d --with-lines --fit=none

matrix d = kdensity(y,3)
gnuplot 2 1 --matrix=d --with-lines --fit=none

matrix d = kdensity(y,.2)
gnuplot 2 1 --matrix=d --with-lines --fit=none
```

The plots are shown in Figure C.4,C.5, and C.6 below. Notice that the larger the bandwidth, the more smoothing occurs. Note, the bandwidths are not literally set to 1, 2.5 and .2. Gretl computes an automatic bandwidth and these numbers are scales for that automatic selection.

## C.9 Script

```
open "@workdir\data\hip.gdt"
# Example C.1
freq y --plot=display

# Example C.2
summary y --simple

# Example C.3
matrix Y = 17 + 1.8*mnormal(50,10)
summary --matrix=Y --simple

# Example C.5
nulldata 10000
set seed 123456
series y1 = sqrt(uniform(0,1))
freq y1 --plot=display --nbins=12

nulldata 3
loop 1000 --progressive --quiet
    series yt = sqrt(uniform())
    scalar ybar = mean(yt)
    store c5_3.gdt ybar
    endloop

nulldata 10
loop 1000 --progressive --quiet
    series yt = sqrt(uniform())
    scalar ybar = mean(yt)
    store c5_10.gdt ybar
```

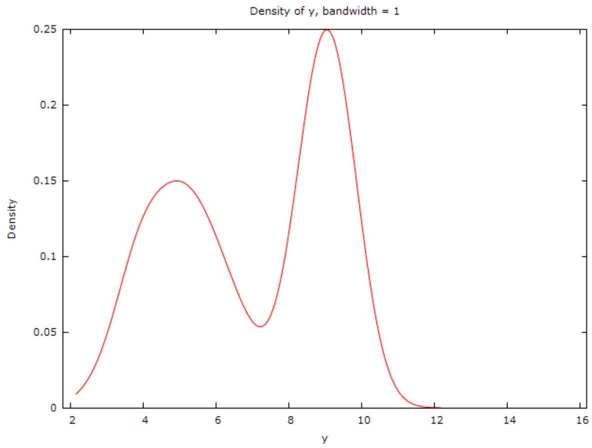


Figure C.4: Kernel density: Bandwidth scaler = 1

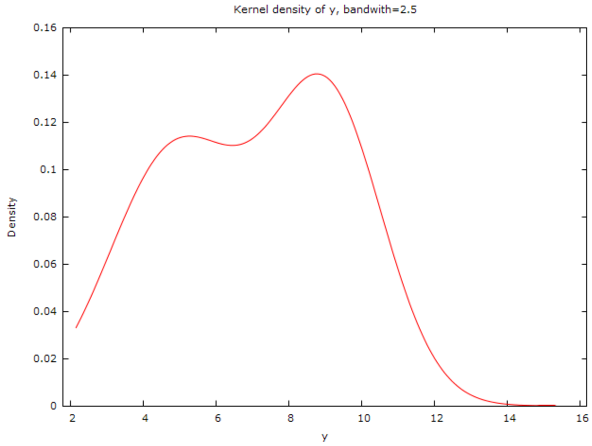


Figure C.5: Kernel density: Bandwidth scale = 2.5

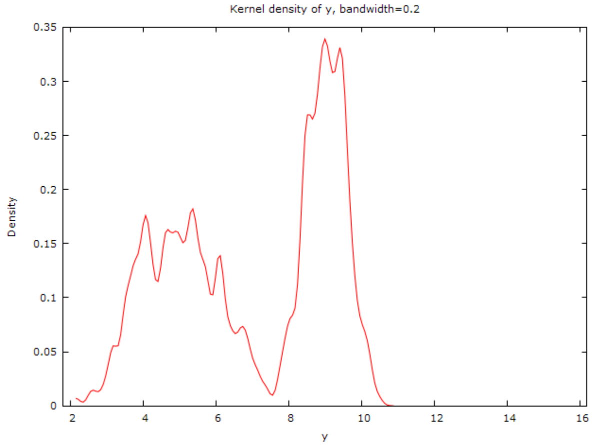


Figure C.6: Kernel density: Bandwidth scale = 0.2

```
endloop

nulldata 30
loop 1000 --progressive --quiet
    series yt = sqrt(uniform())
    scalar ybar = mean(yt)
    store c5_30.gdt ybar
endloop

open c5_3.gdt
freq ybar --plot=display --normal

open c5_10.gdt
freq ybar --plot=display --normal

open c5_30.gdt
freq ybar --plot=display --normal

# Example C.6
open "@workdir\data\hip.gdt"
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

# Example C.7
# 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

# Example C.8
open "@workdir\data\table_c3.gdt"
scalar ybar = mean(y)
scalar std = sqrt(10/$nobs)
scalar lb = ybar - critical(z,.025)*std
scalar ub = ybar + critical(z,.025)*std
printf "\nThe 95%% confidence interval is (%5.4f, %6.4f)\n",\
     lb, ub

t_interval(ybar,std,10e10,.95)

clear
open "@workdir\data\table_c4.gdt"
summary y* --simple

scalar std = sqrt(10/30)
scalar crit = critical(z,.025)
matrix result
list yall = y*
loop foreach i yall
 scalar lb = mean($i) - crit*std
 scalar ub = mean($i) + crit*std
 matrix result = result | (lb ~ ub)
endloop
cnameset(result,"LowerBound UpperBound")
print result

# Example C.9
clear
open "@workdir\data\table_c4.gdt"
summary y* --simple

scalar crit = critical(t,29,.025)
matrix result
list yall = y*
loop foreach i yall
 scalar lb = mean($i) - crit*sd($i)/sqrt($nobs)
 scalar ub = mean($i) + crit*sd($i)/sqrt($nobs)
 matrix result = result | (lb ~ ub)
endloop
cnameset(result,"LowerBound UpperBound")
print result

# Example C.10
 # Confidence interval
open "@workdir\data\hip.gdt"
scalar y_sd = sd(y)
scalar ybar_sd = y_sd/sqrt($nobs)
scalar lb = mean(y) - critical(t,$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

# Example C.11 and C.13
# t-test
open "@workdir\data\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)
printf "\n The test statistic = %.3f\n\
One-sided critical value = %.3f\n\
Two-sided critical value = %.3f\n",\
   tstat, critical(t,df,.05), critical(t,df,0.025)

printf "\n The p-value for the one-sided test = %.4f\n",\
 pvalue(t,df,tstat)

# Example C.12 and C.14
scalar tstat = (y_bar-17)/(ybar_sd)
scalar c = critical(t,df,0.025)
printf "\n The test statistic = %.3f\n\
Two-sided critical value = +/-%.3f\n",\
  abs(tstat), critical(t,df,.025)

printf "\n The p-value for the one-sided test = %.4f\n",\
  2*pvalue(t,df,abs(tstat))

# Example C.15
# 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

/*---POE5 Example C.18---*/
# The "Wheel of Fortune" Game: Maximizing the Log-likelihood

nulldata 1000

series p=.001 * index
series lnl=2*ln(p)+ln(1-p)
setinfo p -d "Probability" -n "P"
setinfo lnl -d "ln(L(p))" -n "ln(L(p))"
gnuplot  lnl p --fit=none --output=display

# Example C.19
eval(75/200)

/*---POE5 Example C.20---*/
# Testing a Population Proportion
nulldata 200
scalar p = 75/200
scalar se_p = sqrt((p*(1-p))/$nobs)
scalar t = (p-0.4)/se_p
scalar pval = pvalue(t,$nobs-1,t)
scalar crit = critical(t,$nobs-1,.025)
print t pval crit

t_interval(p,se_p,$nobs-1,.95)

/*---POE5 Example C.21---*/
# Likelihood Ratio Test of the Population Proportion

scalar p = 75/200
scalar lnL_u = 75*ln(p)+(200-75)*ln(1-p)
scalar lnL_r = 75*ln(0.4)+(200-75)*ln(1-0.4)

scalar LR = 2*(lnL_u-lnL_r)
scalar pval = pvalue(C,1,LR)
scalar crit = critical(C,1,.05)
print lnL_u lnL_r LR pval crit

/*---POE5 Example C.22---*/
# Wald Test of the Population Proportion

matrix V = p*(1-p)/$nobs
matrix Wald = qform(p-0.4,inv(V))
scalar pval = pvalue(C,1,Wald)
scalar crit = critical(C,1,.05)
printf "The Wald statistic is: %.3g\n\
The 5%% critical value is:  %.3f\n\
and the p-value is: %.3f\n", Wald, crit, pval

/*---POE5 Example C.23---*/
# Lagrange Multiplier Test of the Population Proportion

scalar c = 0.4
scalar score = 75/c - (200-75)/(1-c)
scalar LM = qform(score,(c*(1-c))/$nobs)
scalar pval = pvalue(C,1,LM)
scalar crit = critical(C,1,.05)
printf "The LM statistic is: %.3g\n\
The 5%% critical value is:  %.3f\n\
and the p-value is: %.3f\n", LM, crit, pval

/*---POE5 Example C.24---*/
# Hip Data: Minimizing the Sum of Squares Function
open "@workdir\data\hip.gdt"
summary y
ols y const

# Appendix C.10
open "@workdir\data\kernel.gdt"
scalar bw = $nobs^(-.2)
```

```
matrix d = kdensity(y,1,0)
cnameset(d,"y Density")
gnuplot 2 1 --matrix=d --with-lines --fit=none --output=display

matrix d = kdensity(y,2.5)
cnameset(d,"y Density")
gnuplot 2 1 --matrix=d --with-lines --fit=none --output=display

matrix d = kdensity(y,.2)
cnameset(d,"y Density")
gnuplot 2 1 --matrix=d --with-lines --fit=none --output=display

```

# Appendix D

# Functions

The functions used in this work are found in two files. The first includes all of the functions, except the ones used in Chapter 16, which are more specialized.

## D.1 functions_other

```
set echo off
set messages off
# function computes prediction standard errors
    function series in_sample_fcast_error(series y, list xvars)
    ols y xvars
    scalar sig = $sigma^2
    matrix X = { xvars }
    matrix f_e = sig*I($nobs)+sig*X*inv(X'X)*X'
    series se = sqrt(diag(f_e))
    return se
end function

# function estimates confidence intervals based on the t-distribution
function void t_interval(scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# function to compute diagonals of hat matrix
function series h_t (list xvars)
    matrix X = { xvars }
    matrix Px = X*inv(X’X)*X’
    matrix h_t = diag(Px)
    series hats = h_t
    return hats
end function

# delete-one variance function
function series delete_1_variance(series y, list xvars)
    matrix sig = zeros($nobs,1)
    loop i=1..$nobs --quiet
        matrix e_t = zeros($nobs,1)
        matrix e_t[i,1]=1
        series et = e_t
        ols y xvars et --quiet
        matrix sig[i,1]=$sigma^2
    endloop
    series sig_t = sig
    return sig_t
end function

# 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, $rsq, rbar2, aic, bic}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "k = %d, n = %d, R2 = %.4f, Adjusted R2 = %.4f, AIC = %.4f,\
and SC = %.4f\n", k, N, $rsq, rbar2, aic, bic
    return A
end function

# Function to compute RMSE for t1, t2
function matrix rmse (series yvar, list xvars, scalar t1, scalar t2)
    matrix y = yvar                  # Put yvar into matrix
    matrix X_all = { xvars }         # Put xvars into matrix
    matrix y1 = y[1:t1,]             # Estimation subset y
    matrix X = X_all[1:t2,]          # Sample restricted to 1-t2
    matrix X1 = X_all[1:t1,]         # Estimation subset regressors
    matrix Px1 = X*inv(X1’X1)*X1’y1  # Yhat for entire 1:t2 sample
    matrix ehat = y[1:t2,]-Px1       # Y-Yhat for entire 1:t2 sample
    matrix ehatp = ehat[t1+1:t2,]    # Residuals for the prediction sub-period
    matrix RMSE = sqrt(ehatp’ehatp/(t2-t1))# Mean of squared prediction residuals
    return RMSE
end function

# Breusch-Pagan test
```

```
function void BP_test (series y, list xvars, list zvars)
    ols y xvars --quiet
    series ehat_2 = $uhat^2
    ols ehat_2 zvars --quiet
    scalar pval = pvalue(X,nelem(zvars)-1,$trsq)
    printf "Z-Variables: %s", varname(zvars)
    printf "\nBreusch-Pagan test: nR2 = %.3f\
    p-value = %.3f \n", $trsq, pval
end function

# Example 9.8 Choosing lag lengths, SC criterion
# 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, $rsq, rbar2, aic, bic}
    printf "\nRegressors: %s\n",varname(xvars)
    printf "k = %d, n = %d, R2 = %.4f, Adjusted R2 = %.4f, AIC = %.4f,\
and SC = %.4f\n", k, N, $rsq, rbar2, aic, bic
    return A
end function

# Same as modelsel except the print statements are supressed
function matrix modelsel_np (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, $rsq, rbar2, aic, bic}
    return A
end function

# Function returns a 1 if reject Hausman null
function scalar Hausman (series y, list xvars, list zvars)
    list endogvars = xvars - zvars
    ols endogvars zvars --quiet
    series vhat = $uhat
    ols y xvars vhat --quiet
    scalar t = $coeff(vhat)/$stderr(vhat)
    scalar reject = abs(t)>1.96
    return reject
end function

```

```
# Example 10.8
# canonical correlations in gretl--Weak IV example 3
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

# Optional Fuller Modified LIML a=1
function void LIML (series depvar "dependent variable",
      list xvars "regressor list",
      list zvars "instrument list",
      string a   "Fuller or No Fuller")
    list endogvars = xvars - zvars
    list yvars = depvar endogvars
    matrix Y = { yvars }                    # All Endogenous vars, y and Y
    matrix y = { depvar }
    matrix w = { zvars }                    # w=All instruments
    matrix z = { xvars - endogvars }        # z=Internal instruments only
    matrix X = { xvars }

    matrix Mz = I($nobs)-z*invpd(z'*z)*z'   # Projection off of Z
    matrix Mw = I($nobs)-w*invpd(w'*w)*w'   # Projection off of w
    matrix Ez = Mz*Y                        # Residuals
    matrix Ew = Mw*Y                        # Residuals
    matrix W0 = Ez'*Ez                      # SSE
    matrix W1 = Ew'*Ew                      # SSE
    matrix G = inv(W1)*W0
    matrix l = eigengen(G, null)

    if a == "Fuller"
        scalar k=min(l)-(1/($nobs-nelem(xvars)))
    else
        scalar k=min(l)
    endif

    matrix kM = (I($nobs)-(k*Mw))
    matrix b =invpd(X'*kM*X)*X'*kM*y
    matrix sig2=(y-X*b)'*(y-X*b)/($nobs-nelem(xvars))
    matrix covmat = sig2*invpd(X'*kM*X)
    matrix se = sqrt(diag(covmat))
    matrix results = b~se~b./se

    cnameset(results, "Coeff Std_Error t-ratio")
    rnameset(results, "mtr educ kidsl6 nwifeinc const ")
    printf "\nThe LIML estimates using %s adjustment with k=%3f \n %12.3f\n", a, k, re
end function

function matrix gim_filter(series y, \
      scalar mu, scalar theta, scalar delta, scalar alpha, \
      scalar gam, scalar beta, series *h)

    series lh = var(y)               # initialize the variance series
    series le = y - mu               # initialize the residual series
    scalar T = $nobs                 # Number of Observations
    loop i=2..T --quiet
        scalar ilag = $i - 1
        scalar d = (le[ilag]<0)      # Create the negative threshold
        scalar e2lag = le[ilag]^2    # Square the residual
        lh[i] = delta + alpha*e2lag + gam*e2lag*d + beta*lh[ilag] # ht
        le[i] = le[i] - theta*lh[i]  # residual
    endloop

    series h = lh                    # Puts ht into series h (pointer in function)
    matrix matvar = { le, h}         # The  matrix return
    return matvar
end function
```

## D.2 functions_ch16

```
set echo off
# This function computes a t-dist confidence interval based on a statistic
function void t_interval (scalar b, scalar se, scalar df, scalar p)
    scalar alpha = (1-p)
    scalar lb = b - critical(t,df,alpha/2)*se
    scalar ub = b + critical(t,df,alpha/2)*se
    printf "\nThe %2g%% confidence interval centered at %.3f is\
(%.4f, %.4f)\n", p*100, b, lb, ub
end function

# This function computes t-dist confidence intervals after a model
function matrix t_interval_m (matrix b "Coefficients",
      matrix v "Variance-covariance matrix",
      int df "Degrees-of-freedom",
      scalar p "Coverage probability for CI")

    scalar alpha = (1-p)               # Convert p to alpha
    matrix c = critical(t,df,alpha/2)  # alpha/2 critical value
    matrix se = sqrt(diag(v))          # standard errors
    matrix lb = b - c*se               # lower bound
    matrix ub = b + c* se              # upper bound
    matrix result = b ~ se ~ lb ~ ub   # put into matrix

    cnameset(result, "Estimate StdErr (Lower, Upper) ")
    rnameset(result, "b")
    printf "\nThe %2g%% confidence intervals\
(t-distribution)\n%10.4f\n", p*100, result
    return result
end function

function matrix ame_binary(matrix *b "parameter estimates",
      list x "Variables list",
      int dist[1:2:2] "distribution" )
# Computes average marginal effects for probit or logit
    matrix p = lincomb(x, b)       # The index function
    matrix d = (dist==1) ? exp(-p)./(1.+exp(-p)).^2 : dnorm(p)
    matrix ame_matrix = d*b’
    cnameset(ame_matrix, x)            # add column names
    matrix amfx = meanc(ame_matrix)    # find the means
    cnameset(amfx, x)                  # add the column names to amfx
    printf "\n Average Marginal Effects (AME):\
      \n Variables: %s\n%12.4g \n", varname(x), amfx
    return amfx
end function

function matrix ame_cov (matrix b "parameter estimates",
      matrix covmat "Covariance",
      list x "Variables list",
      int dist[1:2:2] "distribution" )
    # Computes std errs for AME probit/logit
    # Requires ame_binary
    matrix amfx = ame_binary(&b, x, dist)
    matrix jac = fdjac(b, ame_binary(&b, x , dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    matrix results = amfx’ ~ se
    rnameset(results, "b")
    cnameset(results, "AME StdErr")
    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    printf "%10.4f\n", results
    return amfx|variance
end function

function scalar p_binary(matrix b "parameter estimates",
      matrix x "Representative Point",
      int dist[1:2:2] "distribution" )
    # Computes the probability of a binary choice: 1 = logit
    scalar p = x*b                   # The index function
    scalar d = (dist==1) ? 1./(1.+exp(-p)) : cnorm(p)
    return d
end function

function void Probs (matrix b "parameter estimates",
      matrix covmat "Covariance",
      matrix x "Representative Point",
      scalar df "Degrees of Freedom",
      int dist[1:2:2] "distribution")
    # Function computes std errors of binary predictions
    # Requires p_binary
    scalar p = p_binary(b, x, dist)
    matrix jac = fdjac(b, p_binary(b, x , dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif

    printf "95%% t(%.2g) confidence interval for probability at\n\
    x = %8.4f\n", df, x
    cnameset(results, " Lower ME Upper StdError" )
    printf "%10.4f\n", results
end function

function scalar me_at(matrix *param "parameter estimates",
      matrix xx "Representative Point",
      scalar q "Parameter of interest",
      int modl[1:2:2] "distribution" )
    # Marginal effects at a point -- continuous variables only
    scalar idx = xx*param
    scalar d = (modl==1)? (exp(-idx)./(1.+exp(-idx)).^2)*param[q] :\
      dnorm(idx)*param[q]
    return d
end function

function void MER (matrix *b "parameter estimates",
      matrix covmat "Covariance",
      matrix x "Representative Point",
      int q "Parameter of interest",
      int df "Degrees of Freedom",
      int modl[1:2:2] "distribution")
    # Std errors for Marginal effects at a point -- continuous vars only
    scalar p = me_at(&b, x, q, modl)
    matrix jac = fdjac(b, me_at(&b, x , q, modl))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se
    if modl == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    printf "95%% t(%.2g) confidence interval for b%.g at\n x =\
    %9.2g \n", df, q, x
    cnameset(results, " Lower ME Upper StdError" )
    printf "%10.4f\n", results
end function

function void MER_lpmfx (matrix b "parameter estimates",
      list  XL "list of regressors",
      matrix covmat "Covariance matrix",
      matrix x_at "Representative point",
      int dist[1:2:1] "distribution",
      int df "degrees-of-freedom")
    # The MER function to be used with lp-mfx.gfn
    # available from gretl's function server
    matrix me = binary_dp_dx(b, XL, x_at, dist)
    matrix jac = fdjac(b, binary_dp_dx(b, XL, x_at, dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    matrix results = me' ~ se
    if dist == 1
        printf "Logit:\n"
    else
        printf "Probit:\n"
    endif
    scalar crit = critical(t,df,0.025)
    matrix results = (me'-crit*se) ~ me' ~ (me'+crit*se) ~ se
    cnameset(results, "Lower ME Upper StdErr")
    rnameset(results, XL[2:nelem(XL)])
    cnameset(x_at, XL )
    printf "Representative Point\n%11.2g\n95%% CI for MER\n%10.4g\n",x_at, results
end function


# Poisson ME at point -- continuous variable
function scalar p_me_at(matrix b, matrix xx, scalar q)
    scalar me = exp(xx*b)*b[q]
    return me
end function

# Poisson ME at point -- indicator variable
function scalar p_me_at_d(matrix b, matrix x1, matrix x2)
    scalar me = exp(x1*b)-exp(x2*b)
    return me
end function
```

```

function list mlogitprob(series y "Dependent variable",
      list x "List of regressors",
      matrix theta "Coefficient vector")
    # computes probabilites of each choice for all data
    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 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 "Dependent variable",
      matrix x "Representative point 1xk",
      matrix theta "Coefficient vector")
    # computes probabilites of each choice at a representative point
    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 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 series mlogitlogprobs(series y "Dependent Variable",
      matrix X "Independent variables",
      matrix theta "Parameters")
    # This function computes the log probabilites for MLE
    # estimation of MNL
    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 i=1..n --quiet
        series x = tmp[,i]
        ret += (y==$i) ? x : 0
    endloop
    return ret
end function

function matrix mnl_se_lpfmx (matrix b "parameter estimates",
      matrix covmat "Covariance of MNL",
      list XL "list of regressors",
      matrix x "vector of x-values",
      int j "1-based index of outcome",
      int m "number of possible outcomes",
      int df "degrees of freedom for CI" )
# Computes MER and std errors for MNL
# must install and use lp-mfx.gfn
    matrix p = mlogit_dpj_dx(b, XL, x, j, m)
    matrix jac = fdjac(b, mlogit_dpj_dx(b, XL, x, j, m))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

    cnameset(results, "Lower ME Upper StdErr")
    printf "95%% CI for MER\n%10.4f\n", results
    return results
end function

# Several Functions for conditional logit.
# These are NOT general
# clprobs       --Conditional logit probability scalar
# clprobs_mat   --Conditional logit probabilities matrix
# clprobs_at    --marginal effects at a point -> 1x3 vector
# cl_me         --marginal effects continuous w/std errors
# cl_me_d       --marginal effects discrete   w/std errors

function scalar clprobs(list y "list of choices",
      list x "list of independent variables",
      matrix theta "parameters")
    # computes the probabilities for Conditional Logit
    # Used in user written MLE
    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
    endloop
    return sumc(ln(P))
end function

function matrix clprobs_mat(list x, matrix theta)
    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 = i1/d ~ i2/d ~ i3/d
        matrix P = P | pp
    endloop
    return P
end function

function matrix clprobs_at(matrix x, matrix theta)
    scalar i1 = exp(theta[1]+theta[3]*x[1])
    scalar i2 = exp(theta[2]+theta[3]*x[2])
    scalar i3 = exp(theta[3]*x[3])
    scalar  d = i1+i2+i3
    matrix pp = i1/d ~ i2/d ~ i3/d
    return pp
end function

function scalar cl_me(matrix *x "vector for the desired point",
      matrix *theta "parameters",
      int q "variable index for own price",
      int p "variable index for other price")
    # Margial effects for CL model -- continuous case
    # Function only works for 3 choice beverage model in poe
    # Inputs: x = point at which to evaluate
    # theta: Cond Logit MLE
    # q: own price index
    # p: other price index
    # op: 1 if own price, 0 otherwise
    matrix mm = clprobs_at(x, theta)
    if p == q
        scalar me = mm[q]*(1-mm[q])*theta[3]  # own price pepsi
    else
        scalar me = -mm[p]*mm[q]*theta[3]      # cross price 7up
    endif
    return me
end function

function matrix cl_me_d(matrix *x1,
      matrix *x2,
      matrix *theta)
    # Margial effects for CL model -- discrete case
    matrix mm = clprobs_at(x1, theta)
    matrix m2 = clprobs_at(x2, theta)
    mat = m2-mm
    return mat
end function

function matrix op_se_lpfmx (matrix b "parameter estimates",
      matrix covmat "Covariance of MNL",
      list XL "list of regressors",
      matrix x "vector of x-values",
      int j "1-based index of outcome",
      int m "number of possible outcomes",
      int df "degrees of freedom for CI",
      int dist[1:2:1] "distribution" )
    # Computes marginal effects and std errors for ordered probit/logit
    # must install and use lp-mfx.gfn
    matrix p = ordered_dpj_dx(b, XL, x, j, m, dist)
    matrix jac = fdjac(b, ordered_dpj_dx(b, XL, x, j, m, dist))
    matrix variance = qform(jac,covmat)
    matrix se = sqrt(diag(variance))
    scalar crit = critical(t,df,0.025)
    matrix results = (p-crit*se) ~ p ~ (p+crit*se) ~ se

    cnameset(results, "Lower ME Upper StdErr")
    printf "95%% CI for MER\n%10.4f\n", results
    return results
end function
```

# Appendix E

# 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**.

Constantin Colonescu had written a guide to using R for *POE5*. It is available in paperback on Amazon.com.

## E.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**. For a more comprehensive guide, see (Cottrell and Lucchetti, 2018, Chapter 39).

### E.1.1 Using the `foreign` command

A `foreign` block can be used to 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**.

### E.1.2 Opening an R session

To illustrate, open the *cola.gdt* data in **gretl**.

```
open "@workdir\data\cola.gdt"
```

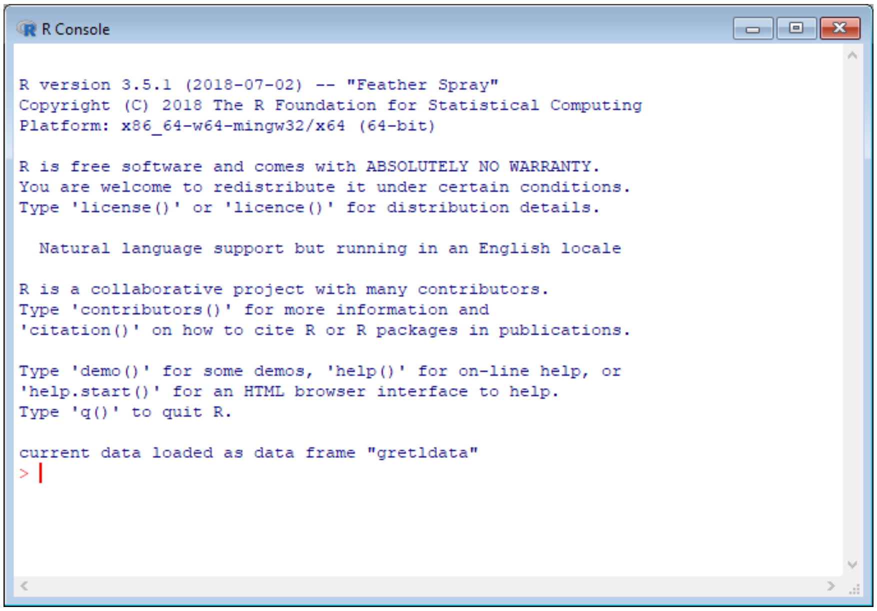
Now, select **Tools>start GNU R** from the pull-down menu. The current **gretl** data set, in this case *cola.gdt*, will be transported into **R**'s required format. You'll see the **R** console which is shown in Figure E.1. The message in **R** tells you that the data are loaded into an **R data frame** 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**.

### E.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>New script>R script** menu entry. By selecting **User file**, one can load a pre-existing **R** script.

Figure E.2

[1] This is taken almost directly from the **gretl** Users Guide, chapter 30

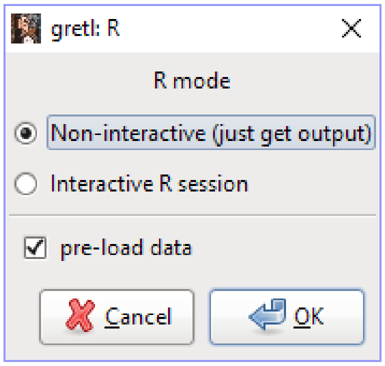


```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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"
> 
```

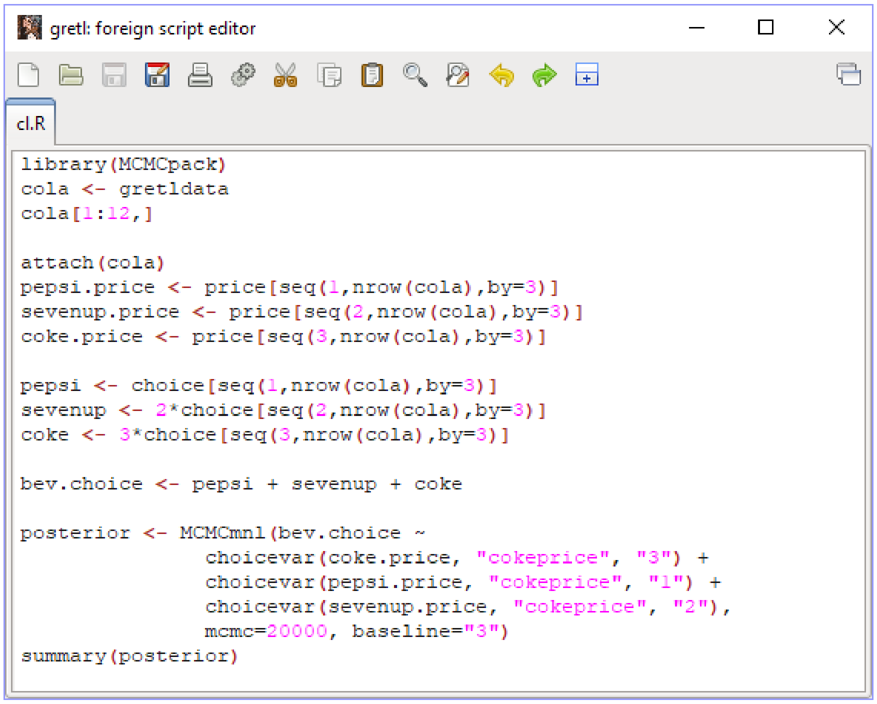
Figure E.1: The **R** console when called from **gretl**. Choose `Tools>Start GNU R` from the main **gretl** window.

In either case, you are presented with a window very similar to the editor window used for ordinary **gretl** scripts, as in Figure E.2.

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:



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.



```
library(MCMCpack)
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)
```

Figure E.2: Using **R** from the **R** script editor in **gretl**.

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**.

## E.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/leead/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**.

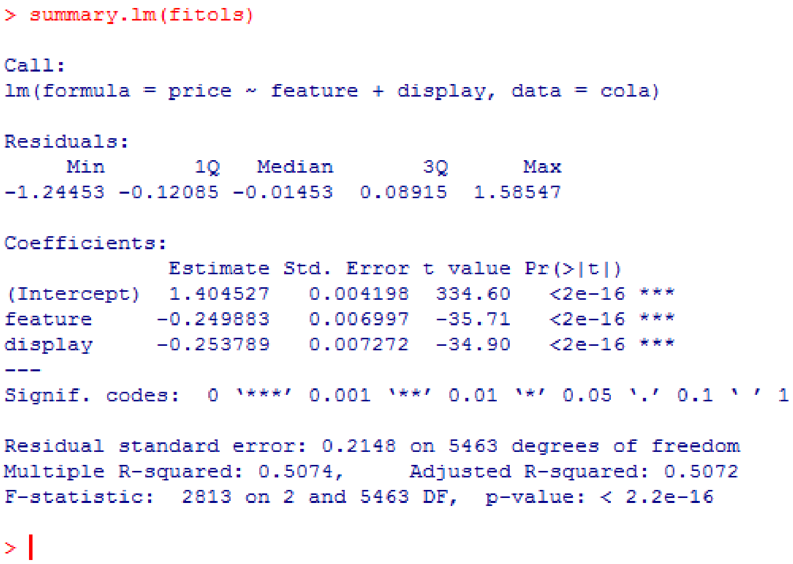
```
> cola <- read.table("C:/Users/leead/AppData/Roaming/gretl/Rdata.tmp",
+                    header = TRUE )
```

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** 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.

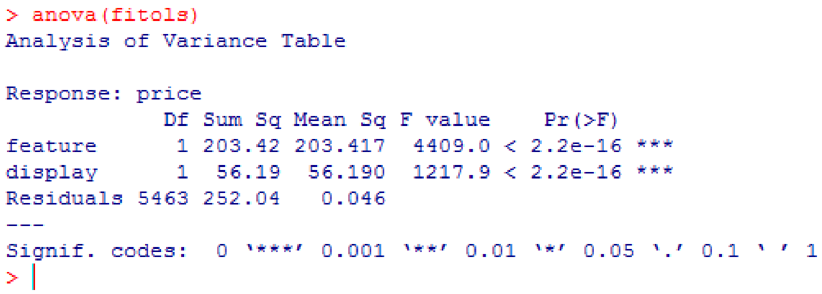


```
> summary.lm(fitols)

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 E.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.

[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 E.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 E.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.

## E.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.

## E.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`. Be sure to point it toward the appropriate directory and file. 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 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)`.

## E.5 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 very fancy econometrics without having to program from scratch. 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.

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 **MCMCpack**. **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.

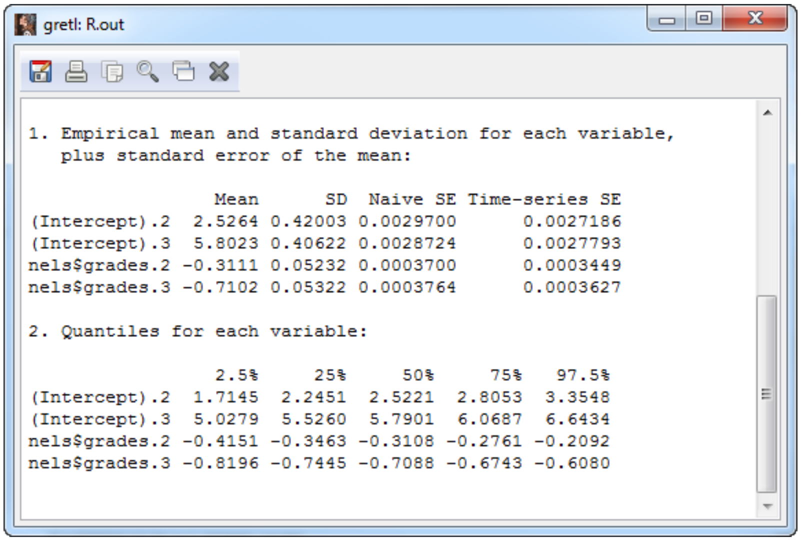
### E.5.1 Multinomial Logit

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**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.

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 E.5 In the MNL model, the estimates from **MCMCpack** are a little different from those produced by **gretl**, but they are reasonably close. The quantiles are useful for several reasons. As you can see,



```
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 E.5: Multinomial logit results from the MCMCmnl estimator in **R**

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.

### E.5.2 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 sequence 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.

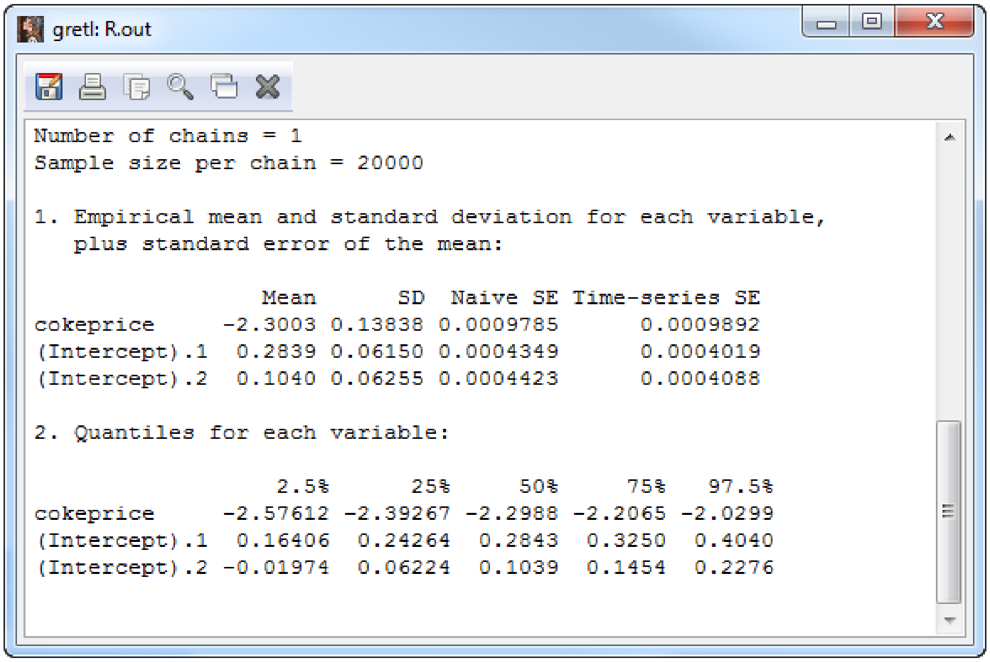
Once the data are arranged, load the **MCMCpack** library and use MCMCmnl to estimate the model. The conditional logit model uses choice specific variables. For MCMCmnl these choice-specific covariates have to be entered using a special syntax: choicevar(cvar,"var","choice") where cvar is the name of 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 E.6.



```
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 E.6: Conditional logit results from the MCMCoprobit estimator in **R**

### E.5.3 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.[3] The difference between **MCMCpack** and 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)
```

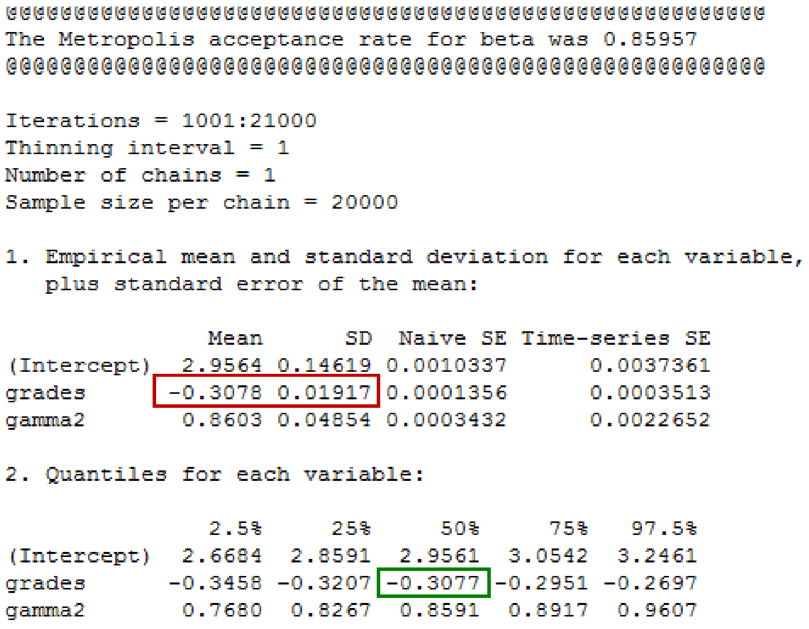
The first line converts the generic `gretldata` data frame that is loaded when you launch R from within **gretl**. The second line creates the data object called `nels`. The `attach(nels)` statement allows you to refer to the variables in `nels` data frame directly by their names.

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 E.7. One important difference between **MCMCpack** 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 $\beta$, which is highlighted in Figure E.7, 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 *POE5*'s no-intercept model is $\mu_2 = -(Intercept - \gamma_2)$, where $\gamma_2$ 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.

[3] Of course, if you decide to use more information in your prior then they can be substantially different.



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
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 E.7: Ordered probit results from the MCMCoprobit estimator in **R**

## E.6 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. Also, see Constantin Colonescu's companion manual for *POE5*, which is available from Amazon.com.

# GNU Free Documentation License

Version 1.3, 3 November 2008

### 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.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

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.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf 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 the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/licenses/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

# ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Bibliography

Adkins, Lee C. (2009), 'An instrumental variables probit estimator using gretl', *http://www.learneconometrics.com/pdf/GC2009.pdf* .

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.

Cottrell, Allin and Riccardo "Jack" Lucchetti (2018), *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.

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 (2018), *Principles of Econometrics*, 5th 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.

McClain, K.T. and J. M. Wooldridge (1995), 'A simple test for the consistency of dynamic linear regression in rational districuted lag models', *Economics Letters* **48**, 235–240.

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.

Staiger, D. and J. H. Stock (1997), 'Instrumental variables with weak instruments', *Econometrica* **65**, pp. 557–586.

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