LISA SCHOPOHL
ROBERT WICHMANN
CHRIS BROOKS

STATA GUIDE TO ACCOMPANY

INTRODUCTORY ECONOMETRICS FOR FINANCE

4TH EDITION

This guide draws on material from 'Introductory Econometrics for Finance', published by Cambridge University Press, © Chris Brooks (2019). The Guide is intended to be used alongside the book, and page numbers from the book are given after each section and subsection heading.

The authors accept no responsibility for the persistence or accuracy of URLs for external or third-party internet websites referred to in this work, and nor do we guarantee that any content on such web sites is, or will remain, accurate or appropriate.

Contents

List of Figures

List of Tables

1 Getting Started

1.1 What is Stata?

Stata is a statistical package for managing, analysing, and graphing data.[1] Stata's main strengths are handling and manipulating large datasets, and its ever-growing capabilities for handling panel and time-series regression analysis. Besides its wealth of diagnostic tests and estimation routines, one feature that makes Stata a very suitable econometrics software package for both novices and experts of econometric analyses is that it may be used either as a point-and-click application or as a command-driven package. Stata's graphical user interface provides an easy interface for those new to Stata and for experienced Stata users who wish to execute a command that they seldom use. The command language provides a fast way to communicate with Stata and to implement more complex ideas. In this guide we will primarily be working with the graphical user interface. However, we also provide the corresponding command language, where suitable.

This guide is based on Stata 15.1. Please note that if you use an earlier version of Stata, the design of the specification windows as well as certain features of the menu structure might differ. As different statistical software packages might use different algorithms for some of their estimation techniques, the results generated by Stata might not be comparable to those generated by EViews in each instance.

A good way of familiarising yourself with Stata is to learn about its main menus and their relationships through the examples given in this guide.

This section assumes that readers have a licensed copy of Stata and have successfully loaded it onto an available computer. There now follows a description of the Stata package, together with instructions to achieve standard tasks and sample output. Any instructions that must be entered or icons to be clicked are illustrated by **bold-faced type**. Note that Stata is CASE-SENSITIVE. Thus, it is important to enter commands in lower-case and to refer to variables as they were originally defined, i.e., either as lower-case or CAPITAL letters.

1.2 What Does Stata Look Like?

When you open Stata, you will be presented with the Stata main window, which should resemble Figure 1. You will soon realise that the main window is actually sub-divided into several smaller windows. The five most important windows are the *Review*, *Output*, *Command*, *Variables*, and *Properties* windows (as indicated in Figure 1). This subsection briefly describes the characteristics and main functions of each window. There are other, more specialized windows such as the Viewer, Data Editor, Variables Manager, and Do-file Editor – which are discussed later in this guide.[2]

[1] This subsection is based on the description provided in the Stata manual [U] User's Guide.

[2] This section is based on the Stata manual [GS] Getting Started. The intention of this subsection is to provide a brief overview of the main windows and features of Stata. If you would like a more detailed introduction to Stata's user interface, refer to Chapter 2, The Stata User Interface, in the above-mentioned manual.

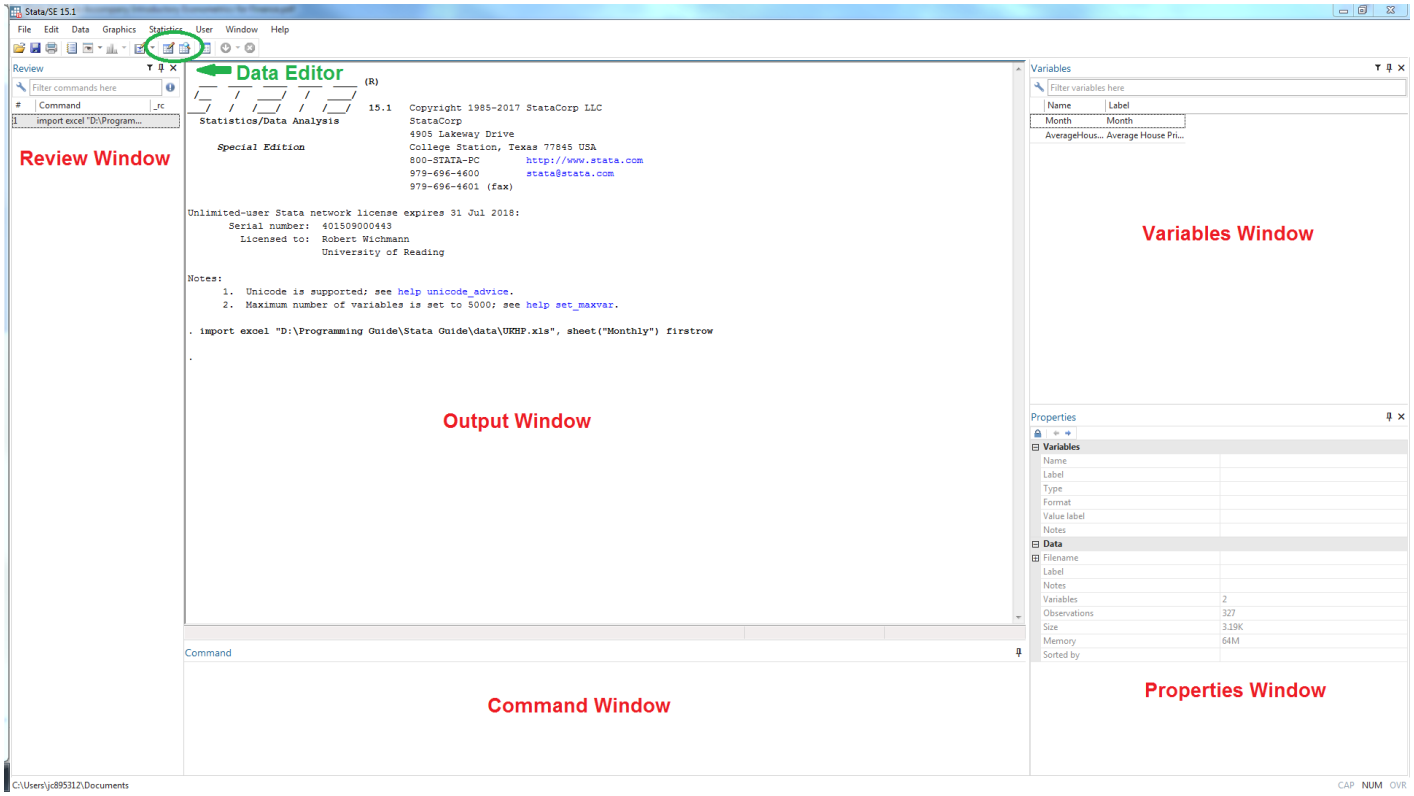


Figure 1: The Stata Main Windows

The *Variables* window to the right shows the list of variables in the dataset, along with selected properties of the variables. By default, it shows all the variables and their labels. You can change the properties that are displayed by right-clicking on the header of any column of the Variables window.

Below the Variables window you will find the *Properties* window. It displays variable and dataset properties. If you select a single variable in the Variables window, this is where its properties are displayed. If there are multiple variables selected in the Variables window, the Properties window will display properties that are common across all selected variables.

Commands are submitted to Stata via the *Command* window. Assuming you know what command you would like to use, just type it into the command window and press **Enter** to execute it. When a command is executed – with or without error – the output of that command (e.g., the table of summary statistics or the estimation output) appears in the *Output* window. It contains all the commands that you have entered during the Stata session and their textual results. Note that the output of a particular test or command shown in the Results window does not differ depending on whether you use the command language or the point-and-click menu to execute it.

Besides being able to see the output of your commands in the Output window, the command line will also appear in the *Review* window at the left-hand side of the main window. The Review window shows the history of all commands that have been entered during one session. Note that it displays successfully executed commands in black and unsuccessful commands – along with their error codes – in red. You may click on any command in the Review window and it will reappear in the Command window, where you can edit and/or resubmit it.

The different windows are interlinked. For instance, by double-clicking on a variable in the *Variables* window you can send it to the *Command* window or you can adjust or re-run certain commands you have previously executed using the *Review* window.

There are two ways by which you can tell Stata what you would like it to do: you can directly type the instruction into the Command window or you can use the click-and-point menu. Access to the click-and-point menu can be found at the top left of the Stata main window. You will find that

the menu is divided into several subcategories based on the features they comprise: *File*, *Edit*, *Data*, *Graphics*, *Statistics*, *User*, *Window*, and *Help*.

The *File* menu icon comprises features to open, import, export, print, or save your data. Under the *Data* icon, you can find commands to explore and manage data as well as functions to create or change variables or single observations, to sort data or to merge datasets. The *Graphics* icon is relatively self-explanatory as it covers all features related to creating and formatting graphics in Stata. Under the **Statistics** icon, you can find all the commands and functions to create customised summary statistics and to run estimations. You can also find post-estimation options and commands to run diagnostic (misspecification) tests. Another useful icon is the *Help* icon under which you can get access to the Stata pdf manual as well as other help and search features.

When accessing certain features through the Stata menu (usually) a new dialogue window appears where you are asked to specify the task you would like Stata to perform.

Below the menu icons we find the *Toolbar*. The *Toolbar* contains buttons that provide quick access to Stata's more commonly used features. If you forget what a button does, hold the mouse pointer over the button for a moment, and a tool-tip will appear with a description of that button. In the following, we will focus on those toolbar buttons of particular interest to us.

The *Log* button begins a new log or closes, suspends, or resumes the current log. Logs are used to document the results of your session – more on this issue in subsection 2.8 – Keeping Track of Your Work.

The *Do-file Editor* opens a new do-file or re-opens a previously stored do-file. Do-files are mainly used for programming in Stata but can also be handy to store a set of commands in order to allow you to replicate certain analyses at a later point in time. We will discuss the use of do-files and programming Stata in more detail in later sections.

There are two icons to open the *Data Editor*. The Data Editor gives a spreadsheet-like view of the data. The icon resembling a spreadsheet with a loop opens the Data Editor in the Browse-mode while the spreadsheet-like icon with the pen opens the editor in the Edit-mode. The former only allows you to inspect the data. In the edit mode, however, you can make changes to the data, e.g., overwriting certain data points or dropping observations.

Finally, there are two icons at the very right of the toolbar: a green downward-facing arrow and a red sign with a cross. The former is the *Clear–more–Condition* icon which tells Stata to continue when it has paused in the middle of a long output. The latter is the *Break* icon and pressing it while Stata executes a command stops the current task.

1.3 Getting Help

There are several different ways to get help when using Stata.[3] Firstly, Stata has a very detailed set of manuals which provide several tutorials to explore some of the software's functionalities. Especially when getting started using Stata, it might be useful to follow some of the examples provided in the Stata pdf manuals. These manuals come with the package but can also be directly accessed via the Stata software by clicking on the **Help** icon in the icon menu. There are separate manuals for different subtopics, e.g., for graphics, data management, panel data, etc. There is also a manual called [GS] Getting started, which covers some of the features briefly described in this introduction in more detail.

Sometimes you might need help regarding a particular Stata function or command. For every command, Statas in-built support can be called by typing **'help'** followed by the command in question in the Command window or via the *Help* menu icon. The information you will receive is an abbreviated version of the Stata pdf manual entry.

[3]For a very good overview of Stata's help features and useful resources, refer to the manual entries '4 Getting help' in the [GS] manual and '3 Resources for learning and using Stata' in the [U] manual.

For a more comprehensive search or if you do not know what command to use, type in **'search'** or **'findit'** followed by specific keywords. Stata then also provides links to external (web) sources or user-written commands regarding your particular enquiry.

Besides the help features that come with the Stata package, there is a variety of external resources. For instance, the online Stata Forum is a great source if you have questions regarding specific functionalities or do not know how to implement particular statistical tests in Stata. Browsing of questions and answers by the Stata community is available without registration; if you would like to post questions and or provide answers to a posted question you need to register to the forum first (`http://www.statalist.org/`). Another great resource is the Stata homepage. There you can find several video tutorials on a variety of topics (`http://www.stata.com/links/video-tutorials/`). Several academics also publish their Stata code on their institution's website and several U.S. institutions provide Stata tutorials that can be accessed via their homepages (e.g., `http://www.ats.ucla.edu/stat/stata/`).

2 Data Management

2.1 Variables and Data Types

It is useful to learn a bit about the different data types and variable types used in Stata as several Stata commands distinguish between the type of data that you are dealing with and several commands require the data to be stored as a certain data type.

Numeric or String Data

We can broadly distinguish between two data types: numeric and string. Numeric is for storing numbers; string resembles a text variable. Variables stored as numeric data can be used for computation and in estimations. Stata has different numeric formats or storage types which vary according to the number of integers they can capture. The more integers a format type can capture the greater its precision, but also the greater the storage space needed. Stata's default numeric option is float which stores data as a real number with up to 8 digits. This is sufficiently accurate for most work. Stata also has the following numeric storage types available: *byte* (integer, e.g., for dummy variables), *int* (integer, e.g., for year variables), *long* (integer, e.g., for population data), and *double* (real number with 16 digits of accuracy).

Any variable can be designated as a string variable, even numbers. However, in the latter case Stata would not recognise the data as numbers anymore but would treat them as any other text input. No computation or estimations can be performed on string variables. String variables can contain up to 244 characters. String values have to be put in quotation marks when being referred to in Stata commands.

To preserve space, only store variables with the minimum storage requirements.[4]

Continuous, Categorical, and Indicator Variables

Stata has very convenient functions that facilitate the work and estimation with categorical and indicator variables but also other convenient data manipulations such as lags and leads of variables.[5]

Missing Values

Stata marks missing values in series by a dot. Missing numeric observations are denoted by a single dot (.), while missing string observations are referred to either by blank double quotes (" ") or dot double quotes (".."). Stata can define multiple different missing values, such as .a, .b, .c, etc. This is useful to distinguish between the reasons why a data point is missing, such as the data point was missing in the original dataset or the data point has been manually removed from the dataset. The largest 27 numbers of each numeric format are preserved for missing values. This is important to keep in mind when applying constraints in Stata. For example, typing the command 'describe age if age>= 27' includes observations for which the person's age is missing, while the command 'describe age if age>=27 & age!=.' excludes observations with missing age data.

2.2 Formats and Variable Labels

Each variable may have its own display format. This does not alter the content or precision of the variable but only affects how it is displayed. You can change the format of a variable by clicking on

[4] A very helpful command to check whether data can be stored in a data type that requires less storage space is by using the **'compress'** command.

[5] More on this topic can be found in Stata Manual [U] User's Guide, 25 Working with categorical data and factor variables.

Data/Variable Manager, by directly changing the format in the Variable Manager window at the bottom right of the Stata main screen or by using the command 'format' followed by the new format specification and the name of the variable. There are different format types that correspond to ordinary numeric values as well as specific formats for dates and time.

We can attach labels to variables, an entire dataset or to specific values of the variable (e.g., in the case of categorical or indicator variables). Labelling a variable might be helpful to document the content of a certain variable or how it has been constructed. You can attach a label to a variable by clicking in the label dialogue box in the *Variable Manager* window or by clicking on **Data/Variable Manager**. A value label can also be attached to a variable using the *Variable Manager* window.[6]

2.3 Data Input and Saving

One of the first steps of every statistical analysis is importing the dataset to be analysed into the software package. Depending on the format of the data there are different ways of accomplishing this task. If the dataset is already in a Stata format which is indicated by the file suffix **'.dta'**, click on **File/Open...** and simply select the dataset to work with. Alternatively, use the **'use'** command followed by the name and location of the dataset, e.g., 'use "G:\Stata training\sp500.dta", clear'. The term **'clear'** tells Stata to close all workfiles that are currently in memory.

If your dataset is in Excel format you click on **File/Import/Excel spreadsheet (*.xls;*.xlsl)** and select the Excel file to import into Stata. A new window appears which provides a preview of the data and in which you can specify certain options as to how you would like the dataset to be imported. If you prefer to use a command to import the data, type the command **'import excel'** into the command window followed by the file name and location as well as potential importing options.

Stata can also import text data, data in SAS format, and other formats (see **File/Import**) or you can directly paste observations into the *Data Editor*. You save changes to the data using the command **save** or by selecting the **File/Save as...** option in the menu. Stata 15 also provides an add-in to import data directly from the FRED data base.[7]

When you read data into Stata, it stores it in the RAM (memory). All changes you make are temporary and will be lost if you close the file without saving it. It is also important to keep in mind that Stata has no Undo options so that some changes cannot be undone (e.g., dropping of variables). You can generate a snapshot of the data in your Data Editor or by the command **'snapshot save'** to be able to reset data to a previous stage.

[6] We will not focus on value labels in this guide. The interested reader is advised to refer to the corresponding entries in the Stata manual to learn more about value labels.

[7] This data are publicly available on the website of the Federal Reserve Bank of St. Louis (`https://fred.stlouisfed.org/`). We will use it in section 9.

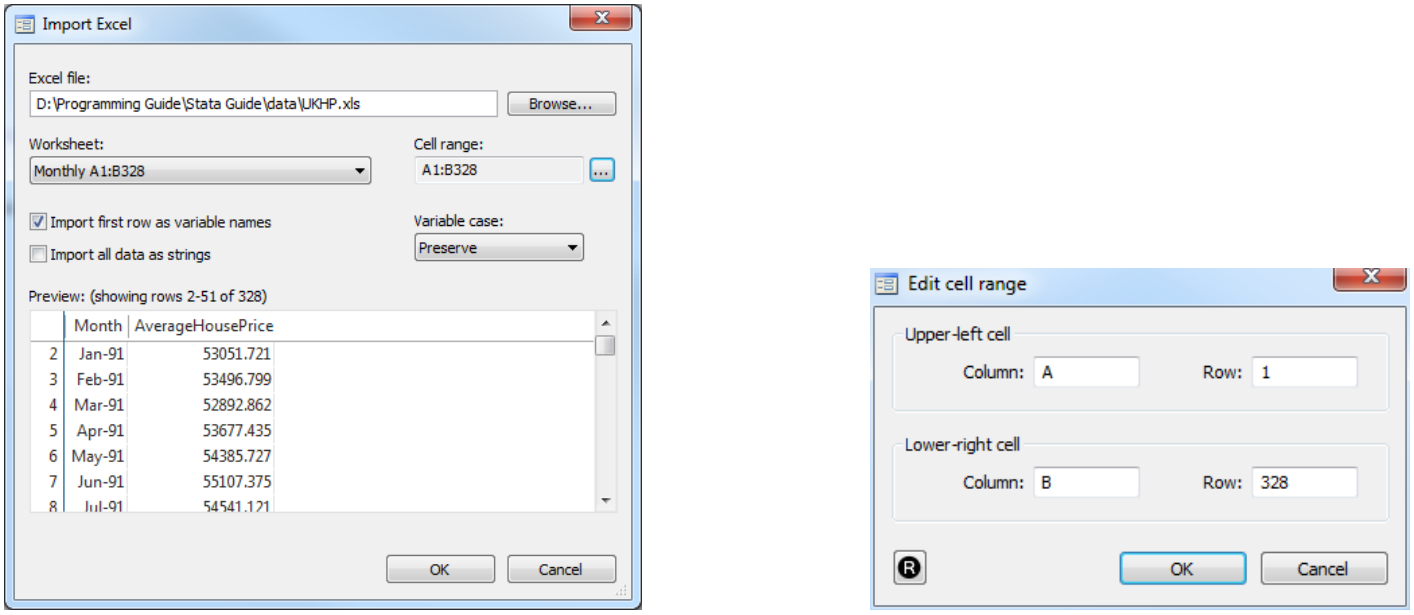


Figure 2: Importing Excel Data into Stata

Now let us import a dataset to see how the single steps would be performed. The dataset to import into Stata is the excel file **UKHP.xls**. First, select **File/Import/Excel spreadsheet (*.xls;*.xlsx)** and click on the button **Browse...** . Now choose the 'UKHP.xls' file, click **Open** and you should find a preview of the data at the bottom of the dialogue window, as shown in Figure 2, left panel. We could further specify the worksheet of the Excel file we would like to import by clicking on the drop-down menu next to the box **Worksheet**; however, since the 'UKHP.xls' file only has one worksheet we leave this box as it is.

The box headed **Edit cell range** allows the user to specify the cells to import from a specific worksheet. By clicking on the button with the three dots we can define the cell range. In our case we want the Upper-left cell to be A1 and the lower-right cell to be B328 (see Figure 2, right panel). Finally, there are two boxes that can be checked: **Import first row as variable names** tells Stata that the first rows are not to be interpreted as data points but as the names of the variables. Since this is the case for **'UKHP.xls'** we check this box. The second option, **Import all data as strings** tells Stata to store all series as string variables, independent of whether the original series might contain numbers. We want Stata to import our data as numeric values and thus we leave this box unchecked. Once you have undertaken all these specifications, the dialogue window should resemble Figure 2, left panel. The last thing we need to do to import the data into Stata is to press **OK**. If the task has been successful we should find the command line **import excel** followed by the file location and further specifications in the *Output* window as well as the *Review* window. Additionally, there should now be two variables in the Variables window: **Month** and **AverageHousePrice**. Note that Stata automatically attached a variable label to the two series in the workfile which are identical to the variable names.

You can save the imported dataset as a Stata workfile by clicking on **File/Save as...** and specifying the name of the dataset – in our case 'ukhp.dta'. Note that we recognise that the dataset is now stored as a Stata workfile by the file-suffix '.dta'.

2.4 Data Description

Once you have imported the data, you want to get an idea of what the data are like and you want to check that all the values have been correctly imported and that all variables are stored in the correct format.

There are several Stata functions to examine and describe your data. It is often useful to visually inspect the data first. This can be done in the *Data Editor* mode. You can access the *Data Editor* by clicking on the respective icons in the Stata icon menu as described above. Alternatively, you can directly use the commands **browse** or **edit**.

Additionally, you can let Stata describe the data for you. If you click on **Data/Describe data** you will find a variety of options to get information about the content and structure of the dataset.

In order to describe the data structure of the **'UKHP.dta'** file you can use the **describe** command which you can find in the Stata menu under **Describe data in memory or in a file**. It provides the number of observations and variables, the size of the dataset as well as variable specifics such as storage type and display format. If we do this for our example session, we are presented the following output in the Output window.[8]

```
. describe

Contains data from D:\Programming Guide\Stata Guide\stata_files\ukhp.dta
  obs:           327
 vars:             2                          16 Apr 2018 17:29
 size:         3,270
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
Month           int     %tdMon-YY              Month
AverageHouseP~e double  %10.0g                 Average House Price
-------------------------------------------------------------------------------
Sorted by:

.
```

We see that our dataset contains two variables, one that is stored as **'int'** and one as **'double'**. We can also see the the display format and that the two variables have a variable label but no value label attached.

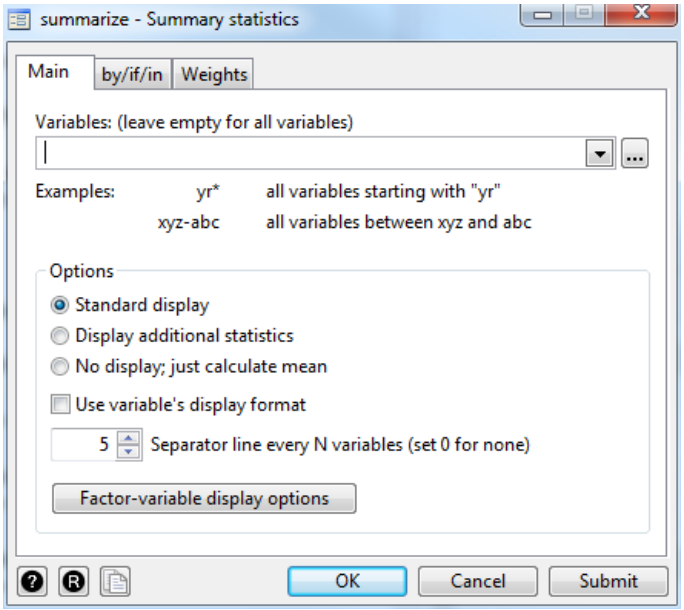


Figure 3: Generating Summary Statistics

[8]To separate output from text, we will always mark Stata output with a thick horizontal line above and below.

The command **summarize** provides summary statistics of the variables. You can find it under **Data/Describe data/Summary statistics**. If we click on this option, a new dialogue window appears where we can specify what variables we would like to generate summary statistics for (Figure 3).

If you do not specify any variables, Stata assumes you would like summary statistics for all variables in memory. Again, it provides a variety of options to customise the summary statistics. Besides **Standard display**, we can tell Stata to **Display additional statistics**, or (rather as a program command for subsequent use) to merely calculate the mean without showing any output.[9] Using the tab **by/if/in** allows us to restrict the data to a subsample. The tab **Weights** provides options to weight the data points in your sample. However, we want to create simple summary statistics for the two variables in our dataset so we keep all the default specifications and simply press **OK**. Then the following output should appear in our Output window.

```
. summarize

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       Month |        327    16283.75    2877.614       11323      21244
AverageHou~e |        327    124660.5    56387.17    49601.66   211755.9

.
```

Note that the summary statistics for 'Month' are not intuitive to read as Stata provides summary statistics in the coded format and does not display the variable in the (human readable) date format.

Another useful command is '**codebook**', which can be accessed via **Data/Describe data/Describe data contents (codebook)**. It provides additional information on the variables, such as summary statistics on numeric variables, examples of data points for string variables, the number of missing observations, and some information about the distribution of the series. The 'codebook' command is especially useful if the dataset is unknown and you would like to get a first overview of the characteristics of the data. In order to open the **command** dialogue window we follow the path mentioned above. Similar to the 'summarize' command, we can execute the task for all variables by leaving the **Variables** box blank. Again, we can select further options using the other tabs. If we simply press **OK** without making further specifications, we generate the output on the next page.

We can see from the output below that the variable **Month** is stored as a daily variable with days as units. However, the units of observations are actually months so that we will have to adjust the type of the variable to a monthly time variable (which we will do in the next section).

There is a variety of other tools that can be used to describe the data and you can customise them to your specific needs. For instance, with the 'by' prefix in front of a summary command you can create summary statistics by subgroups. You can also explicitly specify the statistics that should be reported in the table of summary statistics using the command **tabstat**. To generate frequency tables of specific variables, the command **tabulate** can be used. Another great feature of Stata is that many of these commands are also available as panel data versions.

[9] An application where the latter option will prove useful is presented in later sections of this guide that introduce programming in Stata.

```
. codebook

-------------------------------------------------------------------------------
Month
-------------------------------------------------------------------------------

                  type:  numeric daily date (int)

                 range:  [11323,21244]                units:  1
       or equivalently:  [01jan1991,01mar2018]        units:  days
         unique values:  327                      missing .:  0/327

                  mean:   16283.7 = 31jul2004 (+ 18 hours)
             std. dev:   2877.61

           percentiles:        10%       25%       50%       75%       90%
                             12297     13788     16284     18779     20270
                          01sep1993 01oct1997 01aug2004 01jun2011 01jul2015

-------------------------------------------------------------------------------
AverageHousePrice
-------------------------------------------------------------------------------

                  type:  numeric (double)

                 range:  [49601.664,211755.93]        units:  .0001
           unique values:  327                      missing .:  0/327

                  mean:    124660
             std. dev:   56387.2

           percentiles:        10%       25%       50%       75%       90%
                            51696.5   61488.6    150946    169316    195279

.
```

2.5 Changing Data

Often you need to change your data by creating new variables, changing the content of existing data series or by adjusting the display format of the data. In the following, we will focus on some of the most important Stata features to manipulate and format the data, though this list is not exhaustive. There are several ways you can change data in your dataset.

One of the simplest is to rename the variables. For instance, the variable name **'AverageHouse-Price'** is very long and it might be very inconvenient to type such a long name every time you need to refer to it in a Stata task. Thus, we want to change the name of the variable to **'hp'**. To do so, we click on **Data/Data utilities** and then select **Rename groups of variables**. A new dialogue window appears where we can specify exactly how we would like to rename our variable (Figure 4).

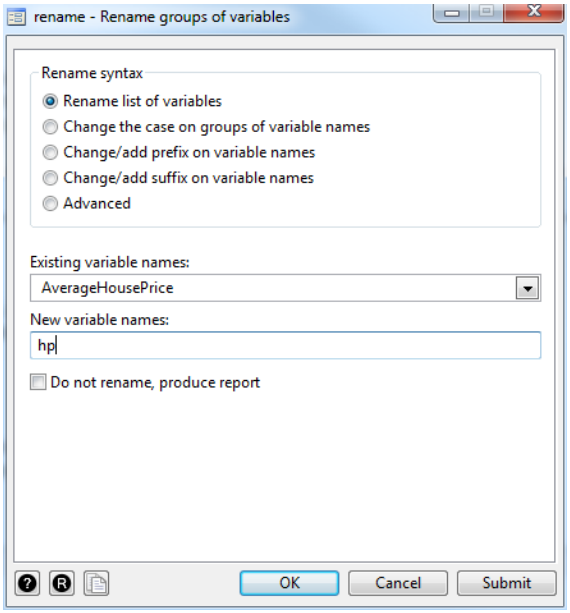


Figure 4: Renaming Variables

As you can see from the list of different renaming options, the dialogue box offers a variety of ways to facilitate renaming a variable, such as changing the case on a variable (from lower-case to upper-case, and vice versa) or by adding a prefix or suffix to a variable. However, we want to simply change the name of the variable to a predefined name so we keep the default option **Rename list of variables**. Next, choose the variable we want to rename from the drop-down menu next to the Existing variable names box, i.e., **AverageHousePrice**. Now we simply need to type in the new name in the dialogue box **New variable names** which is **hp**.[10] By clicking **OK**, Stata performs the command. If we now look at the *Variables* window we should find that the data series bears the new name. Additionally, we see that Stata shows the command line that corresponds to the 'rename' specification we have just performed in the Output window and the *Review* window:

rename (AverageHousePrice) (hp)

Thus, we could have achieved the same result by typing the above command line into the *Command* window and pressing **Enter**. Stata also allows you to drop specific variables or observations from the dataset, or, alternatively, to specify the variables and/or observations that should be kept in the dataset, using the **drop** or **keep** commands, respectively. In the Stata menu, these commands can be accessed via **Data/Create or change data/Drop or keep observations**. To drop or keep variables, you can also simply right-click on a variable in the *Variables* window and select **Drop selected variables** or **Keep only selected variables**, respectively. As we do not want to remove any variables or observations from our 'ukhp.dta' dataset, we leave this exercise for future examples.

If you intend to change the content of variables you use the command **replace**. We can access this command by clicking **Data/Create or change data/Change contents of variable**. It follows a very similar logic to the command that generates a new variable. As we will explain in detail how to generate new variables in the next section and as we will be using the 'replace' command in later sections we will not go into further detail regarding this command.

Other useful commands to change variables are **destring** and **encode**. We only provide a brief description of the functionalities of each command. The interested reader is advised to learn more

[10]Note that using this dialogue window you can rename a group of variables at the same time by selecting all variables you would like to rename in the 'Existing variable names' box and listing the new names for the variables in the matching order in the 'New variable names' box.

about these commands from the Stata pdf manuals. Sometimes Stata does not recognise numeric variables as numeric and stores them as string instead. **destring** converts these string data into numeric variables, while **encode** is another command to convert string into numeric variables. However, unlike **'destring'**, the series that is to be converted into numeric equivalents does not need to be numeric in nature, **'encode'** rather provides a numeric equivalent to a (non-numeric) value, i.e., a coded value.

2.6 Generating New Variables

One of the most commonly used commands is **generate**. It creates a new variable having a particular value or according to a specific expression. When using the Stata menu we can access it following the path **Data/Create or change data/Create new variable**.

Suppose, for example, we have a time-series called Z, the latter can be modified in the different ways so as to create variables (see Table 1, left). Sometimes you might like to construct a variable containing the mean, the absolute, or the standard deviation of another variable or value. To do so, you will need to use the extended version of the 'generate' command, the **egen** function.

Additionally, when creating new variables you might need to employ some logical operators, for example when adding conditions. The right side of Table 1 shows the most commonly used logical operators in Stata.

Table 1: Commonly Used Operators

Z/2	Dividing	==	(exactly) equal to
Z*2	Multiplication	!=	not equal to
Zˆ2	Squaring	>	larger than
log(Z)	Taking the logarithms	<	smaller than
exp(Z)	Taking the exponential	>=	larger than or equal to
L.Z	Lagging the data	<=	smaller than or equal to
log(Z/L.Z)	Creating the log returns	&	AND
		|	OR
		!	NOT

Let us have a look how to practically generate a new variable. For example, as mentioned earlier, the variable **Month** is stored as a daily variable; however, we would like to have a time variable that is stored as a monthly series. Thus, we can use Stata's 'generate' function to create a new monthly time variable. Using the Stata menu to access the 'generate' command, a dialogue box appears where we can specify the contents of the new variable (Figure 5, left panel).

The first thing to be specified is the **Variable type**. The default is **float** and for the time being we will keep the default. Next, we define the name of the new variable. In our case, we could simply use the lower-case version of month to indicate the monthly series, i.e., **Variable name: month**.

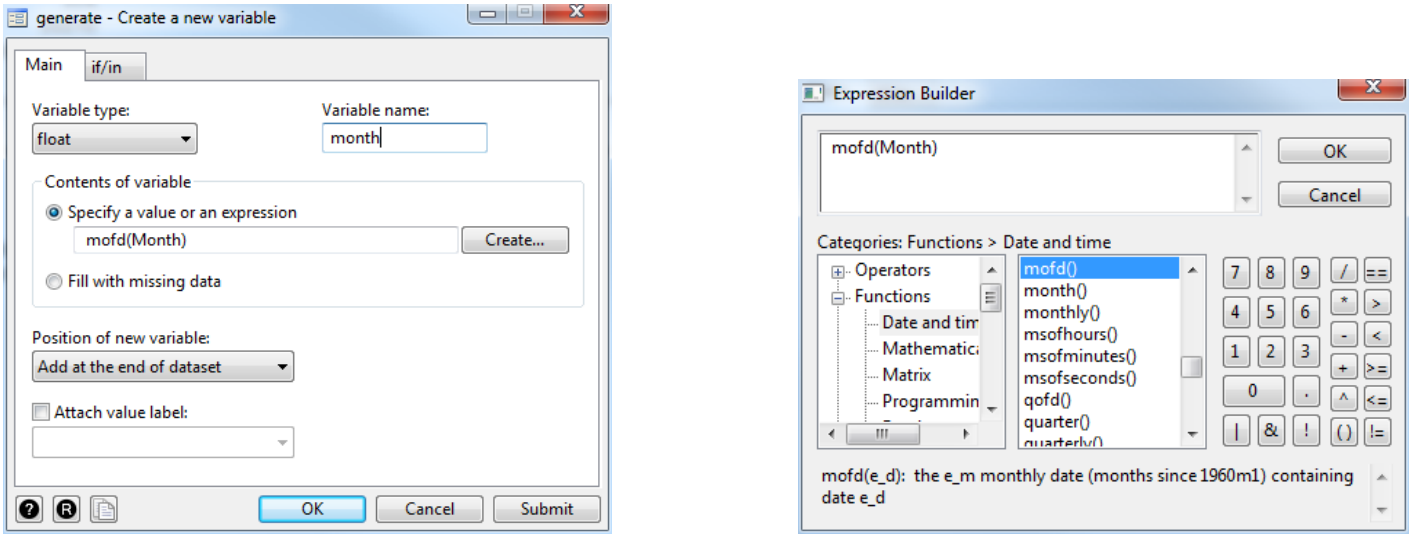


Figure 5: Generating a Monthly Date Series

Next we need to define the **Contents of variable** by selecting the option **Specify a value or an expression**. As stated in the header, the variable content can either be a value, e.g., '2', or it can be an expression. In our case, we need to specify an expression. If we know the function to convert daily to monthly data by heart we can simply type it into the respective box. Otherwise, we could use the 'Expression Builder' by clicking on **Create...** next to the box. The 'Expression Builder' opens and under **Category** we select **Functions/Date and time** (Figure 5, right panel). We now see all date, and time-related functions on the left of the window. We look for **mofd** – i.e., **m**onth **of d**ay – and double click on it. It is then sent to the dialogue box at the top of the 'Expression Builder' window. Type 'Month' in between the brackets and press **OK**. The 'generate' dialogue window should now look like the left panel of Figure 5. We click **OK** again and we should now find a new variable in the Variable window called 'month'.

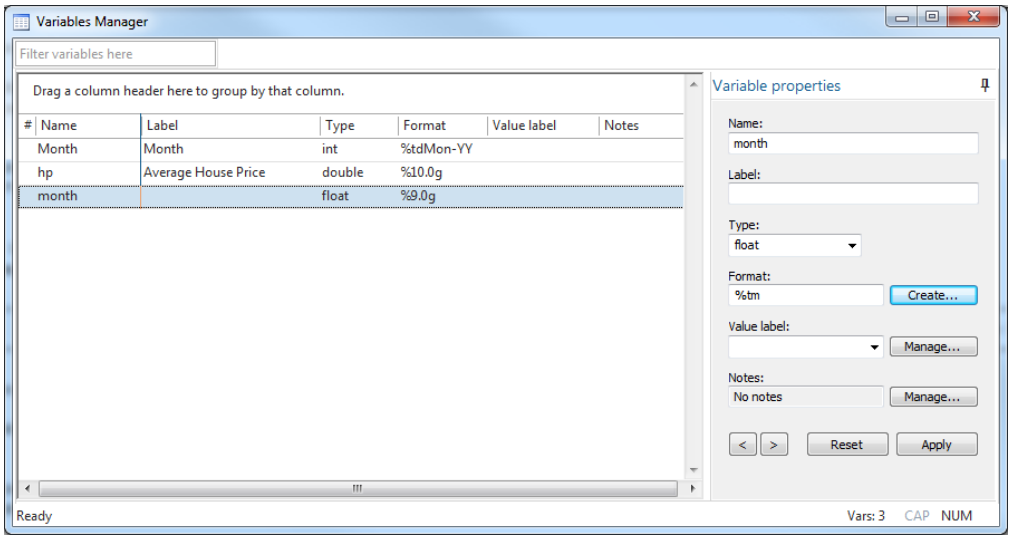


Figure 6: The Variables Manager Window

It is useful to perform the **codebook** command on this variable to see whether we need to make further adjustments. We can either use the Stata menu to access the 'codebook' dialog window (as described above) or we could simply type **codebook month** into the *Command* window and press **Enter**. We see that the format type of the variable of 'month' is not one of the date-related formats. So in order to

make the variable more easily readable we can change its format type. To do so we open the *Variables Manger* by clicking on **Data/Variables Manager** (Figure 6).

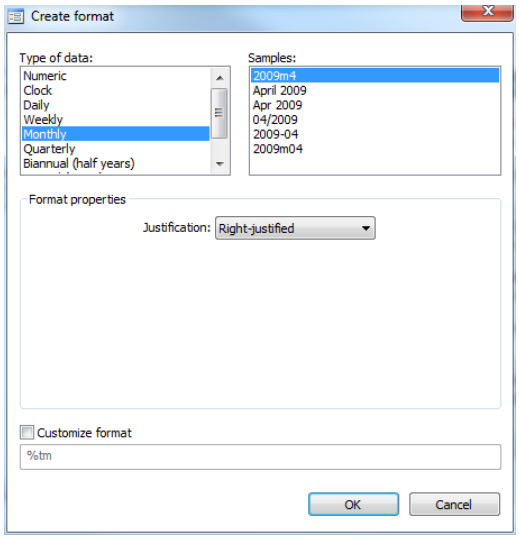


Figure 7: Changing the Format Type for a Monthly Date Variable

Select the variable month from the list and in the **Variable properties** column on the left we search for the box **Format** and click on the button **Create...** next to it. A new dialogue window appears which asks to select the appropriate format (Figure 7). Search for **Monthly** under the **Type of data** heading. We then see several samples of monthly formats and we now simply need to select our preferred format type. In this case, stick with the default, which is the first option in the list. Press **OK** to return to the Variables Manager and then select **Apply** at the bottom-left of the window so that the changes become effective. After having done so we can simply close the window.

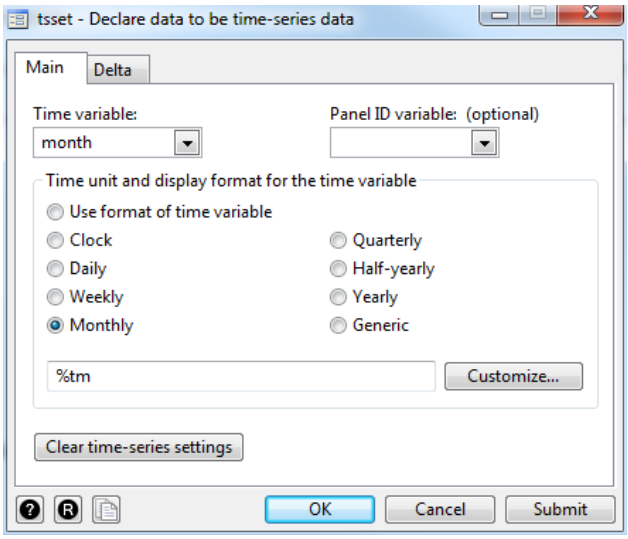


Figure 8: Declaring Your Dataset to be Time-Series Data

If you re-run the **'codebook'** command on **'month'** you can see that the format type has successfully been changed to a monthly format. Finally, tell Stata that the data are in time-series form (as compared to cross-sectional or panel data). This is necessary to use certain time-series commands or to use leads and lags of variables. You can do this by selecting **Statistics/Time series/Setup and utilities/Declare data to be time-series data**. In the dialog window that appears choose **Time**

variable: month and further select **Monthly** from the **Time unit and display format** options (Figure 8). Leave everything else blank and press **OK**. If we have successfully set up our time-series data, we should read the following lines in the *Output* window:

```
. tsset month, monthly
        time variable:  month, 1991m1 to 2018m3
                delta:  1 month

.
```

Moreover, it is of interest to generate a further variable containing simple percentage changes in house prices. Again, open the **'generate'** dialog window (Figure 9). Define the new series to have the **Variable name: dhp**, while keeping the **float** variable type. With respect to the contents of the new variable, type the following expression:

100*(hp-L.hp)/L.hp

Note that the term **L.hp** indicates a one-period lag of the **'hp'** series. We then simply press **OK** and the new variable **'dhp'** should appear in the *Variable* window.

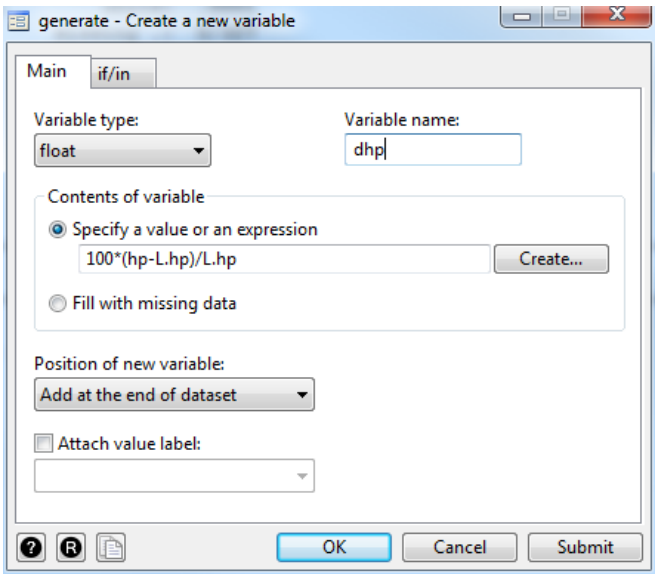


Figure 9: Generating Simple Percentage Changes in House Prices

2.7 Plots

Stata supports a wide range of graph types that can be found under the **Graphics** icon in the menu, including line graphs, bar graphs, pie charts, mixed-line graphs, scatterplots and a number of dataset-specific graphs such as time-series-specific or panel-specific plots. There is a variety of options to select the line types, colour, border characteristics, headings, shading, and scalding, including dual scale graphs. Legends are automatically created (although they can be removed if desired), and customised graphs can be incorporated into other Windows applications using copy-and-paste, or by exporting as another file format, including Windows metafiles, portable networks graphics, or pdf. For the latter you simply click on **File/Save as...** in the 'Graph' window and select the desired file format.

Assume that we would like to construct a line plot of the 'hp' series. To do so, we select **Graphics/Time-series graphs/Line plots** (Figure 10, left panel). In the dialogue window that appears we click on **Create...** and a new window pops up, entitled 'Plot 1' (Figure 10, right panel).

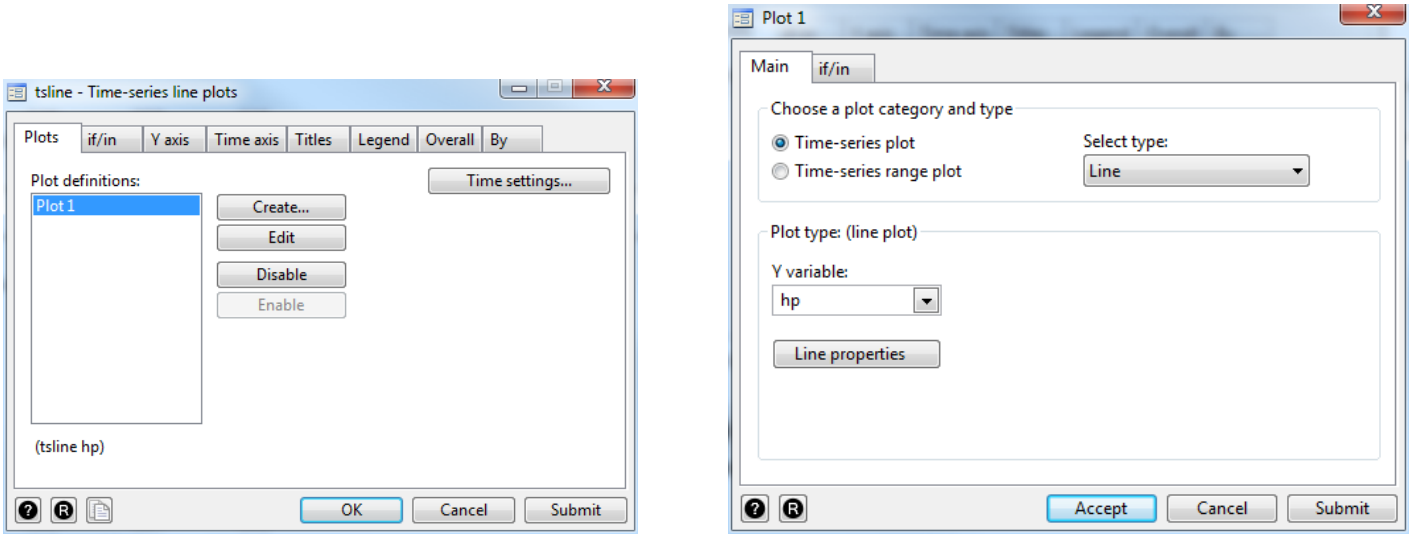


Figure 10: Creating a Line Plot of the Average House Price Series

In this window, we can first **Choose a plot category and type**. As we want to create a line plot we keep the default option **Time-series plot**. Next we **Select type** of the graph. From a number of different graph types, including 'Line', 'Scatter', 'Connected', 'Bar', etc., we choose **Line**. Now we just specify which variable we want to plot. As we want to plot the average house price series we select **Y variable: hp** from the drop-down menu. Note that by clicking on the button **Line properties** we could adjust the line type and colour. However, for the time being we keep the default options and simply select **Accept** in order to return to the main dialogue window. We could now add more series to our graphs by clicking on **Create...** again. We can also **Edit** 'Plot 1' by clicking the respective button. Additionally, there are a number of other options available using the further tabs, e.g., restricting the sample being plotted or formatting the Y and X axes, the titles and the legend. By clicking **OK**, the specified graph appears in a new window (Figure 11).

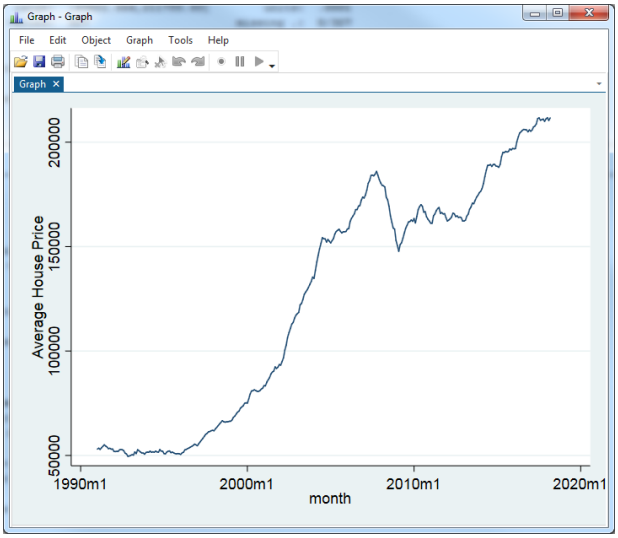


Figure 11: Line Plot of the Average House Price Series

The 'Graph' window also has a menu and toolbar which enables you to save and copy the graph or to customise it further. For instance, by clicking on the icon resembling a bar chart with a pen you open the *Graph Editor* which allows you to adjust the scaling, the style of the axes and many more.

Another commonly used plot is the histogram that gives us a graphical illustration of the distribution of a series. It is basically a bar chart that demonstrates the proportion of the series that falls in a specific range of values. To show how to create a histogram using Stata we will employ the 'dhp' series. Again we select the **Graphics** section in the Stata menu and click on **Histogram** to open a new dialogue window (Figure 12, upper panel).

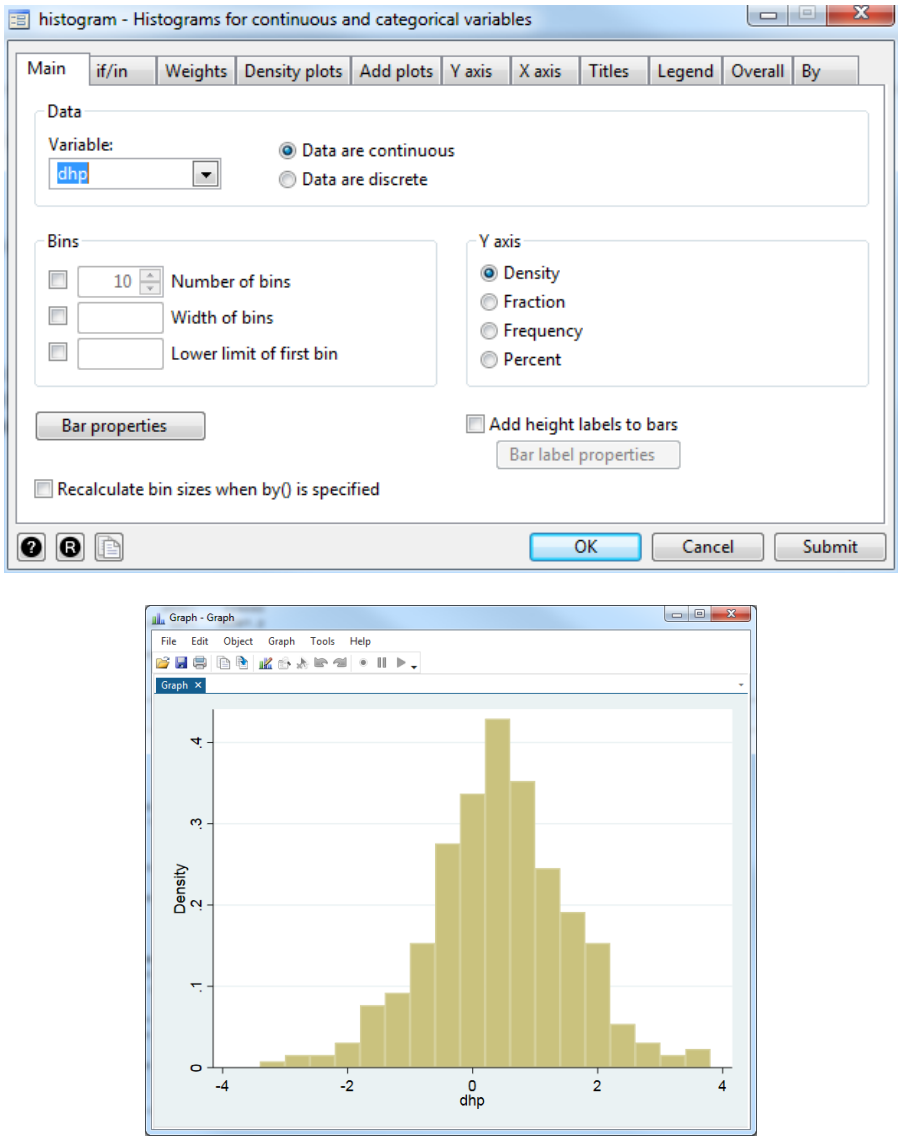


Figure 12: Creating a Histogram

First, we need to specify **Variable: dhp** from the drop-down menu. Stata also asks us whether the data are continuous or discrete. As the 'dhp' series is a continuous variable we can keep the default option. The next boxes allow us to further customise the graph. The **Bins** box, for instance, enables us to choose the number of bins or the bin width. Using the **Y axis** box we can also select whether the histogram is expressed in terms of the density, the faction, frequency, or percentage of the distribution. Additionally, a series of other adjustment options is available when using the further tabs. Simply click **OK** and the histogram of the **'dhp'** series should resemble Figure 12, lower panel.

2.8 Keeping Track of Your Work

In order to be able to reproduce your work and remember what you have done after some time, you can record both the results and the list of commands. To record the outputs that appear in the *Results* window, use **Log files**. Open a new log file by clicking on the respective icon in the Stata toolbar. Log files can be printed or saved in other file formats. When returning to a Stata session, you can also append a log file saved during a previous session.

Do-files can be used to record the set of commands that you have used during a session. Put simply, a do-file is a single file that lists a whole set of commands. This enables you to execute all commands in one go or only a selection of the commands. By saving the do-files you are able to easily replicate analyses done in previous Stata sessions. Do-files can be written in any text editor, such as Word or Notepad. It is good practice to keep extensive notes within your do-file so that when you look back over it you know what you were trying to achieve with each command or set of commands. We will be using do-files in later sections of this guide so will leave a more detailed description of do-files and the 'Do-file Editor' until then.

2.9 Saving Data and Results

Data generated in Stata can be exported to other Windows applications, e.g., Microsoft Excel, or to a data format used by other statistical software packages. To export data, click on **File/Export** and then choose the file type required. In the **'export'** dialogue window you can then specify the variables to export or you can restrict the sample to export. You also need to specify a file name and location for the new file.

You can export single outputs by highlighting them in the *Results* window, then right-clicking on the highlighted section and choosing the 'Copy' option required. For example, you can copy as a table (and paste the results into Excel) or you can copy as a picture.

Additionally, there are several other commands that can be used to save estimation results or to generate formatted tables of results. Some useful commands are, for example, **'estimates save'** or the user-written command **'outreg2'**.[11]

[11]The latter is a user-written ado-file that needs to be installed before you can use it. This can be easily done by typing **'findit outreg2'** into your command window and clicking on the link for outreg2. Once you have installed it, you can use it as if it was a built-in Stata function. It also has its own help description that you can access via the command **'help outreg2'**.

3 Linear Regression – Estimation of an Optimal Hedge Ratio

Reading: Brooks (2019, Section 3.3)

This section shows how to run a bivariate regression using Stata. In our example an investor wishes to hedge a long position in the S&P500 (or its constituent stocks) using a short position in futures contracts. The investor is interested in the optimal hedge ratio, i.e., the number of units of the futures asset to sell per unit of the spot assets held.[12]

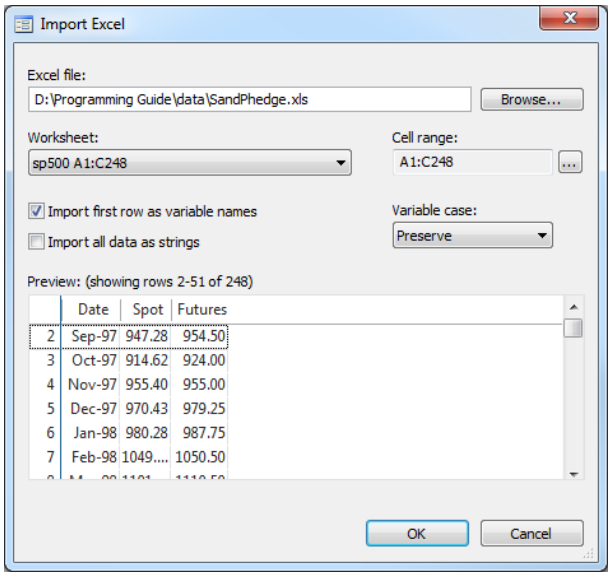


Figure 13: Importing Time-Series Data into Stata

This regression will be run using the file 'SandPhedge.xls', which contains monthly returns for the S&P500 Index (in Column 2) and the S&P500 Futures (in Column 3). Before we run the regression, we need to import this Excel workfile into Stata. For this, we click on **File** in the top left corner of the Stata main screen and select **Import** and **Excel spreadsheet (*.xls; *.xlsx)**. The **Import Excel** window appears (Figure 13). We click on **Browse...** and select the file 'SandPhedge.xls'. As the first row of the data contains the variable names, check the box **Import first row as variable names**. Stata provides a preview of the data contained in the file at the bottom of the window. Click **OK** in order to import the data into Stata.

We can see the three imported data series in the *Variables* window on the right-hand side of the screen. In order to examine the data and verify some data entries, you can click on the *Data Editor (Edit)* symbol in the toolbar. Alternatively, type the command **edit** into the *Command* window at the very bottom of the Stata screen.

Stata automatically identifies the series 'date' as a series containing dates. However, it codes the variable as a daily series instead of a monthly one. In order to change this, we click on **Data** in the top left corner and select **Create or change data/Change contents of variable**. A new dialogue window appears (Figure 14, left panel).

[12] See also Chapter 9 in Brooks (2019).

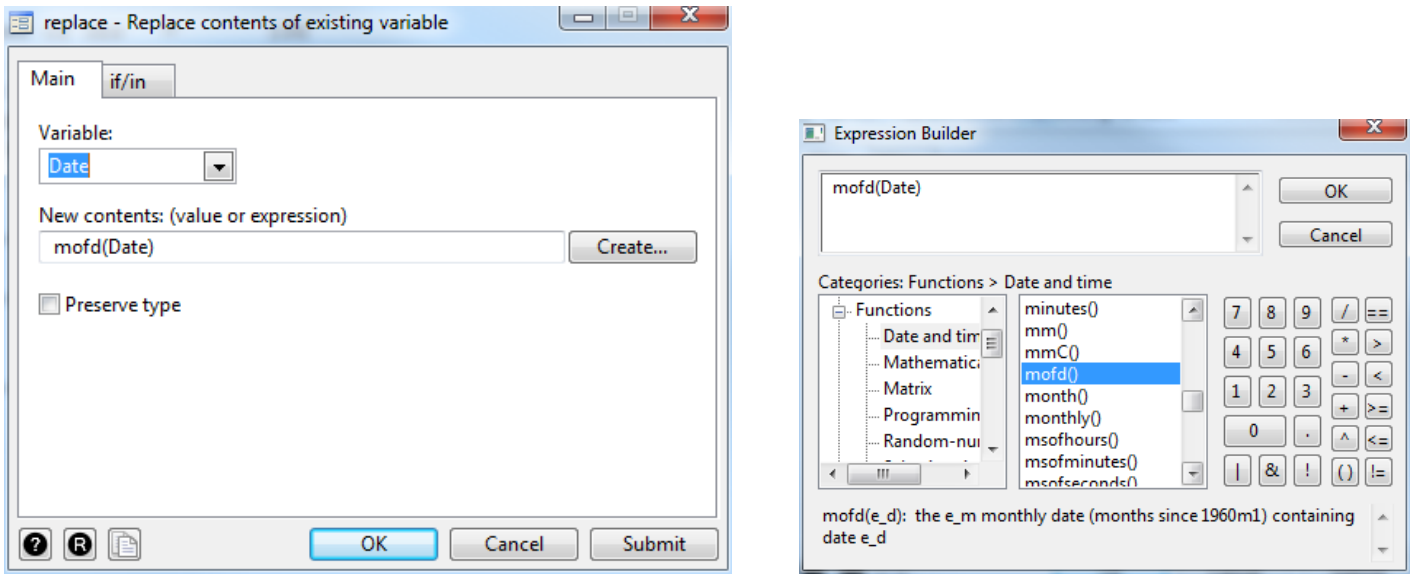


Figure 14: Changing the Unit of Date Variables

First, we select the variable to change, i.e., **Date**, in the dialog box **Variable**. Next, we click on the button **Create...** and a new window appears (Figure 14, right panel). In the **Expression builder** window we click on the 'plus' sign next to the option **Functions**. Since the variable is a time variable we select **Date and time** and scroll down the options on the right until we find the function **mofd()**.[13] We double-click on **mofd()** and the expression appears in the dialog box in the top of the window. We type **Date** in between the parentheses and click on **OK**. The expression now appears in the dialog box **New contents: (value or expression)**.[14] By clicking **OK** Stata will replace the daily variable 'Date' with its monthly equivalent.[15]

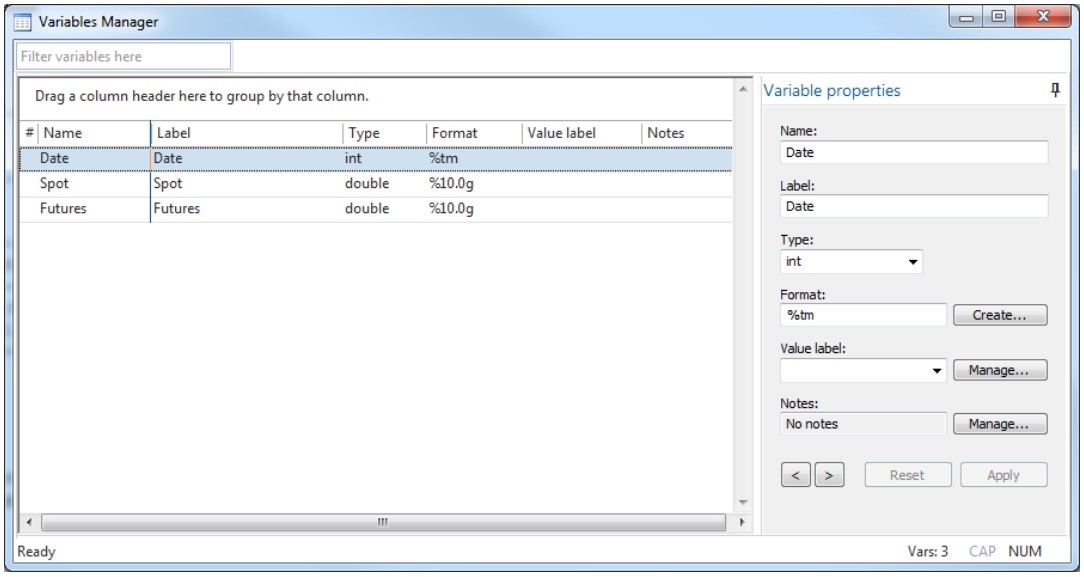


Figure 15: Variables Manager for S&P Dataset

[13] **mofd()** is an abbreviation for **m**onth **of** **d**ate and returns the monthly date of a variable that contains daily dates.

[14] Alternatively, if you know the function to use, you can directly input the expression in the dialogue box **New contents: (value or expression)**.

[15] The command for this data manipulation appears in the *Review* window on the left. By clicking on it, it appears in the *Command* window at the bottom of the Stata screen. As a (faster) alternative than using the 'point-or-click' menu to execute the data manipulation, we could have directly typed the command in the *Command* window.

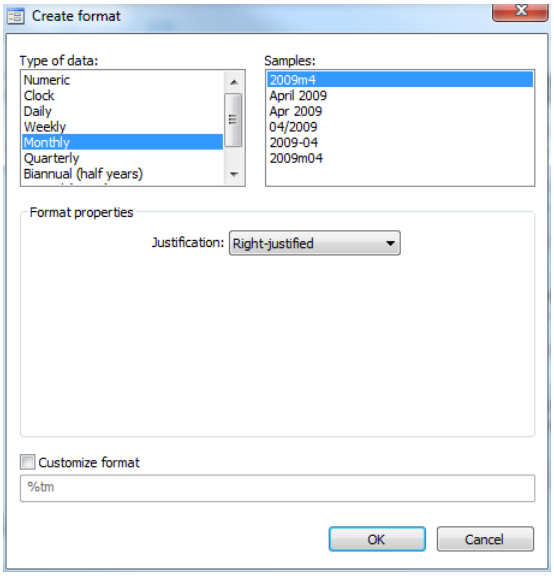


Figure 16: Variable Manager

In order to change the display format of the changed variable from a daily to a monthly format, we click **Data** in the top left menu and select *Variable Manager*. In the newly appearing window we select the variable **Date** and click the box **Create...** next to the **Format** dialog box on the left (Figure 15). A new window appears (Figure 16). We specify the **Type of data** to be **Monthly** and click **OK**.[16] To make these changes effective, we click on **Apply** at the bottom right of the window. Afterwards, we just close the *Variable Manager* window. In order to see the effect of these changes, change to the *Data Editor* by clicking on the respective symbol in toolbar or, alternatively, type in the command **edit** in the *Command* window.

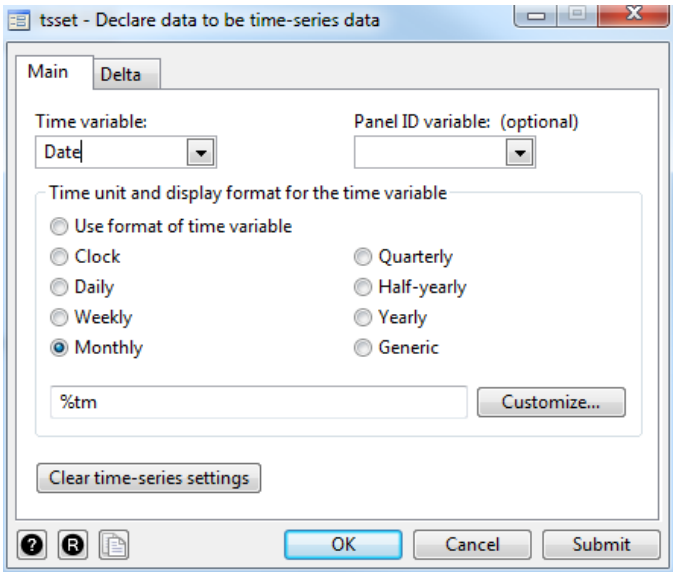


Figure 17: Declare Dataset to be Time-Series Data

As we want to perform time-series regressions, we need to tell Stata which of the three variables is the time indicator. To do so, we click on **Statistics** in the top right menu and select **Time Series/Setup**

[16] In the dialogue box **Samples**, we can choose between different ways of displaying the monthly data series. These options merely represent different display formats and do not change the characteristics of the underlying series. The choice of the display format depends on personal taste, so choose the format that is most appealing to you.

and Utilities/Declare dataset to be time-series data. In the dialogue box we select the time variable **Date** and further specify that the **Time unit and display format of the time variable** is **Monthly** (Figure 17). We can ignore the other options for now.

By clicking **OK**, the command together with the specifications of the time variable 'Date' appear in the *Output* window. 'delta: 1 month' indicates that the time variable is monthly data and that the difference between a data entry and the consecutive data entry is 1 month.[17]

We run our analysis based on returns of the S&P500 index instead of price levels; so the next step is to transform the 'spot' and 'future' price series into percentage returns. For our analysis we use continuously compounded returns, i.e., logarithmic returns, instead of simple returns as it is common in academic finance research. To generate a new data series of continuously compounded returns, we click on **Data** and select **Create or change data/Create new variable**. A new window appears where we can specify the new variable we want to create (Figure 18).

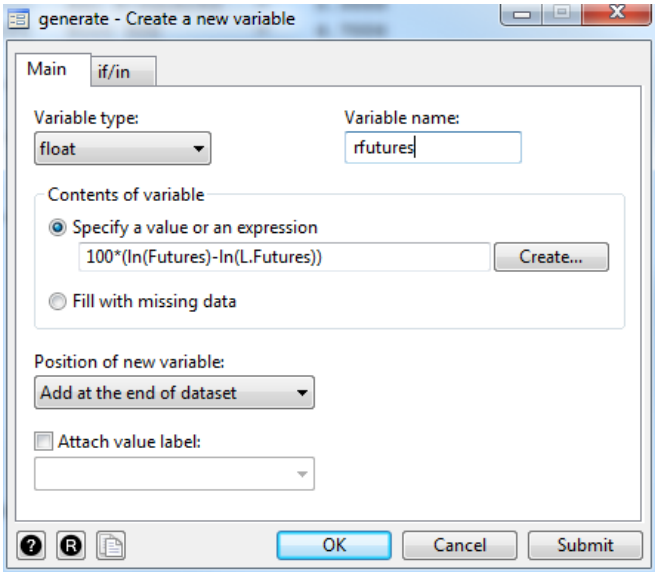


Figure 18: Computing Continuously Compounded Returns

We keep the default **Variable type** specification of **float** which specifies that the newly created variable is a data series of real numbers with about 8 digits.[18] We name the new variable **rfutures** in the **Variable name** dialogue box. Next, we enter **100*(ln(Futures)-ln(L.Futures))** into the dialog box that asks us to **Specify a value or an expression**. 'ln()' generates the natural logarithm of the data series and 'L.' represents a time operator indicating a 1-period lagged value of the variable 'future'.[19] To generate the new variable, we click on **OK**.

Next, we generate continuously compounded returns for the 'spot' price series. We can either use the Stata 'point-or-click' menu and repeat the steps described above with the 'spot' series; or, as a faster alternative, we can directly type in the command in the *Command* window. In the latter case, we type

generate rspot = 100*(ln(Spot)-ln(L.Spot))

[17]Note that if we had not changed 'Date' from a daily data series into a monthly data series the data unit would be indicated as '1 day'. This might lead to problems in case we want to execute time operators in further analyses.

[18]An accuracy of eight digits is sufficiently accurate for most work. However, if you need a higher accuracy of the data, e.g., as you are working with very small decimal number, you can change the variable type to double which creates a variable with an accuracy of 16 digits.

[19]In order to use the time operators, the time-series operator needs to be set using the **tsset** command described above. Time-series operators are a very useful Stata tool. For more information on time-series operators type in **help tsvarlist** in the **Command** box and corresponding chapter in the Data documentation.

and press **Enter** to execute the command. Do not forget to save the workfile selecting **File** and **Save as...**. Call the workfile 'hedge.dta'. Continue to re-save the file at regular intervals to ensure that no work is lost.

Before proceeding to estimate the regression, now that we have imported more than one series, we can examine a number of descriptive statistics together and measures of association between the series. For the summary statistics, we click on **Data** and **Describe data/Summary statistics**. In the dialogue box that appears, we type (or select from the drop-down menu) **rspot rfutures** and click **OK** (Figure 19).

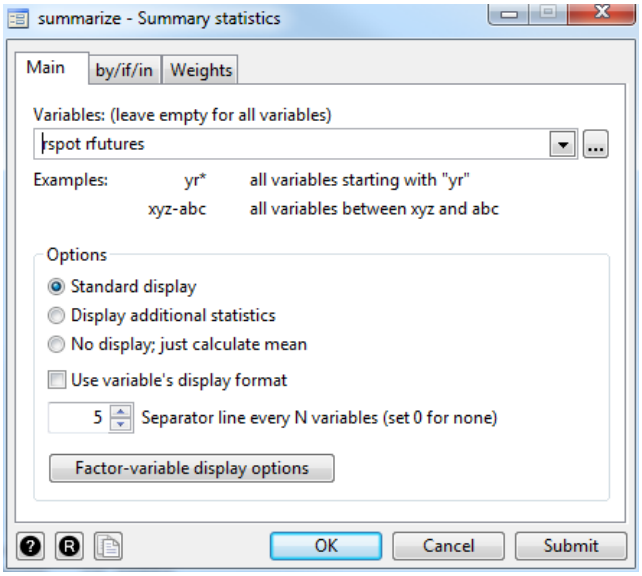


Figure 19: Generating Summary Statistics

The following summary statistics appear in the *Output* window:

```
. summarize rspot rfutures

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       rspot |        246     .416776    4.333323  -18.56365   10.23066
    rfutures |        246    .4140172    4.419049   -18.9447   10.38718

.
```

We observe that the two return series are quite similar as based on their mean values, standard deviations, as well as minimum and maximum values, as one would expect from economic theory.[20] Note that the number of observations has reduced from 247 for the levels of the series to 246 when we computed the returns (as one observation is 'lost' in constructing the $t-1$ value of the prices in the returns formula). If you want to save the summary statistics, you must either highlight the table, left-click, **(Copy table)** and paste it into an Excel spreadsheet; or you open a log file before undertaking the analysis which will record all command outputs displayed in the *Output* window. To start a log, we click on the **Log**

[20]Stata allows you to customise your summary statistics by choosing the type of statistics and their order. For this you can use the **tabstat** command. Click on **Statistics** and **Summaries, tables, and tests**. Select **Other tables** and **Compact table of summary statistics**. Specify the variables in the 'Variables' dialogue box. Next select and specify the number and type of **Statistics to display** by checking the box next to the statistic and selecting the statistic of choice from the drop-down menu. Once you have finalised your selection, click on **OK** and the summary statistics will appear in the *Output* window.

symbol (which resembles a notepad) in the menu and save it. Remember to close the log file when you have finished the analyses by clicking on the **Log** symbol again and selecting **Close log file**.

We can now proceed to estimate the regression. We click on **Statistics** and choose the second option in the list, **Linear models and related** and **Linear regression**. In the dialogue window entitled 'regress - Linear regression', we first specify the dependent variable (y), **rspot** (Figure 20).

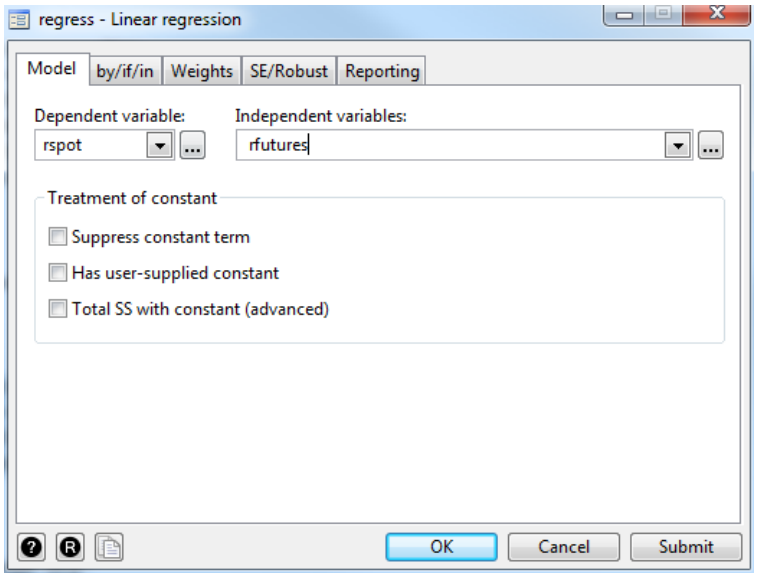


Figure 20: Estimating OLS Regressions

The independent variable is the **rfutures** series. Thus, we are trying to explain variations in the spot rates with variations in the futures rates. Stata automatically includes a constant term in the linear regression; therefore you do not specifically include it among the independent variables. If you do not wish to include a constant term in the regression you need to check the box **Suppress constant term**. The **regress** command estimates an OLS regression based on the whole sample. You can customise the regression specification, e.g., with respect to the selected sample or the treatment of standard errors, by using the other tabs. However, for now we stick with the default specification and press **OK** in order to run the regression. The regression results will appear in the *Output* window as follows:

```
. regress rspot rfutures

      Source |       SS           df       MS      Number of obs   =       246
-------------+----------------------------------   F(1, 244)       =  21474.92
       Model |  4548.8499         1   4548.8499     Prob > F        =    0.0000
    Residual |  51.6844364      244  .211821461     R-squared       =    0.9888
-------------+----------------------------------   Adj R-squared   =    0.9887
       Total |  4600.53433      245  18.7776912     Root MSE        =    .46024

------------------------------------------------------------------------------
       rspot |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    rfutures |   .9750771   .0066539   146.54   0.000     .9619708    .9881834
       _cons |   .0130773   .0294729     0.44   0.658    -.0449764    .0711311
------------------------------------------------------------------------------

.
```

The parameter estimates for the intercept ($\hat{\alpha}$) and slope ($\hat{\beta}$) are 0.013 and 0.975, respectively.[21] A large

[21] Note that in order to save the regression output you have to **Copy table** as described above for the summary statistics

number of other statistics are also presented in the regression output – the purpose and interpretation of these will be discussed later.

Now we estimate a regression for the levels of the series rather than the returns (i.e., we run a regression of 'spot' on a constant and 'future') and examine the parameter estimates. We can either follow the steps described above and specify 'spot' as the dependent variable and 'future' as the independent variable; or we can directly type the command into the *Command* window as

regress Spot Futures

and press **Enter** to run the regression (see below). The intercept estimate ($\hat{\alpha}$) in this regression is –2.838 and the slope estimate ($\hat{\beta}$) is 1.002.

```
. regress Spot Futures

      Source |       SS           df       MS      Number of obs   =       247
-------------+----------------------------------   F(1, 245)       >  99999.00
       Model |  47948435.2         1  47948435.2   Prob > F        =    0.0000
    Residual |  11692.7969       245  47.7257015   R-squared       =    0.9998
-------------+----------------------------------   Adj R-squared   =    0.9998
       Total |   47960128       246  194959.87     Root MSE        =    6.9084

------------------------------------------------------------------------------
        Spot |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     Futures |    1.001607   .0009993  1002.33   0.000     .9996383    1.003575
       _cons |   -2.837837   1.488972    -1.91   0.058    -5.770657    .0949827
------------------------------------------------------------------------------

. browse
```

Let us now turn to the (economic) interpretation of the parameter estimates from both regressions. The estimated return regression slope parameter measures the optimal hedge ratio as well as the short run relationship between the two series. By contrast, the slope parameter in a regression using the raw spot and futures indices (or the log of the spot series and the log of the futures series) can be interpreted as measuring the long run relationship between them. The intercept of the price level regression can be considered to approximate the cost of carry. Looking at the actual results, we find that the long-term relationship between spot and futures prices is almost 1:1 (as expected). Before exiting Stata, do not forget to click the **Save** button to save the whole workfile.

or remember to start a log file before undertaking the analysis. There are also other ways to export regression results. For more details on saving results, refer to Section 2.9 of this guide.

4 Hypothesis Testing – Example 1: Hedging Revisited

Reading: Brooks (2019, Sections 3.8 and 3.9)

Let us now have a closer look at the results table from the returns regressions in the previous section where we regressed S&P500 spot returns on futures returns in order to estimate the optimal hedge ratio for a long position in the S&P500. If you do not have the results ready on your Stata main screen, reload the **'SandPhedge.dta'** file now and re-estimate the returns regression using the steps described in the previous section (or, alternatively, type in **regress rspot rfutures** into the *Command* window and execute the command). While we have so far mainly focused on the coefficient estimates, i.e., α and β estimates, Stata has also calculated several other statistics which are presented next to the coefficient estimates: standard errors, the t-ratios, and the p-values.

The t-ratios are presented in the third column indicated by the 't' in the column heading. They are the test statistics for a test of the null hypothesis that the true values of the parameter estimates are zero against a two-sided alternative, i.e., they are either larger or smaller than zero. In mathematical terms, we can express this test with respect to our coefficient estimates as testing $H_0 : \alpha = 0$ versus $H_1 : \alpha \neq 0$ for the constant '_cons' in the second row of numbers and $H_0 : \beta = 0$ versus $H_1 : \beta \neq 0$ for 'rfutures' in the first row. Let us focus on the t-ratio for the α estimate first. We see that with a value of only 0.44 the t-ratio is very small which indicates that the corresponding null hypothesis $H_0 : \alpha = 0$ is likely not to be rejected. Turning to the slope estimate for 'rfutures', the t-ratio is high with 146.54 suggesting that $H_0 : \beta = 0$ is to be rejected against the alternative hypothesis of $H_1 : \beta \neq 0$. The p-values presented in the fourth column, '$P > |t|$', confirm our expectations: the p-value for the constant is considerably larger than 0.1 meaning that the corresponding t-statistic is not even significant at a 10% level; in comparison, the p-value for the slope coefficient is zero to, at least, three decimal points. Thus, the null hypothesis for the slope coefficient is rejected at the 1% level.

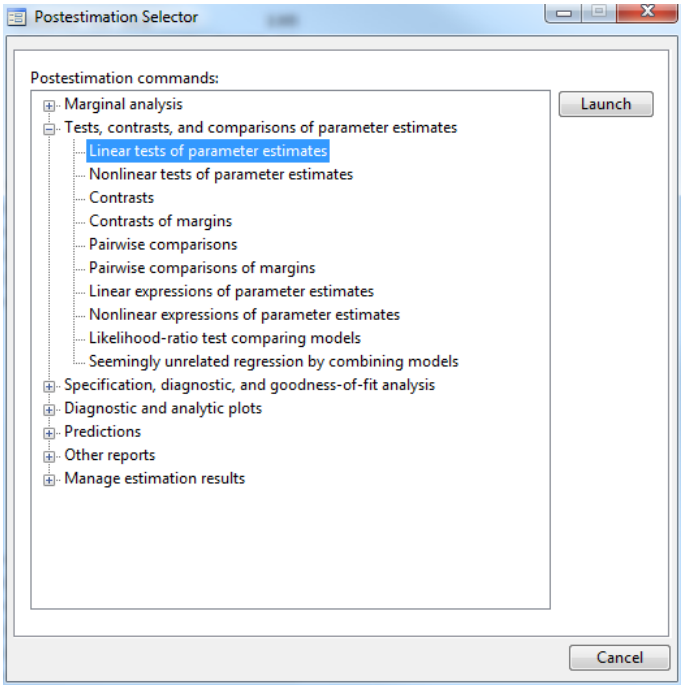


Figure 21: Postestimation Selector

While Stata automatically computes and reports the test statistics for the null hypothesis that the coefficient estimates are zero, we can also test other hypotheses about the values of these coefficient estimates. Suppose that we want to test the null hypothesis that $H_0 : \beta = 1$. We can, of course,

calculate the test statistics for this hypothesis test by hand; however, it is easier if we let Stata do this work. For this we use the Stata command 'test'. 'test' performs Wald tests of simple and composite linear hypotheses about the parameters of the most recently fit model.[22] To access the Wald test, we click on **Statistics** and select **Postestimation** (second option from the bottom). The new window entitled 'Postestimation Selector' appears (Figure 21). We select **Tests, contrasts, and comparisons of parameter estimates** and then chose the option **Linear tests of parameter estimates**.

The 'test' specification window should appear (Figure 22, left panel). We click on **Create...** next to **Specification 1** and a new window, entitled 'Specification 1' appears (Figure 22, right panel). In the box **Test type** we choose the second option **Linear expressions are equal** and we select **Coefficient: rfutures**. This is because we want to test a linear hypothesis that the coefficient estimate for 'rfutures' is equal to 1. Note that the first specification **Coefficients are 0** is equal to the test statistics for the null hypothesis $H_0 : \beta = 0$ versus $H_1 : \beta \neq 0$ for the case of 'rfutures' that we discussed above and that is automatically reported in the regression output.[23] Next, we specify the linear expression we would like to test in the dialog box **Linear expression**. In our case we type in **rfutures=1**.

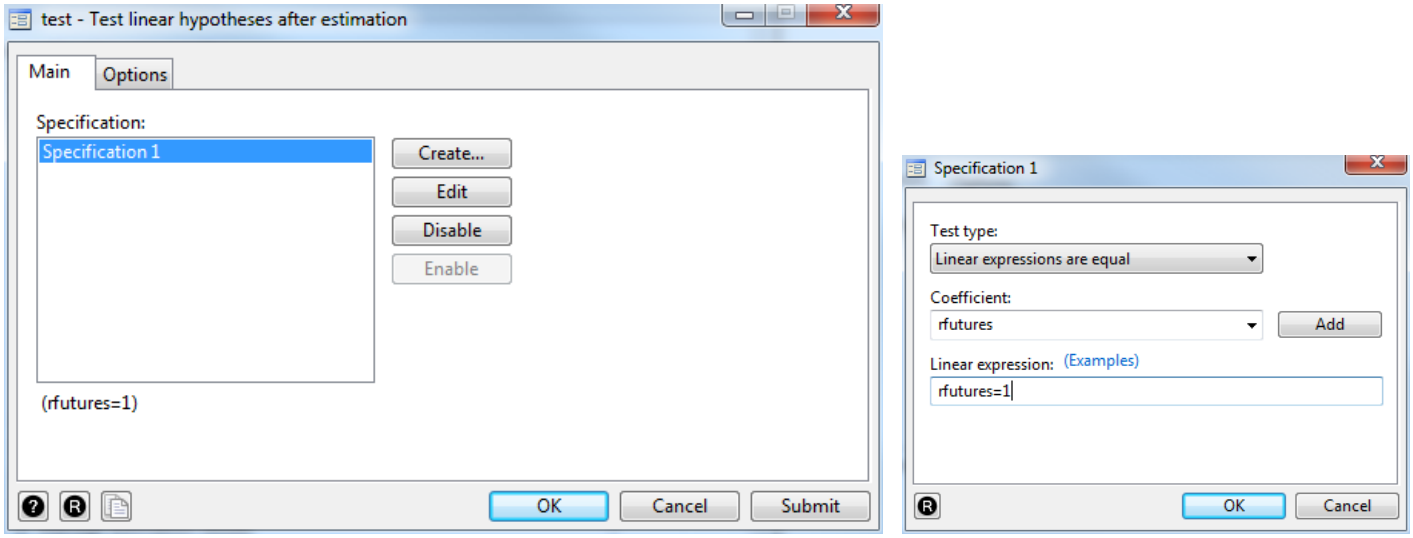


Figure 22: Specifying the Wald Linear Hypothesis Test

To generate the test statistic, we press **OK** and again **OK**. The Stata output should appear in the *Output* window as follows.

```
. test(rfutures =1)

 ( 1)  rfutures = 1

       F(  1,   244) =   14.03
            Prob > F =    0.0002

.
```

The first line 'test (rfutures=1)' repeats our test command and the second line'(1) rfutures = 1' reformulates it. Below we find the test statistics: 'F(1,244) = 14.03' states the value of the F-test with

[22] Thus, if you want to do a t-test based on the coefficient estimates of the return regression make sure that the return regression is the last regression you estimated (and not the regression on price levels).

[23] For more information on the 'test' command and the specific tests it can perform refer to the Stata Manual entry **test** or type in **help test** in the *Command* window.

one restriction and $(T-k) = 246-2 = 244$ which is 14.03. The corresponding p-value is 0.0002, stated in the last output line. As it is considerably smaller than 0.05 we can reject the null hypothesis that the coefficient estimate is equal to 1 at the 5% level. Note that Stata only presents one test statistic, the F-test statistic, and not as we might have expected the t-statistic. This is because the t-test is a special version of the F-test for single restrictions, i.e., one numerator degree of freedom. Thus, they will give the same conclusion and for brevity Stata only reports the F-test results.[24]

We can also perform hypothesis testing on the levels regressions. For this we re-estimate the regression in levels by typing **regress spot future** into the *Command* window and press **Enter**. Alternatively, we use the menu to access the 'regress' dialogue box, as described in the previous section. Again, we want to test the null hypothesis that $H_0 : \beta = 1$ on the coefficient estimate for 'Futures', so we can just type the command **test (future=1)** in the *Command* window and press **Enter** to generate the corresponding F-statistics for this test. Alternatively, you can follow the steps described above using the menu and the dialogue boxes. Both ways should generate the Stata output presented below.

```
. test(Futures = 1)

 ( 1)  Futures = 1

       F(  1,   245) =    2.58
            Prob > F =    0.1092

.
```

With an F-statistic of 2.58 and a corresponding p-value of 0.1092, we find that the null hypothesis is not rejected at the 5% significance level.

[24]For details on the relationship between the t- and the F-distribution, refer to Chapter 4 in Brooks (2019).

5 Hypothesis Testing – Example 2: The CAPM

Reading: Brooks (2019, Sections 3.10 and 3.11)

This exercise will estimate and test some hypotheses about the CAPM beta for several US stocks. The data for this example are contained in the excel file 'capm.xls'. We first import this data file into Stata by selecting **File/Import/Excel spreadsheet (*.xls; *.xlsx)**. As in the previous example, we first need to format the 'Date' variable. To do so, type the following sequence of commands in the *Command window*: first, change the Date type from daily to monthly with **replace Date=mofd(Date)**; then format the Date variable into human-readable format with **format Date %tm**; finally, define the time variable using **tsset Date**.[25] The imported data file contains monthly stock prices for the S&P500 index ('SANDP'), the four companies Ford ('FORD'), General Electric ('GE'), Microsoft ('MICROSOFT') and Oracle ('ORACLE'), as well as the 3-month US-Treasury bills ('USTB3M') from January 2002 until February 2018. You can check that Stata has imported the data correctly by checking the variables in the **Variables** window on the right of the Stata main screen and by typing in the command **codebook** which will provide information on the data content of the variables in your workfile.[26] Before proceeding to the estimation, save the Stata workfile as 'capm.dta' selecting **File/Save as...** .

It is standard in the academic literature to use five years of monthly data for estimating betas, but we will use all of the observations (over fifteen years) for now. In order to estimate a CAPM equation for the Ford stock, for example, we need to first transform the price series into (continuously compounded) returns and then to transform the returns into excess returns over the risk free rate. To generate continuously compounded returns for the S&P500 index, click **Data/Create or change data/Create new variable**. Specify the variable name to be **rsandp** and in the dialogue box **Specify a value or an expression** type in **100*(ln(SANDP/L.SANDP))**. Recall that the operator 'L.' is used to instruct Stata to use one-period lagged observations of the series. Once completed, the input in the dialogue box should resemble the input in Figure 23.

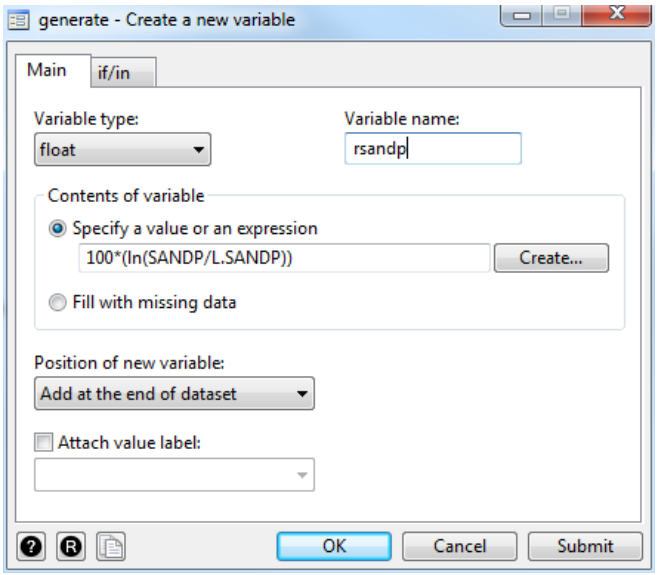


Figure 23: Generating Continuously Compounded Returns

By pressing **OK**, Stata creates a new data series named 'rsandp' that will contain continuously com-

[25] Alternatively, you can execute these changes using the menu by following the steps outlined at the beginning of the previous section.

[26] Alternatively, click **Data** in the Stata menu and select **Describe data/Describe data contents (codebook)** to access the command.

pounded returns of the S&P500. We need to repeat these steps for the stock prices of the four companies. To accomplish this, we can either follow the same process as described for the S&P500 index or we can directly type the commands in the *Command* window. For the latter, we type the command

generate rford=100*(ln(FORD/L.FORD))

into the *Command* window and press **Enter**. We should then find the new variable **rford** in the *Variable* window on the right.[27] Do the same for the remaining stock returns (except the 3-month Treasury bills, of course), resulting in the series **rford**, **rge** and **rmicrosoft**.

In order to transform the returns into excess returns, we need to deduct the risk free rate, in our case the 3-month US-Treasury bill rate, from the continuously compounded returns. However, we need to be slightly careful because the stock returns are monthly, whereas the Treasury bill yields are annualised. When estimating the model it is not important whether we use annualised or monthly rates; however, it is crucial that all series in the model are measured consistently, i.e., either all of them are monthly rates or all are annualised figures. We decide to transform the T-bill yields into monthly figures. To do so we use the command **replace USTB3M=USTB3M/12**. We can directly input this command in the *Command* window and press **Enter** to execute it; or we could use the Stata menu by selecting **Data/Create or change data/Change contents of variable** and specify the new variable content in the dialog box. Now that the risk free rate is a monthly rate, we can compute excess returns. For example, to generate the excess returns for the S&P500 we type **generate ersandp=rsandp-USTB3M** into the *Command* window and generate the new series by pressing **Enter**. We similarly generate excess returns for the four stock returns.

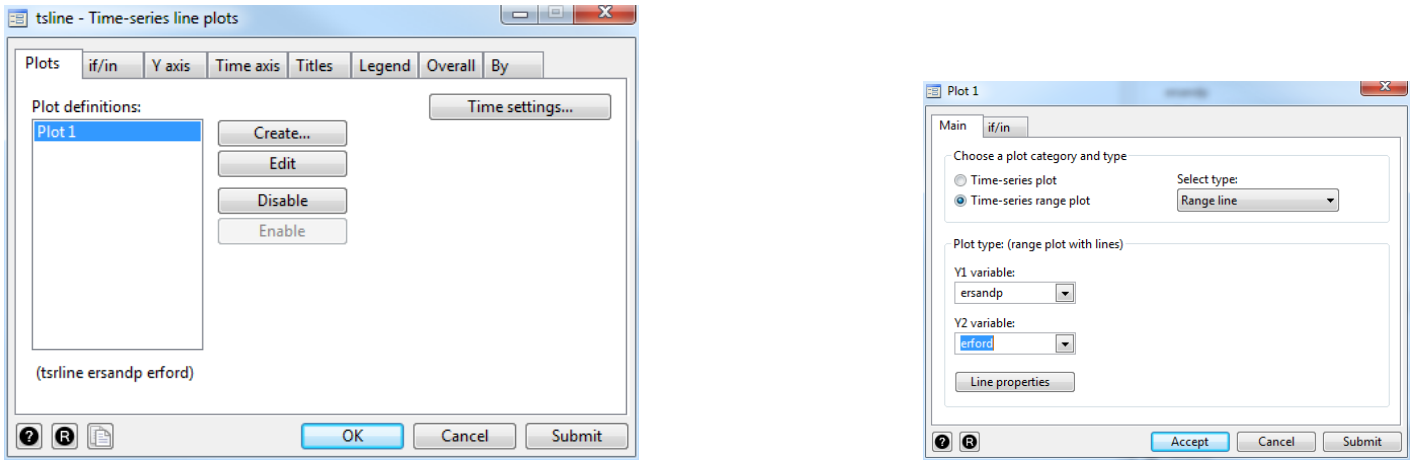


Figure 24: Generating a Time-Series Plot of Two Series

Before running the CAPM regression, we plot the data series to examine whether they appear to move together. We do this for the S&P500 and the Ford series. We click on **Graphics** in the Stata Menu and choose **Time-series graphs/Line plots** and a dialogue box appears (Figure 24, left panel). Click **Create...** and a new dialogue box appears (Figure 24, right panel). We first **Choose a plot category and type** to be **Time-series range plot**. We keep the default type **Range line** and select **ersandp** as **Y1 variable** and **erford** as **Y2 variable**. Then we close the dialogue box by clicking **Accept**. By selecting **OK** in the 'tsline' dialogue box, Stata generates the time-series plot of the two data series (Figure 25).[28]

[27] If you have misspecified the command, Stata will not execute the command but will issue an error message that you can read on the *Output* window. It usually provides further information as to the source of the error which should help you to detect and correct the misspecification.

[28] Note that the command for generating the plot appears in the **Review** window. In order to inspect the other data series in the data file we can simply click on the **tsrline** command and substitute the data series of our choice.

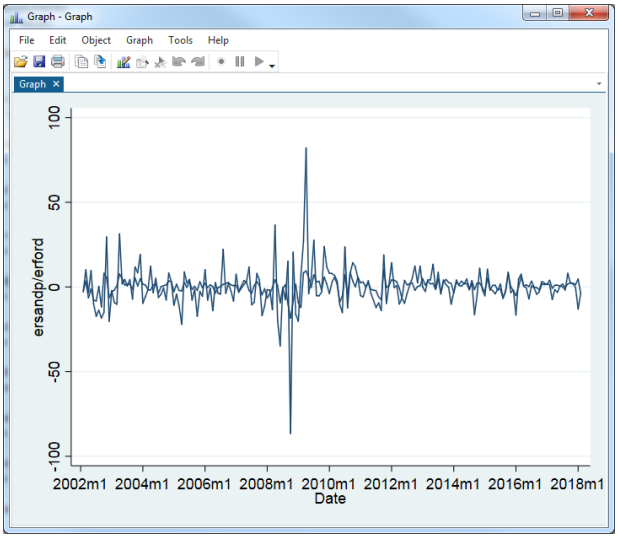
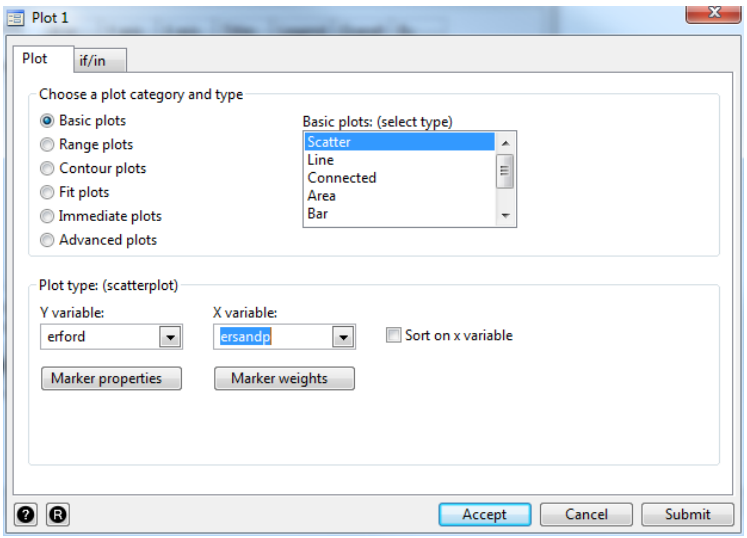


Figure 25: Time-Series Plot of Two Series

However, in order to get an idea about the association between two series a scatter plot might be more informative. To generate a scatter plot we first close the time-series plot and then we click on **Graphics** and select **Twoway graph (scatter, line, etc.)**. We select **Create...**. In the dialogue box that appears, we specify the following graph characteristics: We keep the default options **Basic plots** and **Scatter** and select **erford** as **Y variable** and **ersandp** as **X variable** (see the input in Figure 26, left panel).



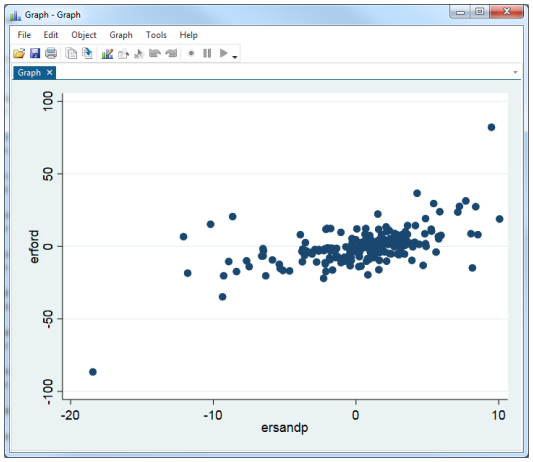


Figure 26: Generating a Scatter Plot of Two Series

We press **Accept** and **OK** and Stata generates a scatter plot of the excess S&P500 return and the excess Ford return as depicted in Figure 26, right panel. We see from this scatter plot that there appears to be a weak association between 'ersandp' and 'erford'. We can also create similar scatter plots for the other data series and the S&P500. Once finished, we just close the window of the graph.

To estimate the CAPM equation, we click on **Statistics** and then **Linear models and related** and **Linear regression** so that the familiar dialogue window 'regress - Linear regression' appears.

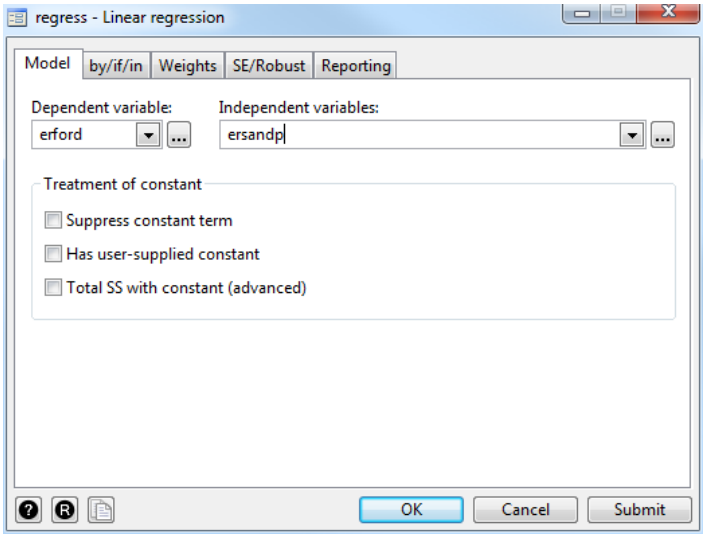


Figure 27: Estimating the CAPM Regression Equation

For the case of the Ford stock, the CAPM regression equation takes the form

$$(R_{Ford} - r_f)_t = \alpha + \beta(R_M - r_f)_t + u_t \tag{1}$$

Thus, the dependent variable (y) is the excess return of Ford 'rspot' and it is regressed on a constant as well as the excess market return 'ersandp'.[29] Once you have specified the variables, the dialogue window should resemble Figure 27. To estimate the equation press **OK**. The results appear in the **Output** window as below.

```
. regress erford ersandp

      Source |       SS       df       MS      Number of obs   =       193
-------------+------------------------------   F(1, 191)       =     97.26
       Model |  11721.7394     1  11721.7394   Prob > F        =    0.0000
    Residual |   23019.605   191  120.521492   R-squared       =    0.3374
-------------+------------------------------   Adj R-squared   =    0.3339
       Total |  34741.3443   192  180.944502   Root MSE        =    10.978

------------------------------------------------------------------------------
     erford |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    ersandp |    1.889755   .1916204     9.86   0.000     1.511791    2.267719
      _cons |   -.9559846   .7930851    -1.21   0.230    -2.520315    .6083456
------------------------------------------------------------------------------

.
```

Take a couple of minutes to examine the results of the regression. What is the slope coefficient estimate and what does it signify? Is this coefficient statistically significant? The beta coefficient (the slope coefficient) estimate is 1.89 with a t-ratio of 9.86 and a corresponding p-value of 0.000. This suggests that the excess return on the market proxy has highly significant explanatory power for the variability of the excess return of Ford stock.

Let us turn to the intercept now. What is the interpretation of the intercept estimate? Is it statistically significant? The α estimate is –0.96 with a t-ratio of –1.21 and a p-value of 0.230. Thus,

[29] Remember that the Stata command **regress** automatically includes a constant in the regression; thus, we do not need to manually include it among the independent variables.

we cannot reject that the α estimate is different from 0, indicating that the Ford stock does not seem to significantly outperform or underperform the overall market.

Assume we want to test that the value of the population coefficient on 'ersandp' is equal to 1. How can we achieve this? The answer is to click on **Statistics/Postestimation** to launch **Tests, contrasts, and comparisons of parameter estimates/Linear tests of parameter estimates** and then specify **Test type: Linear expressions are equal** and **Coefficient: ersandp**. Finally, we need to define the **linear expression: ersandp=1**.[30] By clicking **OK** the F-statistics for this hypothesis test appears in the *Output* window.

```
. test (ersandp=1)

 ( 1)  ersandp = 1

       F(  1,   191) =   21.56
            Prob > F =    0.0000

.
```

The F-statistic of 21.56 with a corresponding p-value of 0 (at least up to the fourth decimal point) implying that the null hypothesis that the CAPM beta of Ford stock is 1 is convincingly rejected and hence the estimated beta of 1.89 is significantly different from 1.[31]

[30] Alternatively, just type **test (ersandp=1)** into the *Command* window and press **Enter**.

[31] This is hardly surprising given the distance between 1 and 1.89. However, it is sometimes the case, especially if the sample size is quite small and this leads to large standard errors, that many different hypotheses will all result in non-rejection – for example, both $H_0 : \beta = 0$ and $H_0 : \beta = 1$ not rejected.

6 Sample Output for Multiple Hypothesis Tests

Reading: Brooks (2019, Section 4.4)

This example uses the **'capm.dta'** workfile constructed in the previous section. So in case you are starting a new session, re-load the Stata workfile and re-estimate the CAPM regression equation for the Ford stock.[32] If we examine the regression F-test (F-stat 97.26), this also shows that the regression slope coefficient is significantly different from zero, which in this case is exactly the same result as the t-test (t-stat 9.86) for the beta coefficient. Since in this instance there is only one slope coefficient, the F-test statistic is equal to the square of the slope t-ratio.

Now suppose that we wish to conduct a joint test that both the intercept and slope parameters are one. We would perform this test in a similar way to the test involving only one coefficient. First, we launch the 'Postestimation Selector' by selecting **Statistics/Postestimation**. To open the specification window for the Wald test we choose **Tests, contrasts, and comparisons of parameter estimates/Linear tests of parameter estimates**. We **Create...** the first restriction by selecting the option **Linear expressions are equal** and as linear expression we specify **ersandp=1**. We click **OK**. This is the first specification. In order to add the second specification, i.e., the coefficient on the constant is 1 as well, we click on **Create...** again. In the **Specification 2** dialogue box that appears we select the **Test type:Linear expressions are equal** and specify the following linear expression **_cons=1** in the dialog box. Once we have defined both specifications we click **OK** to generate the F-test statistics.[33]

```
. test (ersandp=1) (_cons=1)

 ( 1)  ersandp = 1
 ( 2)  _cons = 1

       F(  2,   191) =   12.94
            Prob > F =    0.0000

.
```

In the *Output* window above, Stata produces the familiar output for the F-test. However, we note that the joint hypothesis test is indicated by the two conditions that are stated, '(1) ersandp = 1' and in the next row '(2) _cons = 1'. Looking at the value of the F-statistic of 12.94 with a corresponding p-value of 0.0000, we conclude that the null hypothesis, $H_0 : \beta_1 = 1$ and $\beta_2 = 1$, is strongly rejected at the 1% significance level.

[32] To estimate the regression use the command **regress erford ersandp**.

[33] You can also execute this command using the command line. You would use the command **test (ersandp=1) (_cons=1)**. Note that it is important to set the parentheses around each of the terms as otherwise Stata will not execute the command and produce an error message.

7 Multiple Regression Using an APT-Style Model

Reading: Brooks (2019, Section 4.4)

The following example will show how we can extend the linear regression model introduced in the previous sections to estimate multiple regressions in Stata. In the spirit of arbitrage pricing theory (APT), we will examine regressions that seek to determine whether the monthly returns on Microsoft stock can be explained by reference to unexpected changes in a set of macroeconomic and financial variables. For this, we rely on the dataset **'macro.xls'** which contains 9 data series of financial and economic variables as well as a date variable spanning the time period from March 1986 until March 2018 (i.e., 385 monthly observations for each of the series). In particular, the set of financial and economic variables comprises the Microsoft stock price, the S&P500 index value, the consumer price index, an industrial production index, Treasury bill yields for three months and ten years, a measure of 'narrow' money supply, a consumer credit series, and a 'credit spread' series. The latter is defined as the difference in annualised average yields between a portfolio of bonds rated AAA and a portfolio of bonds rated BAA.

Before we can start with our analysis we need to import the dataset 'macro.xls' into Stata and adjust the 'Date' variable according to the steps outlined in previous sections.[34] Remember to **Save** the workfile as 'macro.dta'.

Now that we have prepared our dataset we can start with the actual analysis. The first stage is to generate a set of changes or differences for each of the variables, since the APT posits that the stock returns can be explained by reference to the unexpected changes in the macroeconomic variables rather than their levels. The unexpected value of a variable can be defined as the difference between the actual (realised) value of the variable and its expected value. The question then arises about how we believe that investors might have formed their expectations, and while there are many ways to construct measures of expectations, the easiest is to assume that investors have naive expectations that the next period value of the variable is equal to the current value. This being the case, the entire change in the variable from one period to the next is the unexpected change (because investors are assumed to expect no change).[35]

To transform the variables, we either use the Stata Menu (**(Data/Create or change data/Create new variable)**) or we directly type the commands into the *Command* window:

generate dspread=BMINUSA-L.BMINUSA

generate dcredit=CCREDIT-L.CCREDIT

generate dprod=INDPRO-L.INDPRO

generate rmsoft=100*(ln(MICROSOFT/L.MICROSOFT))

generate rsandp=100*(ln(SANDP/L.SANDP))

generate dmoney=M1SUPPLY-L.M1SUPPLY

generate inflation=100*(ln(CPI/L.CPI))

[34] Recall that we first need to transform the Date type from daily to monthly with **replace Date=mofd(Date)**. Then we format the Date variable into human-readable format with **format Date %tm**. Finally, we define the time variable using **tsset Date**.

[35] It is an interesting question as to whether the differences should be taken on the levels of the variables or their logarithms. If the former, we have absolute changes in the variables, whereas the latter would lead to proportionate changes. The choice between the two is essentially an empirical one, and this example assumes that the former is chosen, apart from for the stock price series themselves and the consumer price series.

generate term=USTB10Y-USTB3M

and press **Enter** to execute them. Next we need to apply further transformations to some of the transformed series, so we generate another set of variables:

generate dinflation=inflation-L.inflation

generate mustb3m=USTB3M/12

generate rterm=term-L.term

generate ermsoft=rmsoft-mustb3m

generate ersandp=rsandp-mustb3m

The final two of these calculate excess returns for the stock and for the index. We can now run the regression. To open the regression specification window we click on **Statistics/Linear models and related/Linear regression**. The variable whose behaviour we seek to explain is the excess return of the Microsoft stock, so we select **Dependent variable: ermsoft**. The explanatory variables are the excess market return (**ersandp**) as well as unexpected changes in: the industrial production (**dprod**), the consumer credit (**dcredit**), the inflation rate (**dinflation**), the money supply (**dmoney**), the credit spread (**dspread**), and the term spread (**rterm**). We type these variables into the **Independent variables** dialog box or select them from the drop-down menu so that the entry in the box should look like:

ersandp dprod dcredit dinflation dmoney dspread rterm

Note that you do not need to include a comma between the variables but only separate them using a blank. Also remember that we do not manually include the constant term as Stata automatically estimates the regression including a constant. Once you have included all variables just press **OK** and the regression results will be reported in the *Output* window, as follows.

```
. regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm

      Source |       SS           df       MS      Number of obs   =       383
-------------+----------------------------------   F(7, 375)       =     28.24
       Model |  12166.6858         7  1738.09797   Prob > F        =    0.0000
    Residual |  23078.1087       375  61.5416232   R-squared       =    0.3452
-------------+----------------------------------   Adj R-squared   =    0.3330
       Total |  35244.7945       382   92.26386   Root MSE        =    7.8448

------------------------------------------------------------------------------
     ermsoft |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     ersandp |    1.280799   .0943544    13.57   0.000     1.095269    1.466329
       dprod |   -.3030317   .7368811    -0.41   0.681    -1.751969    1.145905
     dcredit |   -.0253637    .027149    -0.93   0.351     -.078747    .0280196
   dinflation |     2.19467   1.264299     1.74   0.083    -.2913343    4.680675
      dmoney |   -.0068714   .0155679    -0.44   0.659    -.0374829     .02374
     dspread |    2.260065   4.140284     0.55   0.585    -5.881018    10.40115
       rterm |    4.733069   1.715814     2.76   0.006     1.359247    8.106892
       _cons |    1.326003   .4754806     2.79   0.006     .3910601    2.260945
------------------------------------------------------------------------------

.
```

Take a few minutes to examine the main regression results. Which of the variables has a statistically significant impact on the Microsoft excess returns? Using your knowledge of the effects of the financial

and macroeconomic environment on stock returns, examine whether the coefficients have their expected signs and whether the sizes of the parameters are plausible.

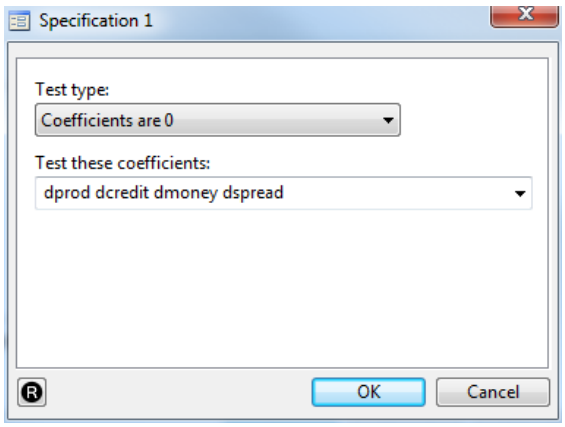


Figure 28: Multiple Hypothesis Test for APT-Style Model

The regression F-statistic (top right, second row) takes a value of 28.24. Remember that this tests the null hypothesis that all of the slope parameters are jointly zero. The p-value of zero attached to the test statistic shows that this null hypothesis should be rejected. However, there are a number of parameter estimates that are not significantly different from zero – specifically those on the 'dprod', 'dcredit', 'dmoney' and 'dspread' variables. Let us test the null hypothesis that the parameters on these four variables are jointly zero using an F-test. To test this, we click on **Statistics/Postestimation/Tests, contrasts, and comparisons of parameter estimates/Linear tests of parameter estimates** and then select **Create...**. As testing the hypothesis that the coefficients are (jointly) zero is one of the most common tests, there is a pre-defined option for this test available in Stata, namely the test type **Coefficients are 0** , as shown in Figure 28. We select this option and now all we need to do is specify in the box at the bottom of the window the variable names we want to perform the test on **dprod dcredit dinflation dmoney dspread** and press **OK** two times. We can now view the results of this F-test in the *Output* window:

```
. test(dprod dcredit dmoney dspread )

 ( 1)  dprod = 0
 ( 2)  dcredit = 0
 ( 3)  dmoney = 0
 ( 4)  dspread = 0

       F(  4,   375) =    0.41
            Prob > F =    0.7986

.
```

The resulting F-test statistic follows an $F(4, 375)$ distribution as there are 4 restrictions, 383 usable observations and 8 parameters to estimate in the unrestricted regression. The F-statistic value is 0.41 with p-value 0.7986, suggesting that the null hypothesis cannot be rejected. The parameters on **'rterm'** and **'dinflation'** are significant at the 10% level. Hence they are not included in this F-test and the variables are retained.

7.1 Stepwise Regression

There are a number of different stepwise regression procedures, but the simplest is the uni-directional forwards method. This starts with no variables in the regression (or only those variables that are always required by the researcher to be in the regression) and then it selects first the variable with the lowest p-value (largest t-ratio) if it were included, then the variable with the second lowest p-value conditional upon the first variable already being included, and so on. The procedure continues until the next lowest p-value relative to those already included variables is larger than some specified threshold value, then the selection stops, with no more variables being incorporated into the model.

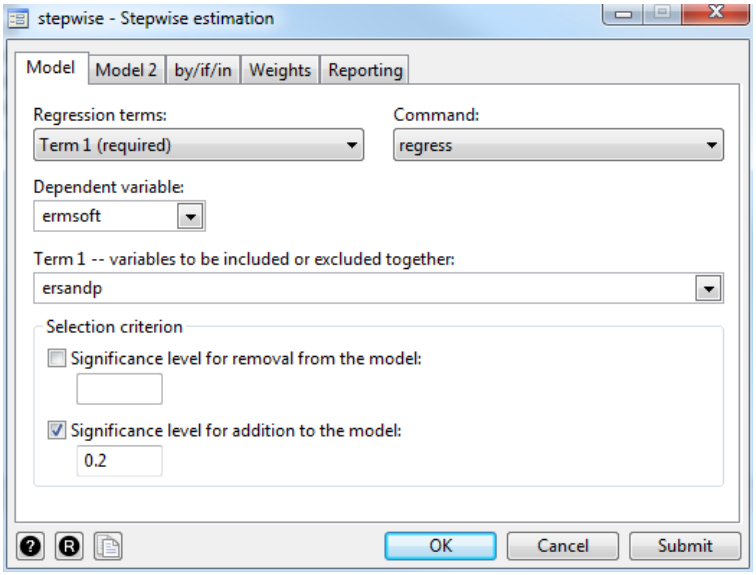


Figure 29: Stepwise Procedure Equation Estimation Window

We want to conduct a stepwise regression which will automatically select the most important variables for explaining the variations in Microsoft stock returns. We click **Statistics/Other** and select **Stepwise estimation**. A new dialog window appears (Figure 29). In Stata you need to specify each variable that you want to stepwise-include as a separate term. We start with the first variable we want to include which is 'ersandp'. We keep the default option of **Regression terms: Term 1 (required)**. Next to this box we are asked to specify the regression **Command** that we want to perform the stepwise estimation with and we select **regress**, i.e., a simple linear regression. Next, we select our **Dependent Variable: ermsoft** as well as the first term to be included which is **ersandp**. Finally, we are asked to select a significance level for removal from or addition to the model. We specify that we only want to add a variable to a model if it is significant at least at the 20% level so we check the box next to **Significance level for addition to the model** and input **0.2** in the text box below. To specify the second variable that should be included we click on the drop-down menu in the **Regression terms** box in the top left corner and select **Term 2**. We keep the command **regress** as well as the dependent variable **ermsoft** and the **Significance level for addition to the model: 0.2** and only change the **Term 2** to **dprod**. We keep including terms in this way until we have included all seven terms. Once we have done so, we press **OK** to execute the command. The results are as below:

```
. stepwise, pe(0.2) :  regress ermsoft (ersandp) (dprod) (dcredit) (dinflation) (dmoney) (dspread) (rterm)
                     begin with empty model
p = 0.0000 <  0.2000  adding  ersandp
p = 0.0047 <  0.2000  adding  rterm
p = 0.0710 <  0.2000  adding  dinflation

      Source |       SS           df       MS      Number of obs   =       383
-------------+----------------------------------   F(3, 379)       =     65.75
       Model |  12064.8028         3  4021.60092   Prob > F        =    0.0000
    Residual |  23179.9917       379  61.1609281   R-squared       =    0.3423
-------------+----------------------------------   Adj R-squared   =    0.3371
       Total |  35244.7945       382    92.26386   Root MSE        =    7.8205

------------------------------------------------------------------------------
     ermsoft |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     ersandp |   1.266282   .092096    13.75   0.000     1.085199    1.447365
       rterm |   4.738802   1.70873     2.77   0.006     1.379024    8.098581
  dinflation |   2.187422   1.208189    1.81   0.071    -.1881702    4.563015
       _cons |   1.020893    .40097     2.55   0.011     .2324882    1.809297
------------------------------------------------------------------------------

.
```

Note that a stepwise regression can be executed in different ways, e.g., either 'forward' or 'backward'. 'Forward' will start with the list of required regressors (the intercept only in this case) and will sequentially add to them, while 'backward' will start by including all of the variables and will sequentially delete variables from the regression. The way we have specified our stepwise estimation we perform a 'forward'-selection estimation and only add a variable to the model if it is significant at the 20% significance level, or higher.[36]

Turning to the results of the stepwise estimation, we find that the excess market return, the term structure, and unexpected inflation variables have been included, while the money supply, default spread and credit variables have been omitted.

[36]We will not perform a backward-selection estimation. For details on the backward specification, refer to the chapter on **stepwise** in the Stata Manual.

8 Quantile Regression

Brooks (2019, Section 4.10)

To illustrate how to run quantile regressions using Stata, we will now employ the simple CAPM beta estimation conducted in a previous section. We **re-open the 'capm.dta'** workfile. We select **Non-parametric analysis/Quantile regression** in the **Statistics** menu to open the quantile regression specification window. We select **erford** as the **Dependent Variable** and **ersandp** as the **Independent Variable** (Figure 30).

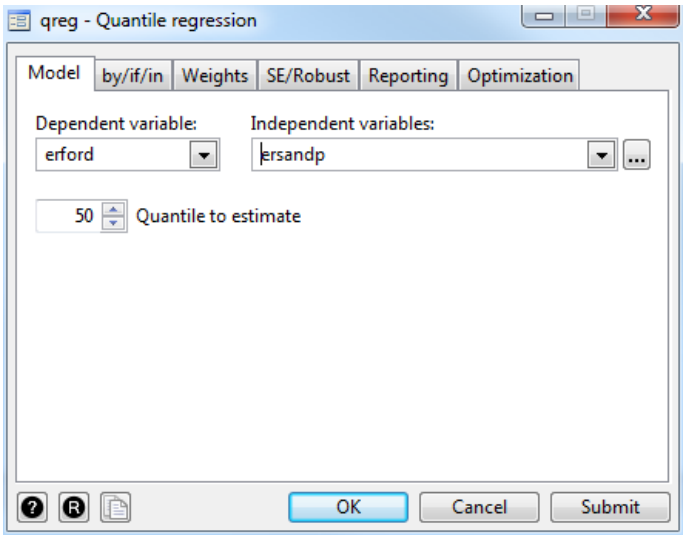


Figure 30: Quantile Regression Specification Window

As usual in Stata, we do not need to specify the constant as Stata will automatically include a constant term. Finally, we can choose the **Quantile to estimate**. The default option is **50** which is the median, but any integer values between 1 and 100 can be chosen. We can further customise the quantile regressions using the different tabs but we will stick with the default settings and press **OK** to run the regression. The output will appear as follows.

```
. qreg erford ersandp, quantile(50)
Iteration  1:  WLS sum of weighted deviations =  681.25001

Iteration  1: sum of abs. weighted deviations =  681.13656
Iteration  2: sum of abs. weighted deviations =  668.74505
Iteration  3: sum of abs. weighted deviations =  668.54542
Iteration  4: sum of abs. weighted deviations =  667.85944
Iteration  5: sum of abs. weighted deviations =   667.7999
Iteration  6: sum of abs. weighted deviations =  667.78657
Iteration  7: sum of abs. weighted deviations =  667.77929

Median regression                               Number of obs =        193
  Raw sum of deviations 806.8401 (about -1.0619814)
  Min sum of deviations 667.7793                Pseudo R2     =     0.1724
```

erford	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
ersandp	1.438457	.1544837	9.31	0.000	1.133744	1.743171
_cons	-1.489679	.6393825	-2.33	0.021	-2.750837	-.2285213

```
.
```

While this command only provides estimates for one particular quantile, we might be interested in differences in the estimates across quantiles. Next, we generate estimates for a set of quantiles. To run simultaneous quantile estimations, we click on **Statistics/Nonparametric analysis** and select **Simultaneous-quantile regression**.

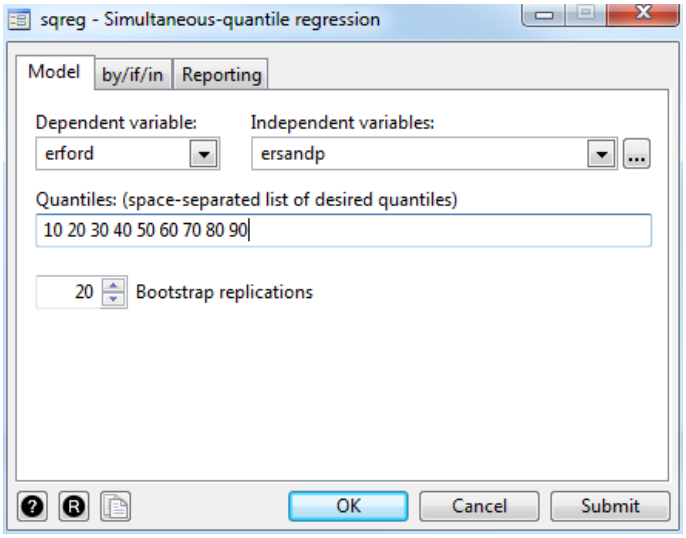


Figure 31: Specifying Simultaneous Quantile Regressions

In the regression specification window that appears we specify the regression parameters (**Dependent variable: erford ; Independent variables: ersandp**) as well as the list of quantiles that we want to simultaneously estimate (Figure 31). Let us assume we would like to generate estimates for 10 (evenly-spaced) quantiles. Thus, we specify the following set of quantiles, separated by spaces: **10 20 30 40 50 60 70 80 90** and click **OK** in order to generate the estimation output on the next page.

```
. sqreg erford ersandp, quantiles(10 20 30 40 50 60 70 80 90) reps(20)
(fitting base model)

Bootstrap replications (20)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
....................
```

Simultaneous quantile regression — Number of obs = 193

bootstrap(20) SEs

.10 Pseudo R2 =	0.2035
.20 Pseudo R2 =	0.2008
.30 Pseudo R2 =	0.1923
.40 Pseudo R2 =	0.1753
.50 Pseudo R2 =	0.1724
.60 Pseudo R2 =	0.1679
.70 Pseudo R2 =	0.1611
.80 Pseudo R2 =	0.1357
.90 Pseudo R2 =	0.1062

erford	Coef.	Bootstrap Std. Err.	t	P>|t|	[95% Conf.	Interval]
q10						
ersandp	2.34042	.6979466	3.35	0.001	.9637467	3.717093
_cons	-11.92905	1.646491	-7.25	0.000	-15.17669	-8.681407
q20						
ersandp	1.804414	.3731061	4.84	0.000	1.068477	2.540352
_cons	-7.3496	.9776906	-7.52	0.000	-9.278058	-5.421143
q30						
ersandp	1.659638	.2334878	7.11	0.000	1.199092	2.120184
_cons	-4.878333	.6404462	-7.62	0.000	-6.141589	-3.615077
q40						
ersandp	1.500533	.2498469	6.01	0.000	1.00772	1.993347
_cons	-3.347897	.7414417	-4.52	0.000	-4.810363	-1.885432
q50						
ersandp	1.438457	.2134461	6.74	0.000	1.017443	1.859472
_cons	-1.489679	.4887637	-3.05	0.003	-2.453747	-.5256112
q60						
ersandp	1.296706	.2493576	5.20	0.000	.8048571	1.788554
_cons	-.0188444	.5376496	-0.04	0.972	-1.079338	1.041649
q70						
ersandp	1.528341	.3076293	4.97	0.000	.9215543	2.135128
_cons	1.756783	.6461862	2.72	0.007	.4822053	3.031361
q80						
ersandp	1.619863	.3941284	4.11	0.000	.8424592	2.397266
_cons	4.001335	1.332711	3.00	0.003	1.372612	6.630058
q90						
ersandp	1.847333	.43191	4.28	0.000	.9954066	2.699259
_cons	10.45627	1.832884	5.70	0.000	6.840977	14.07156

.

For each quantile (**q10 to q90**), Stata reports two estimates together with their respective test statistics: the β-coefficient on 'ersandp' and the coefficient for the constant term. Take some time to examine and compare the coefficient estimates across quantiles. What do you observe? We find a monotonic rise in the intercept coefficients as the quantiles increase. This is to be expected since the data on y have been arranged that way. But the slope estimates are very revealing – they show that the beta estimate is much higher in the lower tail than in the rest of the distribution of ordered data. Thus the relationship

between excess returns on Ford stock and those of the S&P500 is much stronger when Ford share prices are falling most sharply. This is worrying, for it shows that the 'tail systematic risk' of the stock is greater than for the distribution as a whole. This is related to the observation that when stock prices fall, they tend to all fall at the same time, and thus the benefits of diversification that would be expected from examining only a standard regression of y on x could be much overstated.

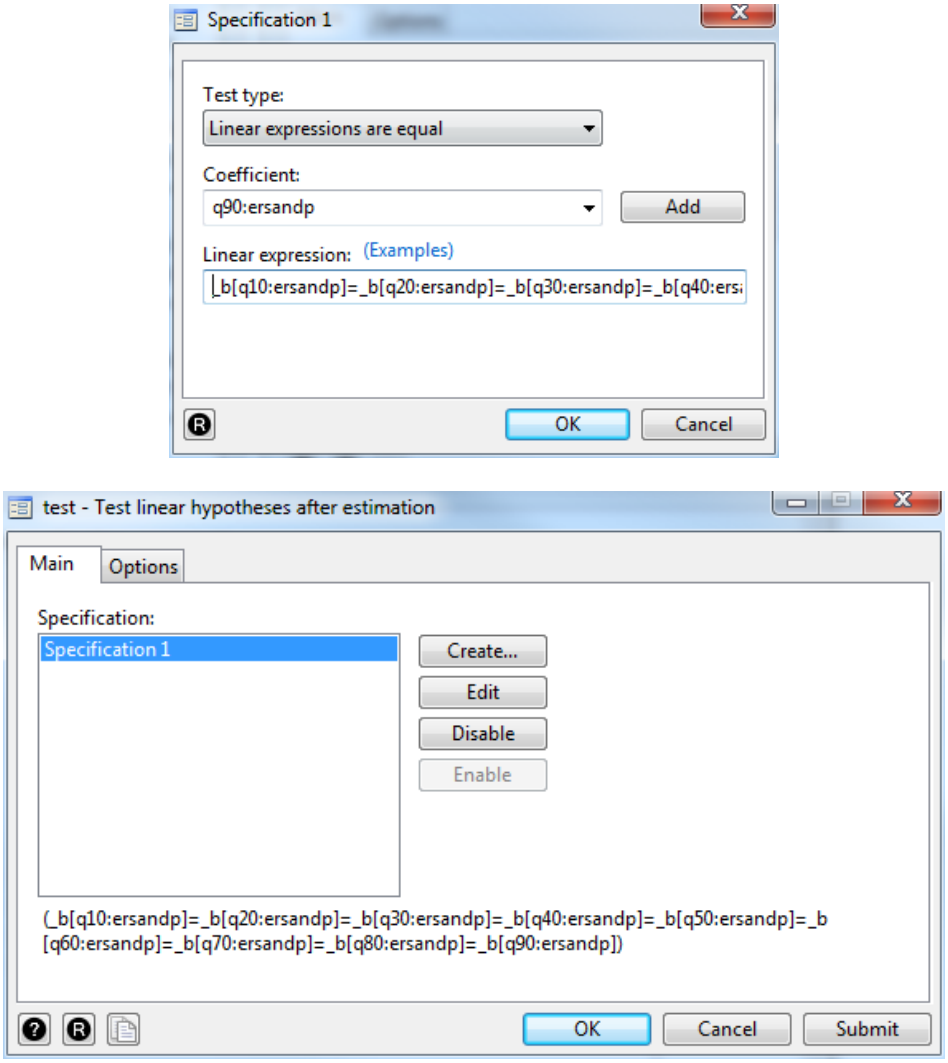


Figure 32: Equality of Quantile Estimates

Several diagnostics and specification tests for quantile regressions may be computed, and one of particular interest is whether the coefficients for each quantile can be restricted to be the same. To perform the equality test we rely on the **test** command that we have used in previous sections for testing linear hypotheses. It can be accessed via **Statistics/Postestimation/Tests, contrasts, and comparisons of parameter estimates/Linear tests of parameter estimates**. Note that it is important that the last estimation that you performed was the simultaneous quantile regression as Stata always performs hypothesis tests based on the most recent estimates. In the **test** specification window we click **Create...** and choose the test type **Linear expressions are equal** (Figure 32, upper panel). We click on the drop-down menu for **Coefficient** and select **q10:ersandp**, where [**q10**] indicates the coefficient estimate from the regression based on the 10th quantile. Then we press **Add** and the coefficient estimate for 'q10:ersandp' appears in the 'Linear expression' box at the bottom. Next we select **q20:ersandp** and also **Add** this variable to the 'Linear expression'. Note that Stata automatically adds an 'equal' sign between the two parameters. We keep doing this until we have included all 9 quantile estimates for

'ersandp' in the 'Linear expression'. Then we click **OK** and we see that expression appears at the bottom of the 'test' specification window (Figure 32, lower panel). Clicking **OK** again generates the following test results.

```
. test (_b[q10:ersandp] = _b[q20:ersandp] = _b[q30:ersandp] = _b[q40:ersandp] = _b[q50:ersandp]
> = _b[q60:ersandp] = _b[q70:ersandp] = _b[q80:ersandp] = _b[q90:ersandp])

 ( 1)  [q10]ersandp - [q20]ersandp = 0
 ( 2)  [q10]ersandp - [q30]ersandp = 0
 ( 3)  [q10]ersandp - [q40]ersandp = 0
 ( 4)  [q10]ersandp - [q50]ersandp = 0
 ( 5)  [q10]ersandp - [q60]ersandp = 0
 ( 6)  [q10]ersandp - [q70]ersandp = 0
 ( 7)  [q10]ersandp - [q80]ersandp = 0
 ( 8)  [q10]ersandp - [q90]ersandp = 0

       F(  8,   191) =    1.46
            Prob > F =    0.1727

.
```

We see that Stata has rearranged our initial test equation to express it in a way that makes it easier for the program to execute the command. The rearrangement is innocuous and, in fact, allows Stata to perform fairly complicated algebraic restrictions. In our case, we see that the test whether all coefficients are equal is the same as testing whether the difference between the coefficient on 'ersand' for quantile 10 and the respective coefficient for each of the other quantiles is zero.[37] Turning to the test-statistic we find that the F-value is 1.46 with a p-value of 0.1727.[38] In other words, we cannot reject the hypothesis that all coefficient estimates are equal, although the F-statistic is close to being significant at the 10% level. If we would have found that the coefficient estimates across quantiles was not equal, this would have implied that the association between the excess returns of Ford stock and the S&P500 index would vary depending on the part of the return distribution we are looking at, i.e., whether we are looking at very negative or very positive excess returns.

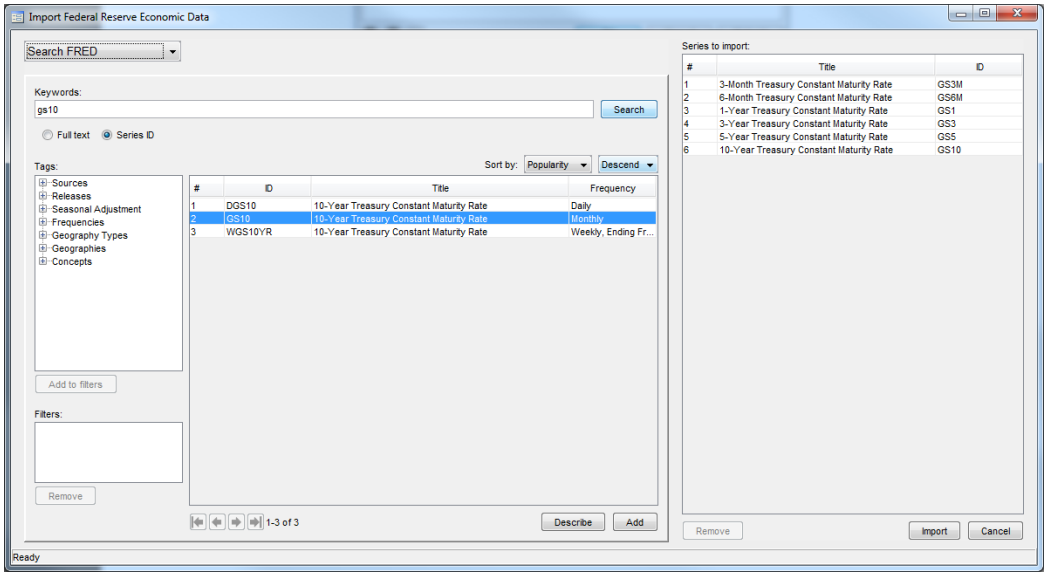
[37] To see that this is the case let us do some simple rearrangements. If all coefficients are equal then it has to be the case that individual pairs of the set of coefficients are equal to one another, i.e., [q10]ersandp=[q20]ersandp and [q10]ersandp=[q30]ersandp, etc. However, if the previous two equations are true, i.e., both [q20]ersandp and [q30]ersandp are equal to [q10]ersandp, then this implies that [q20]ersandp=[q30]ersandp. Thus, in order to test the equality of all coefficients to each other it is sufficient to test that all coefficients are equal to one specific coefficient, e.g., [q10]ersand. Now, we can rearrange pairwise equalities by expression them as differences as [q10]ersandp=[q20]ersandp is the same as writing [q10]ersandp-[q20]ersandp=0.

[38] Remember that this statistic is based on a bootstrap and is hence subject to randomness. To make your results replicable, type **set seed 1** into the command Window before you run the regression.

9 Calculating Principal Components

Reading: Brooks (2019, Appendix 4.2)

In this section we will examine a set of interest rates of different maturities and calculate the principal components for this set of variables in Stata. We can import the Treasury bill data directly from the FRED data base. If we click **File/Import/Federal Reserve Economic Data (FRED)**, a panel will open to choose the variables we are interested in (Figure 33, upper panel). By searching for 'GS3M', we can add the 3-month Treasury Constant Maturity Rate to the series to import. Afterwards, we do the same for the other maturities, GS6M, GS1, GS3, GS5, GS10, and click **Import**. This will open a window (Figure 33, lower panel) to specify the data range and aggregation method. We set the data range to '1982-01-01' to '2018-05-31' and use the monthly frequency with aggregation method 'End of Period'. After clicking **OK**, we can find the six series in the Variables window.[39].



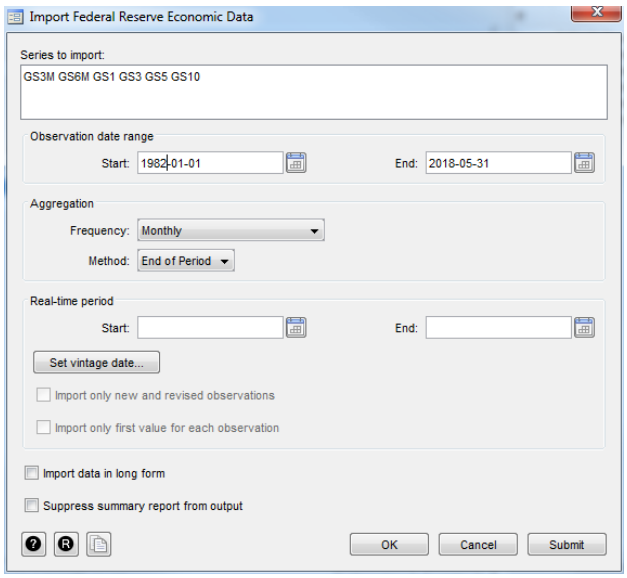


Figure 33: Import Variables from the FRED Data Base

[39]FRED also delivers two Variables for the dates (datestr and daten). They contain the underlying time-series in string and numerical format. We do not need these for the remaining part of this exercise.

After **saving** the workfile as **'fred.dta'**, we click on **Statistics** in the Stata menu and select **Multivariate analysis/Factor and principal component analysis/Principal component analysis (PCA)**. A new window appears where we are asked to input the **Variables** for which we want to generate principal components (Figure 34). We type in the six treasury bill rates **GS3M GS6M GS1 GS3 GS5 GS10** and click **OK**.

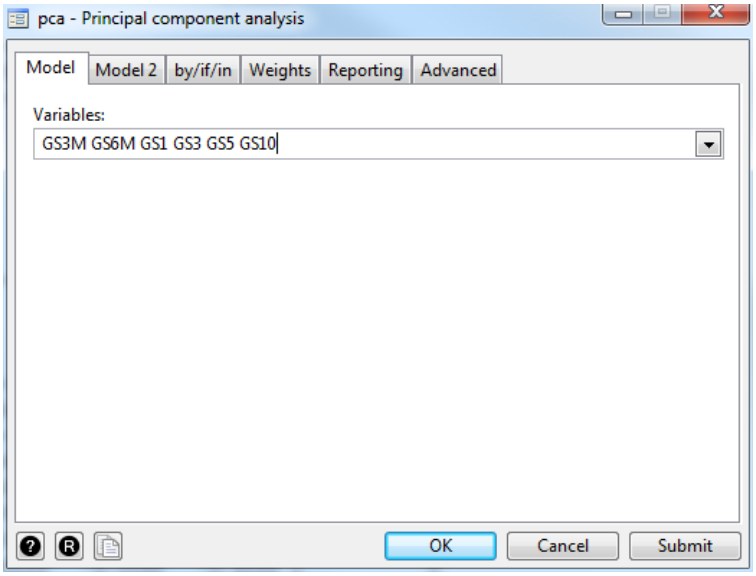


Figure 34: Principal Component Analysis Specification Window

Note that there are multiple ways to customise the principal component analysis using the options in the tabs. However, we keep the default settings for now. The results are presented in the *Output* window, as shown below. The first panel lists the eigenvalues of the correlation matrix, ordered from largest to smallest; the second panel reports the corresponding eigenvectors.

```
. pca GS3M GS6M GS1 GS3 GS5 GS10

Principal components/correlation                  Number of obs    =        437
                                                  Number of comp.  =          6
                                                  Trace            =          6
    Rotation: (unrotated = principal)             Rho              =     1.0000
```

Component	Eigenvalue	Difference	Proportion	Cumulative
Comp1	5.89401	5.79491	0.9823	0.9823
Comp2	.0991056	.0939144	0.0165	0.9989
Comp3	.00519112	.00378882	0.0009	0.9997
Comp4	.0014023	.0012011	0.0002	1.0000
Comp5	.000201204	.000112434	0.0000	1.0000
Comp6	.0000887703	.	0.0000	1.0000

Principal components (eigenvectors)

Variable	Comp1	Comp2	Comp3	Comp4	Comp5	Comp6	Unexplained
GS3M	0.4071	-0.4634	0.5278	-0.4858	0.2787	-0.1651	0
GS6M	0.4087	-0.3932	0.0737	0.2948	-0.5816	0.4977	0
GS1	0.4103	-0.2647	-0.3401	0.5499	0.2488	-0.5307	0
GS3	0.4113	0.1201	-0.5447	-0.2901	0.4243	0.5055	0
GS5	0.4090	0.3666	-0.2116	-0.4126	-0.5548	-0.4188	0
GS10	0.4031	0.6417	0.5088	0.3467	0.1850	0.1115	0

.

It is evident that there is a great deal of common variation in the series, since the first principal component captures over 98% of the variation in the series and the first two components capture 99.9%. Consequently, if we wished, we could reduce the dimensionality of the system by using two components rather than the entire six interest rate series. Interestingly, the first component comprises almost exactly equal weights in all six series while the second component puts a larger negative weight on the shortest yield and gradually increasing weights thereafter. This ties in with the common belief that the first component captures the level of interest rates, the second component captures the slope of the term structure (and the third component captures curvature in the yield curve).

10 Diagnostic Testing

Reading: Brooks (2019, Chapter 5)

10.1 Testing for Heteroscedasticity

Reading: Brooks (2019, Section 5.4)

In this example we will undertake a test for heteroscedasticity in Stata, using the **'macro.dta'** workfile. We will inspect the residuals of the APT-style regression of the excess return of Microsoft shares, 'ermsoft', on unexpected changes in a set of financial and macroeconomic variables, which we have estimated above. Thus, the first step is to reproduce the regression results. The simplest way is to re-estimate the regression by typing the command

regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm

in the *Command* window and press **Enter**. This time we are less interested in the coefficient estimates reported in the *Output* window, but we focus on the properties of the residuals from this regression. To get a first impression about the properties of the residuals we want to plot them. When Stata performs an estimation it keeps specific estimates in its memory which can then be used in postestimation analysis; among them the residuals of a regression. To obtain the residuals we use the command **predict** which allows us to create a variable of the predicted variables in memory.[40] We click on **Statistics/Postestimation** and in the 'Postestimation Selector' we select **Predictions** and **Predictions and their SEs, leverage statistics, distance statistics, etc.** (Figure 35).

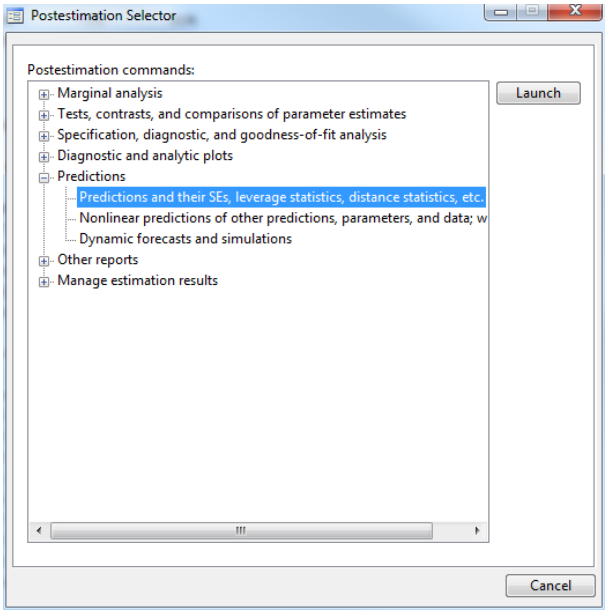


Figure 35: Postestimation Selector for Predictions

In the 'predict' specification window that appears we name the residual series we want to generate in the **New variable name:** box as **resid** and select the option **Residuals** which specifies that the new **'resid'** series should contain the (unadjusted) residuals (Figure 36). By pressing **OK** we should find that the variable 'resid' now appears as a new variable in the *Variables* window.

[40] For more information about the functionalities of **predict**, refer to the respective entry in the Stata Manual.

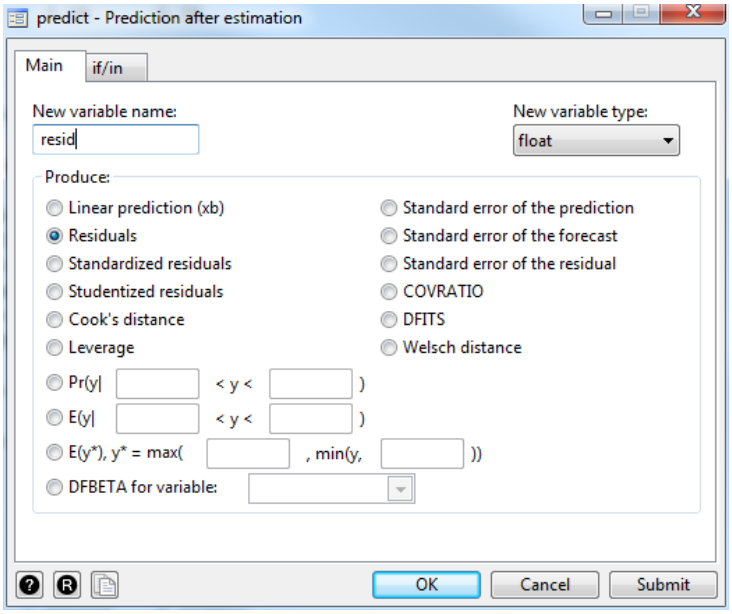


Figure 36: Obtaining Residuals after Estimation

To plot this series we simply select **Graphics/Time-series graphs/Line plots**, click on **Create...** and in the new window we select **resid** as the **Y Variable**. Clicking **Accept** and then **OK** should generate a time-series plot of the residual series, similar to Figure 37.[41]

Let us examine the pattern of residuals over time. If the residuals of the regression have systematically changing variability over the sample, that is a sign of heteroscedasticity. In this case, it is hard to see any clear pattern (although it is interesting to note the considerable reduction in volatility post-2003), so we need to run the formal statistical test.

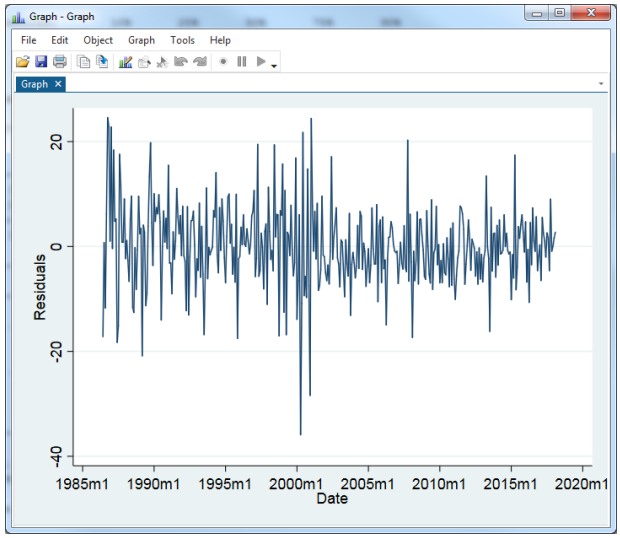


Figure 37: Time-Series Plot of Residuals

To do so we click on **Statistics/Postestimation** and then select **Specification, diagnostic, and goodness-of-fit analysis/Tests for heteroskedasticity** (Figure 38, left panel). In the 'estat' specification window, we are first asked to select the type of **Reports and statistics (subcommand)** as in the right panel of Figure 38. The default option is **Tests for heteroskedasticity (hettest)** which

[41] Alternatively, you can directly use the command **twoway (tsline resid)** to generate the residual plot, after having generated the 'resid' series using the command **predict resid, residuals**.

is exactly the command that we are looking for; thus we do not make any changes at this stage. Next we specify the type of heteroscedasticity test to compute using the drop-down menu next to **Test to compute**. We can choose between three options: (1) the (original) **Breusch-Pagan/Cook-Weisberg** test, which assumes that the regression disturbances are normally distributed; (2) the **N*R2 version of the score test** that drops the normality assumption; (3) the **F-statistic** version which also drops the normality assumption. Let us start by selecting the **Breusch-Pagan/Cook-Weisberg** test.

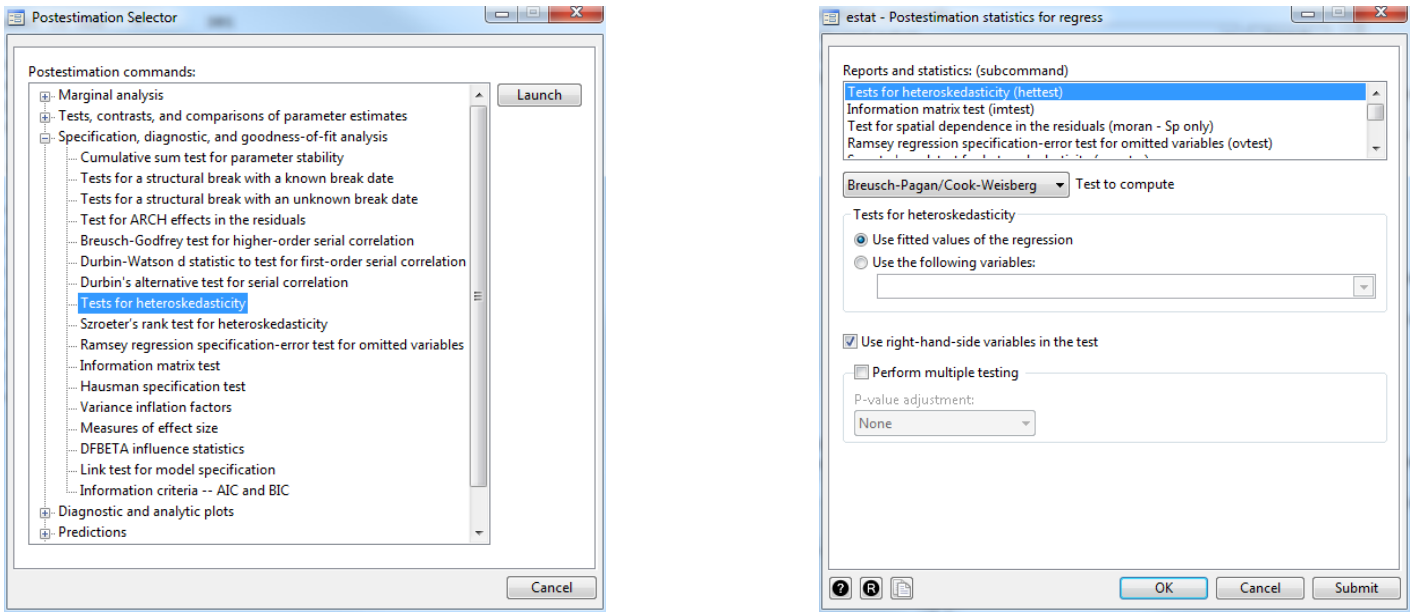


Figure 38: Specification Window for Heteroscedasticity Tests

```
. estat hettest, rhs

Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
         Ho: Constant variance
         Variables: ersandp dprod dcredit dinflation dmoney dspread rterm

         chi2(7)      =     6.31
         Prob > chi2  =   0.5037

. estat hettest, iid rhs

Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
         Ho: Constant variance
         Variables: ersandp dprod dcredit dinflation dmoney dspread rterm

         chi2(7)      =     3.16
         Prob > chi2  =   0.8698

. estat hettest, fstat rhs

Breusch-Pagan / Cook-Weisberg test for heteroskedasticity
         Ho: Constant variance
         Variables: ersandp dprod dcredit dinflation dmoney dspread rterm

         F(7 , 375)   =     0.45
         Prob > F     =   0.8729

.
```

Before we start, we need to tick **Use right-hand-side variables in the test**. This ensures the residual variances will be regressed on the variables of the previous regression. Clicking **OK** will generate the

test statistics in the *Output* window displayed above.

As you can see the null hypothesis is one of constant variance, i.e., homoscedasticity. With a χ^2-value of 6.31 and a corresponding p-value of 0.5037, the Beusch-Pagan/Cook-Weisberg test suggests not to reject the null hypothesis of constant variance of the residuals. To test the robustness of this result to alternative distributional assumptions, we can also run the other two test types. Below is the output for all three tests. As you can see from the test statistics and p-values, both of the other tests confirm the result and suggest that there is not a problem of heteroscedastic errors for our APT-style model.

10.2 Using White's Modified Standard Error Estimates

Reading: Brooks (2019, Subsection 5.4.3)

We can estimate the regression with heteroscedasticity-robust standard errors in Stata. When we open the **regress** specification window we see different tabs. So far we have only focused on the **Model** tab that specifies the dependent and independent variables. If we move to the **SE/Robust** tab, we are presented with different options for adjusting the standard errors (Figure 39).

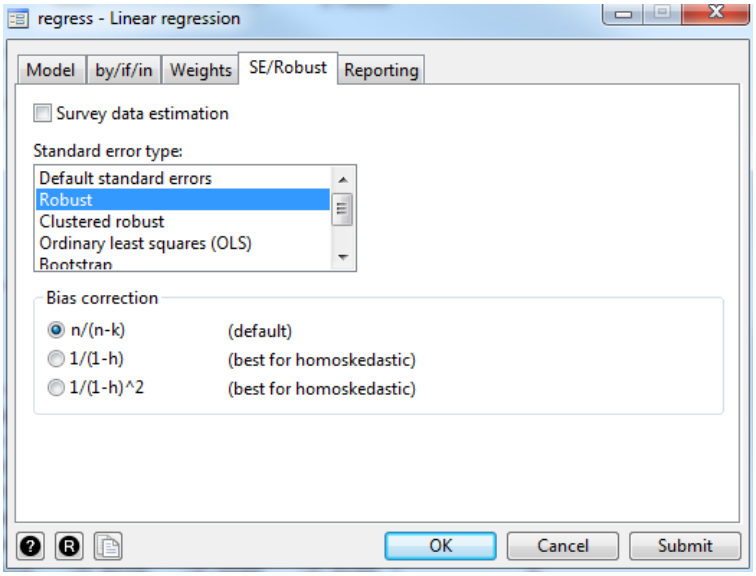


Figure 39: Adjusting Standard Errors for OLS Regressions

In order to obtain standard errors that are robust to heteroscedasticity we select the option **Robust**. Beneath the selection box, three **Bias correction** options appear. We keep the default option.[42]

Comparing the regression output for our APT-style model using robust standard errors with that using ordinary standard errors (see Section 7 of this guide), we find that the changes in significance are only marginal, as shown in the output below using robust standard errors.

[42] Alternatively, you can directly adjust the estimation command in the Command window to account for robust standard errors by adding '**, vce(robust)**' at the end of the command, i.e., regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm, vce(robust). For more information on the different standard error adjustments, refer to the entries **regress** and **vce_option** in the Stata Manual.

```
. regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm, vce(robust)

Linear regression                               Number of obs     =        383
                                                F(7, 375)         =      29.89
                                                Prob > F          =     0.0000
                                                R-squared         =     0.3452
                                                Root MSE          =     7.8448

------------------------------------------------------------------------------
             |               Robust
    ermsoft |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    ersandp |   1.280799   .0929615    13.78   0.000     1.098008    1.46359
      dprod |  -.3030317   .6345495    -0.48   0.633    -1.550753    .9446894
    dcredit |  -.0253637   .0208151    -1.22   0.224    -.0662926    .0155652
  dinflation |    2.19467   1.306803     1.68   0.094    -.3749094    4.76425
     dmoney |  -.0068714   .0109006    -0.63   0.529    -.0283054    .0145625
    dspread |   2.260065   3.427813     0.66   0.510    -4.480078    9.000209
      rterm |   4.733069   1.726547     2.74   0.006     1.338143    8.127996
      _cons |   1.326003   .4590678     2.89   0.004     .4233329    2.228672
------------------------------------------------------------------------------

.
```

Of course, only the standard errors have changed and the parameter estimates remain identical to those estimated before. However, this has not resulted in changes about the conclusions reached about the significance of any variable.

10.3 The Newey–West Procedure for Estimating Standard Errors

Reading: (Brooks (2019, Subsection 5.5.7)

In this subsection, we will apply the Newey–West procedure for estimating heteroscedasticity and autocorrelation robust standard errors in Stata. Unlike the robust standard error adjustment which is an optional feature within the basic **regress** command, the Newey–West procedure is based on a separate estimator and thus a separate Stata command.

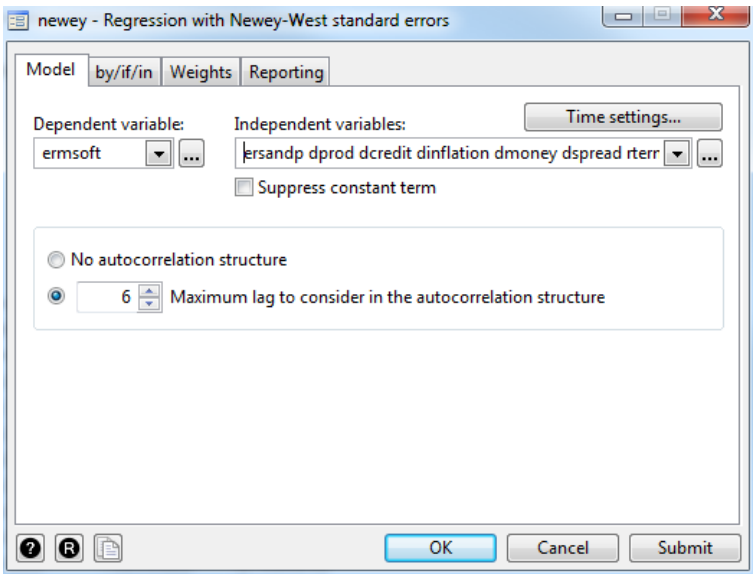


Figure 40: Specifiying Regressions with Newey–West Standard Errors

To access this command, we select **Statistics/Time series/Regressions with Newey-West std. errors**. In the window that appears, we are first asked to define the dependent variable and the independent variables, as shown in Figure 40.

Then we are asked to specify the **Maximum lag to consider in the autocorrelation structure**, i.e., we manually input the maximum number of lagged residuals that should be considered for inclusion in the model. There might be different economic motivations for choosing the maximum lag length, depending on the specific analysis one is undertaking. In our example we decide to include a maximum lag length of six, implying that we assume that the potential autocorrelation in our data does not go beyond the window of six months.[43] By clicking **OK**, the following regression results appear in the *Output* window.

```
. newey ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm, lag(6)

Regression with Newey-West standard errors      Number of obs     =        383
maximum lag: 6                                  F(  7,      375)  =      24.93
                                                Prob > F          =     0.0000

------------------------------------------------------------------------------
             |             Newey-West
    ermsoft |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
    ersandp |   1.280799   .0998922    12.82   0.000      1.08438    1.477218
      dprod |  -.3030317    .521617    -0.58   0.562    -1.328693    .7226291
    dcredit |  -.0253637    .022341    -1.14   0.257     -.069293    .0185656
  dinflation |    2.19467   1.313617     1.67   0.096    -.3883089    4.777649
     dmoney |  -.0068714   .0109474    -0.63   0.531    -.0283974    .0146545
    dspread |   2.260065   2.841813     0.80   0.427    -3.327821    7.847951
      rterm |   4.733069   1.758514     2.69   0.007     1.275285    8.190854
      _cons |   1.326003   .5027048     2.64   0.009      .337529    2.314476
------------------------------------------------------------------------------

.
```

10.4 Autocorrelation and Dynamic Models

Reading: Brooks (2019, Subsections 5.5.7 – 5.5.11)

In Stata, the lagged values of variables can be used as regressors or for other purposes by using the notation $L.x$ for a one-period lag, $L5.x$ for a five-period lag, and so on, where x is the variable name. Stata will automatically adjust the sample period used for estimation to take into account the observations that are lost in constructing the lags. For example, if the regression contains five lags of the dependent variable, five observations will be lost and estimation will commence with observation six. Additionally, Stata also accounts for missing observations when using the time operator L. Note, however, that in order to use the time operator $L.$ it is essential to set the time variable in the dataset using the command **tsset**. In this section, we want to apply different tests for autocorrelation in Stata, using the APT-style model of the previous section ('macro.dta' workfile).[44]

The simplest test for autocorrelation is due to Durbin and Watson (1951). To access the Durbin-Watson (DW) test, we use the 'Postestimation Selector' via **Statistics/Postestimation** and then select **Durbin-Watson d statistic to test for first-order serial correlation** under the category

[43] Note that if we were to specify **No autocorrelation structure** the Newey–West adjusted standard errors would be the same as the robust standard errors introduced in the previous section.

[44] Note that it is important that the last model you have estimated is **regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm**.

Specification, diagnostic, and comparisons of parameter estimates (Figure 41).

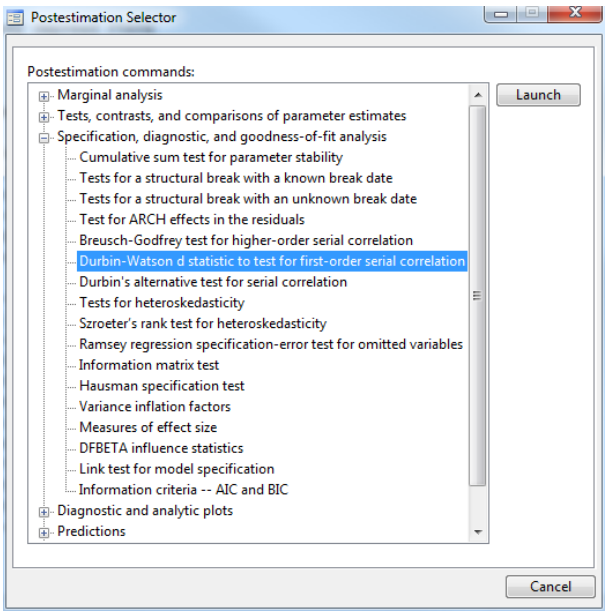


Figure 41: Postestimation Selector for the Durbin-Watson Test

Next we click **OK** as the correct option has already been pre-selected, i.e., **Durbin-Watson d statistic(dwatson - time series only)**. The following test results will appear in the *Output* window.

```
. estat dwatson

Durbin-Watson d-statistic(  8,   383) =  2.097394

.
```

The value of the DW statistic is 2.10. What is the appropriate conclusion regarding the presence or otherwise of first order autocorrelation in this case? An alternative test for autocorrelation is the Breusch–Godfrey test. It is a more general test for autocorrelation than DW and allows to test for higher order autocorrelation. In Stata, the Breusch–Godfrey test can be conducted by selecting **Breusch-Godfrey test for higher-order serial correlation** in the 'Postestimation Selector'. Again, the correct option **Breusch-Godfrey test (bgodfrey - time series only)** is pre-selected and we only need to **Specify a list of lag orders to be tested**. Assuming that we select to employ **10** lags in the test, the results should appear as below.

```
. estat bgodfrey, lags(10)

Breusch-Godfrey LM test for autocorrelation
```

lags(p)	chi2	df	Prob > chi2
10	4.767	10	0.9062

```
                    H0: no serial correlation

.
```

10.5 Testing for Non-Normality

Reading: Brooks (2019, Section 5.7)

One of the most commonly applied tests for normality is the Jarque–Bera (JB) test.[45] Assume we would like to test whether the normality assumption is satisfied for the residuals of the APT-style regression of Microsoft stock on the unexpected changes in the financial and economic factors, i.e., the 'resid' variable that we have created in subsection 10.1. Before calculating the actual test-statistic, it might be useful to have a look at the data as this might give us a first idea whether the residuals might be normally distributed. If the residuals follow a normal distribution we expect a histogram of the residuals to be bell-shaped (with no outliers). To create a histogram of the residuals we click on **Graphics** and select **Histogram**. In the window that appears we are asked to select the variable for which we want to generate the histogram (Figure 42, left panel).

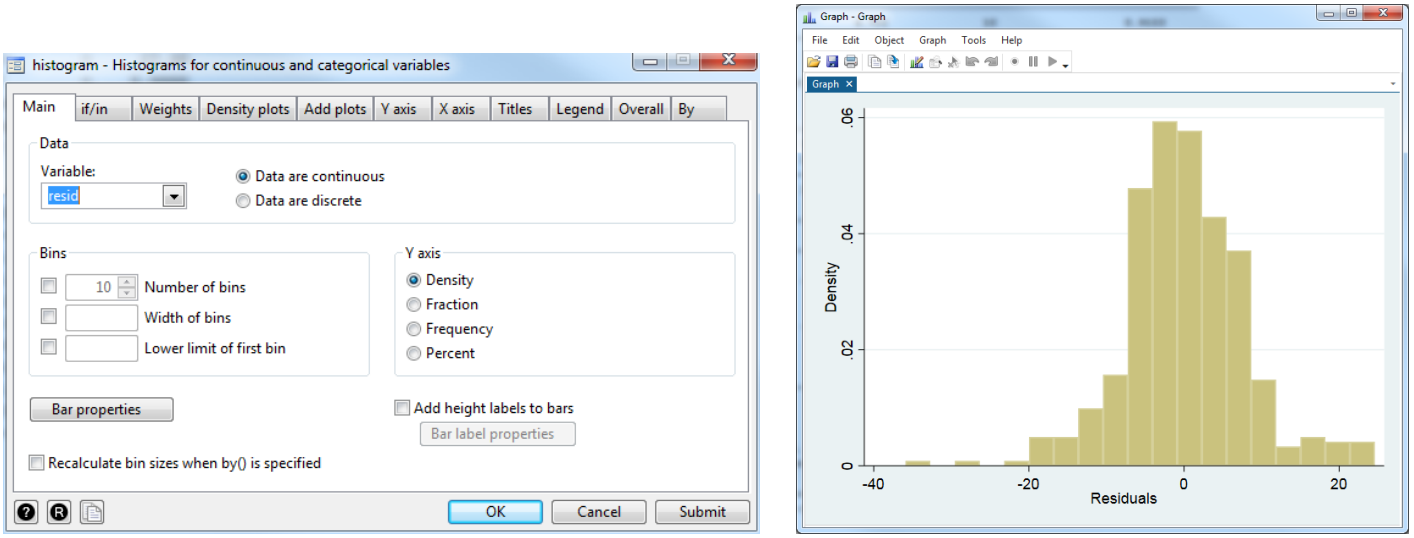


Figure 42: Generating a Histogram of Residuals

In our case we define **Variable: resid**. Our data are continuous so we do not need to make any changes regarding the data type. In the bottom left of the window we can specify the number of bins and as well as the width of the bins. We stick with the default settings for now and click **OK** to generate the histogram (Figure 42, right panel).

Looking at the histogram plot we see that the distribution of the residuals roughly resembles a bell-shape, although we also find that there are some large negative outliers which might lead to a considerable negative skewness of the data series. We could increase the number of bins or lower the width of bins in order to get a more differentiated histogram.

However, if we want to test the normality assumption of the disturbances more formally it is best to turn to a quantitative normality test. The standard test for the normality of a data series in Stata is the Skewness and kurtosis test (**sktest**), which is a variation of the Jarque–Bera test. 'sktest' presents a test for normality based on skewness and another based on kurtosis and then combines the two tests into an overall test statistic. In contrast to the traditional Jarque–Bera test, which is also based on the skewness and kurtosis of a data series, the **sktest** in Stata corrects for the small sample bias of the Jarque–Bera test by using a bootstrapping procedure. Thus it proves to be a particularly useful test if the sample size of the data analysed is small. To access the 'sktest' we click on **Statistics/Summaries, tables, and tests/Distributional plots and tests/Skewness and kurtosis test for normality**.

[45] For more information on the intuition behind the Jarque–Bera test, refer to chapter 5 in Brooks (2019).

In the test specification window, we define the variable on which we want to perform the normality test as **resid** (Figure 43).

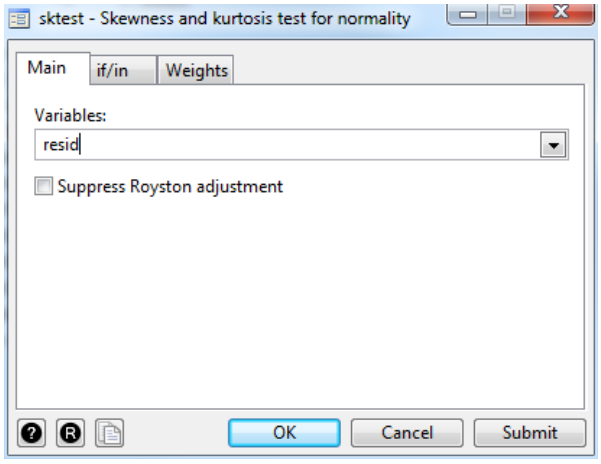


Figure 43: Specifying the Normality Test

The Royston adjustment is for the small sample bias. For now, we keep the Royston adjustment and do not check the box to suppress it. Instead, we press **OK** to generate the following test statistics:

```
. sktest resid

                    Skewness/Kurtosis tests for Normality
                                                          ------ joint ------
    Variable |        Obs  Pr(Skewness)  Pr(Kurtosis) adj chi2(2)    Prob>chi2
-------------+-------------------------------------------------------------
       resid |        383      0.9636        0.0000       17.89         0.0001

.
```

Stata reports the probabilities that the skewness of the residuals resembles that of a normal distribution. Additionally, it reports the adjusted χ^2 value and p-value for the test that the residuals are overall normally distributed, i.e., that both the kurtosis and the skewness are those of the normal distribution. On the one hand, we find that the test for skewness does not reject the normality assumption of a zero skewness. On the other hand, the test strongly rejects the assumption of a kurtosis equal to three. The hypothesis that both kurtosis and skewness jointly resembles those of a normal distribution can be strongly rejected. We can check whether our results change if we do not apply the Royston adjustment for small sample bias. To do this, we check the box **Suppress Royston adjustment** in the specification window for the normality test. However, our overall conclusion is unchanged.

What could cause this strong deviation from normality? Having another look at the histogram, it appears to have been caused by a small number of very large negative residuals representing monthly stock price falls of more than 20%.

10.6 Dummy Variable Construction and Application

Reading: Brooks (2019, Subsection 5.7.2)

As we saw from the plot of the distribution above, the non-normality in the residuals from the Microsoft regression appears to have been caused by a small number of outliers in the sample. Such events can be identified if they are present by plotting the actual values and the residuals of the regression.

We have already generated a data series containing the residuals of the Microsoft regression. Let us now create a series of the fitted values. For this, we use the **predict** command again by opening the 'Postestimation Selector' (**Statistics/Postestimation**) and selecting **Predictions/Predictions and their SEs, leverage statistics, distance statistics, etc.**. In the specification window, we name the variable **fitted** and define that it should contain the **Linear prediction** for the Microsoft regression (first option in list list named **Produce**) (Figure 44). Then we only need to click **OK** and we should find a new variable named fitted in the *Variables* window.

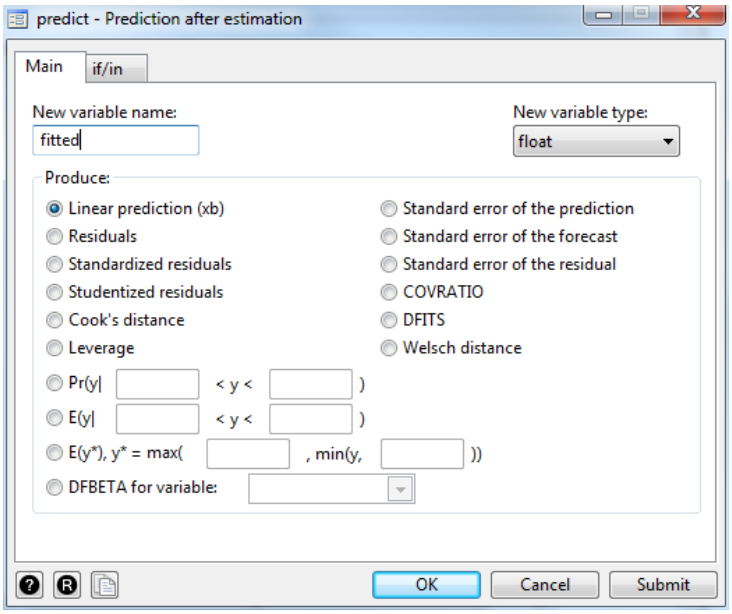


Figure 44: Generating a Series of Fitted Values

In order to plot both the residuals and fitted values in one times-series graph we select **Graphics/Time-series graphs/Line plots**. In the specification window, we press **Create...** and define the **Y variable** to be **resid** while keeping the other default selections. Then we press **Accept**. Next we click on **Create...** again. Now we choose our **Y variable** to be **fitted**. Again, we keep all default options and press **Accept** to return to the main specification window. You should now see that there are two plots specified, **Plot 1** and **Plot 2**. By clicking **OK**, Stata produces a time-series plot of the residual and fitted values that should resemble that in Figure 45.

From the graph, it can be seen that there are several large (negative) outliers, but the largest of all occur in 2000. All of the large outliers correspond to months where the actual return was much smaller (i.e., more negative) than the model would have predicted, resulting in a large residual. Interestingly, the residual in October 1987 is not quite so prominent because even though the stock price fell, the market index value fell as well, so that the stock price fall was at least in part predicted.

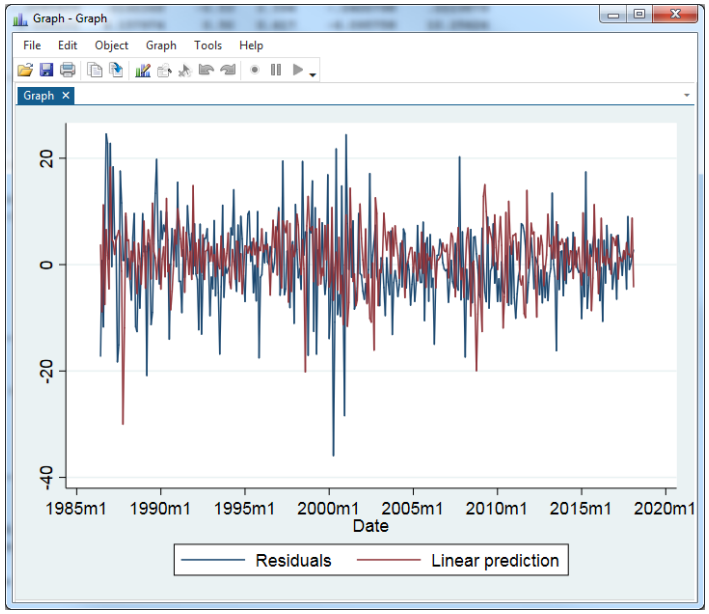


Figure 45: Regression Residuals and Fitted Series

In order to identify the exact dates that the biggest outliers were realised, it is probably easiest to just examine a table of values for the residuals, which can be achieved by changing to the **Data Editor** view (i.e., pressing the *Data Editor* symbol in the Stata menu or entering **edit** into the command window and pressing **Enter**). We can now sort the data by residuals in order to directly spot the largest negative values. To do so we click on **Data/Sort**. In the 'sort' specification window we keep the default option **Standard sort (ascending)** and only specify the variable based on which we want to sort the dataset, which is **resid** in our case (Figure 46). Then we press **OK** in order to execute the sorting. Now the dataset should be sorted by 'resid', starting with the lowest values and ending with the highest values for 'resid'. If we do this, it is evident that the two most extreme residuals were in April (−36.075) and December 2000 (−28.143).

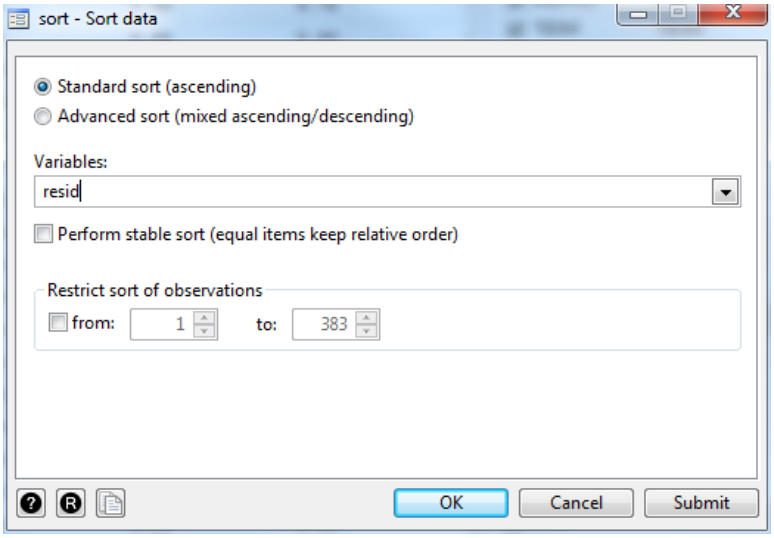


Figure 46: Sorting Data by Values of Residuals

One way of removing the (distorting) effect of big outliers in the data is by using dummy variables. It would be tempting, but incorrect, to construct one dummy variable that takes the value 1 for both April 2000 and December 2000, but this would not have the desired effect of setting both residuals to

zero. Instead, to remove two outliers requires us to construct two separate dummy variables. In order to create the Apr 00 dummy first, we generate a series called 'APR00DUM'. To do this we return to the main Stata screen and click on **Data/Create or change data/Create new variable**. In the variable specification window, we change the **Variable type** to **byte** (as dummy variables are binary variables) and define **Variable name: APR00DUM** (Figure 47).

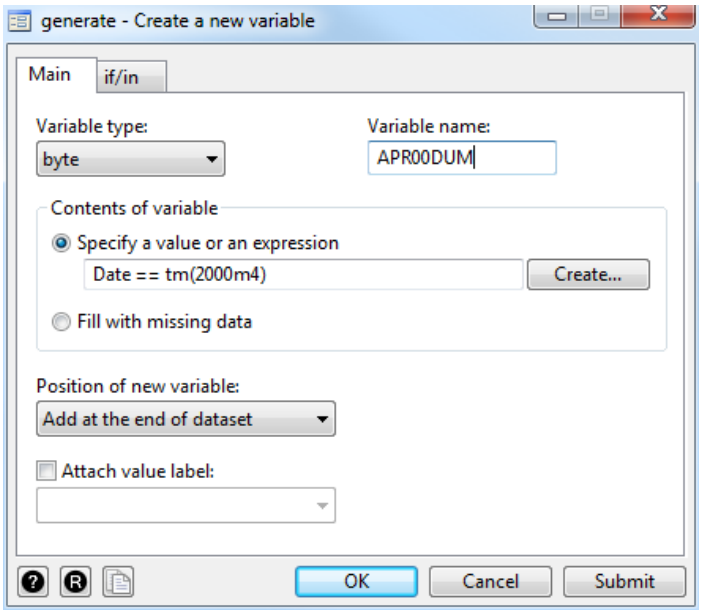


Figure 47: Creating a Dummy Variable for Outliers

Now we need to specify the content of the variable, which we do in the following way: In the **Specify a value or an expression** box we type in:

$$\textbf{Date == tm(2000m4)}$$

which means that the new variable takes the value 1, if the Date is equal to 2000m4 and 0 otherwise. Note that the function **tm()** is used to allow us to type the Date as a human-readable date instead of the coded Stata value.

We can check whether the dummy is correctly specified by visually inspecting the **APR00DUM** series in the *Data Editor*. There should only be one single observation for which the dummy takes the value of one, which is April 2000, whereas it should be zero for all other dates. We repeat the process above to **create another dummy variable** called **'DEC00DUM'** that takes the value 1 in December 2000 and zero elsewhere.

Let us now re-run the regression to see whether the results change after removing the effect of the two largest outliers. For this we just add the two dummy variables **APR00DUM** and **DEC00DUM** to the list of independent variables. This can most easily be achieved by looking for the regression command in the *Review* window. By clicking on it the command reappears in the *Command* window. We add **APR00DUM** and **DEC00DUM** at the end of the equation. The output of this regression should look as follows.

```
. regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm APR00DUM DEC00DUM

      Source |       SS           df       MS      Number of obs   =       383
-------------+----------------------------------   F(9, 373)       =     28.33
       Model |  14311.0654         9  1590.11838   Prob > F        =    0.0000
    Residual |  20933.7291       373  56.1225982   R-squared       =    0.4060
-------------+----------------------------------   Adj R-squared   =    0.3917
       Total |  35244.7945       382  92.26386     Root MSE        =    7.4915

------------------------------------------------------------------------------
     ermsoft |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     ersandp |    1.253853    .090222    13.90   0.000     1.076446    1.431261
       dprod |   -.3211312   .7049451    -0.46   0.649    -1.707296    1.065034
     dcredit |   -.0157176   .0260444    -0.60   0.547    -.0669299    .0354947
  dinflation |    1.442065   1.214637     1.19   0.236    -.9463309     3.83046
      dmoney |   -.0056957    .014868    -0.38   0.702    -.0349312    .0235398
     dspread |     1.86929   3.955274     0.47   0.637    -5.908141     9.64672
       rterm |    4.264174   1.640514     2.60   0.010     1.038359    7.489989
    APR00DUM |   -37.02883   7.576018    -4.89   0.000    -51.92589   -22.13177
    DEC00DUM |   -28.72995   7.546383    -3.81   0.000    -43.56874   -13.89116
       _cons |     1.41976   .4543885     3.12   0.002     .5262755    2.313244
------------------------------------------------------------------------------

.
```

Note that the dummy variable parameters are both highly significant and take approximately the values that the corresponding residuals would have taken if the dummy variables had not been included in the model.[46] By comparing the results with those of the regression above that excluded the dummy variables, it can be seen that the coefficient estimates on the remaining variables change quite a bit in this instance. The inflation parameter is now insignificant and the R^2 value has risen from 0.34 to 0.41 because of the perfect fit of the dummy variables to those two extreme outlying observations.

Finally, we can re-examine the normality test results of the residuals based on this new model specification. First we have to create the new residual series by opening the 'predict' specification window (**Statistics/Postestimation/Predictions/Predictions and their SEs, leverage statistics, distance statistics, etc.**). We name the new residual series **resid_new** and select the second **Produce** option **Residuals**. Then we re-run the Skewness and Kurtosis test (**sktest**) on this new series of residuals using **Statistics/Summaries, tables, and tests/Distributional plots and tests/Skewness and kurtosis normality test**. Note that we can test both versions, with and without the Royston adjustment for small sample bias.

We see that now the skewness test also strongly rejects the normality assumption with a p-value of 0.0016. The residuals are still a long way from following a normal distribution, and that the joint null hypothesis of normality is still strongly rejected, probably because there are still several very large outliers. While it would be possible to continue to generate dummy variables, there is a limit to the extent to which it would be desirable to do so. With this particular regression, we are unlikely to be able to achieve a residual distribution that is close to normality without using an excessive number of dummy variables. As a rule of thumb, in a monthly sample with 381 observations, it is reasonable to include, perhaps, two or three dummy variables for outliers, but more would probably be excessive.

[46] Note the inexact correspondence between the values of the residuals and the values of the dummy variable parameters because two dummies are being used together; had we included only one dummy, the value of the dummy variable coefficient and that which the residual would have taken would be identical.

10.7 Multicollinearity

Reading: Brooks (2019, Section 5.8)

Let us assume that we would like to test for multicollinearity issues in the Microsoft regression ('macro.dta' workfile). To generate a correlation matrix in Stata, we click on **Statistics/Summaries, tables, and test/Summary and descriptive statistics/Correlations and covariances**. In the **Variables** dialog box we enter the list of regressors (not including the regressand or the S&P500 returns), as in Figure 48.

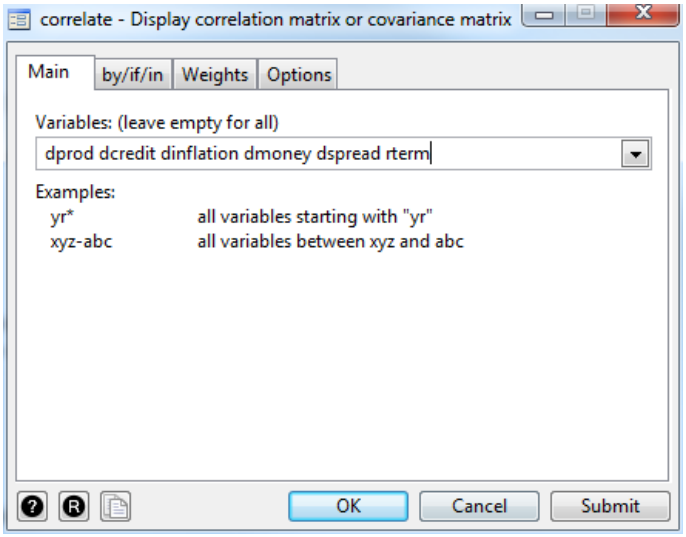


Figure 48: Generating a Correlation Matrix

After clicking **OK**, the following correlation matrix appears in the *Output* window.

```
. correlate dprod dcredit dinflation dmoney dspread rterm
(obs=383)

             |   dprod  dcredit dinfla~n   dmoney  dspread    rterm
-------------+------------------------------------------------------
       dprod |   1.0000
     dcredit |   0.0943   1.0000
   dinflation|  -0.1436  -0.0246   1.0000
      dmoney |  -0.0525   0.1502  -0.0936   1.0000
     dspread |  -0.0528   0.0628  -0.2271   0.1707   1.0000
       rterm |  -0.0438  -0.0040   0.0416   0.0038  -0.0176   1.0000

.
```

Do the results indicate any significant correlations between the independent variables? In this particular case, the largest observed correlations (in absolute value) are 0.17 between the money supply and spread variables, and –0.23 between the spread and unexpected inflation. These figures are probably sufficiently small that they can reasonably be ignored.

10.8 The RESET Test for Functional Form

Reading: Brooks (2019, Section 5.9)

To conduct the RESET test for our Microsoft regression we open the 'Postestimation Selector' and under **Specification, diagnostic, and goodness-of-fit analysis** we select **Ramsey regression specification-error test for omitted variables** (Figure 49).

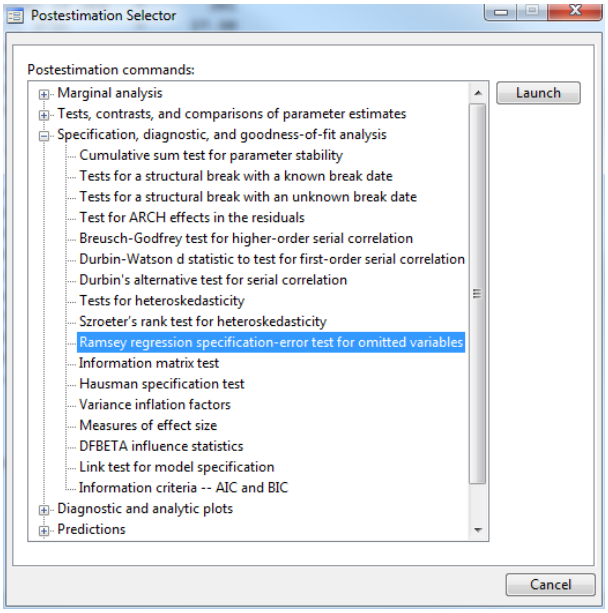


Figure 49: Specifying the RESET Test

In the 'estat' specification window that appears the correct option is already pre-selected from the drop-down menu and we simply press **OK**. Stata reports the $F(3, 372)$-value for the test of the null hypothesis that the model is correctly specified and has no omitted variables, i.e., the coefficient estimates on the powers of the higher order terms of the fitted values are zero:

```
. estat ovtest

Ramsey RESET test using powers of the fitted values of ermsoft
       Ho:  model has no omitted variables
                 F(3, 372) =      0.99
                  Prob > F =      0.3957

.
```

Since the F-statistic has three degrees of freedom, we can infer that Stata included up to three powers of the fitted values, $\hat{y}^2, \hat{y}^3$ and $\hat{y}^4$, in auxiliary the regressions. With an F-value of 0.99 and a corresponding p-value of 0.3957, the RESET test results imply that we cannot reject the null hypothesis that the model has no omitted variables. In other words, we do not find strong evidence that the chosen linear functional form of the model is incorrect.

10.9 Stability Tests

Reading: Brooks (2019, Section 5.12)

There are two types of stability tests that we want to apply: the Chow (analysis of variance) test and the predictive failure test. To access the Chow test, we open the 'Postestimation Selector' (**Statistics/Postestimatio** and select **Specification, diagnostic, and goodness-of-fit analysis**. Note that it is not possible

to conduct a Chow test or a parameter stability test when there are outlier dummy variables in the regression. Thus, we have to ensure that the last estimation that we run is the Microsoft regression omitting the APR00DUM and DEC00DUM dummies from the list of independent variables.[47] This occurs because when the sample is split into two parts, the dummy variable for one of the parts will have values of zero for all observations, which would thus cause perfect multicollinearity with the column of ones that is used for the constant term. Looking at the 'Postestimation Selector', we see that there are two tests for structural breaks, one which tests for structural breaks with a known break date and one test for structural breaks when the break date is unknown. Let us first run **Tests for structural break with a known break date** by selecting the corresponding option (Figure 50, left panel). In the specification window that appears, we are now asked to specify the 'Hypothesized break dates' (Figure 50, right panel).

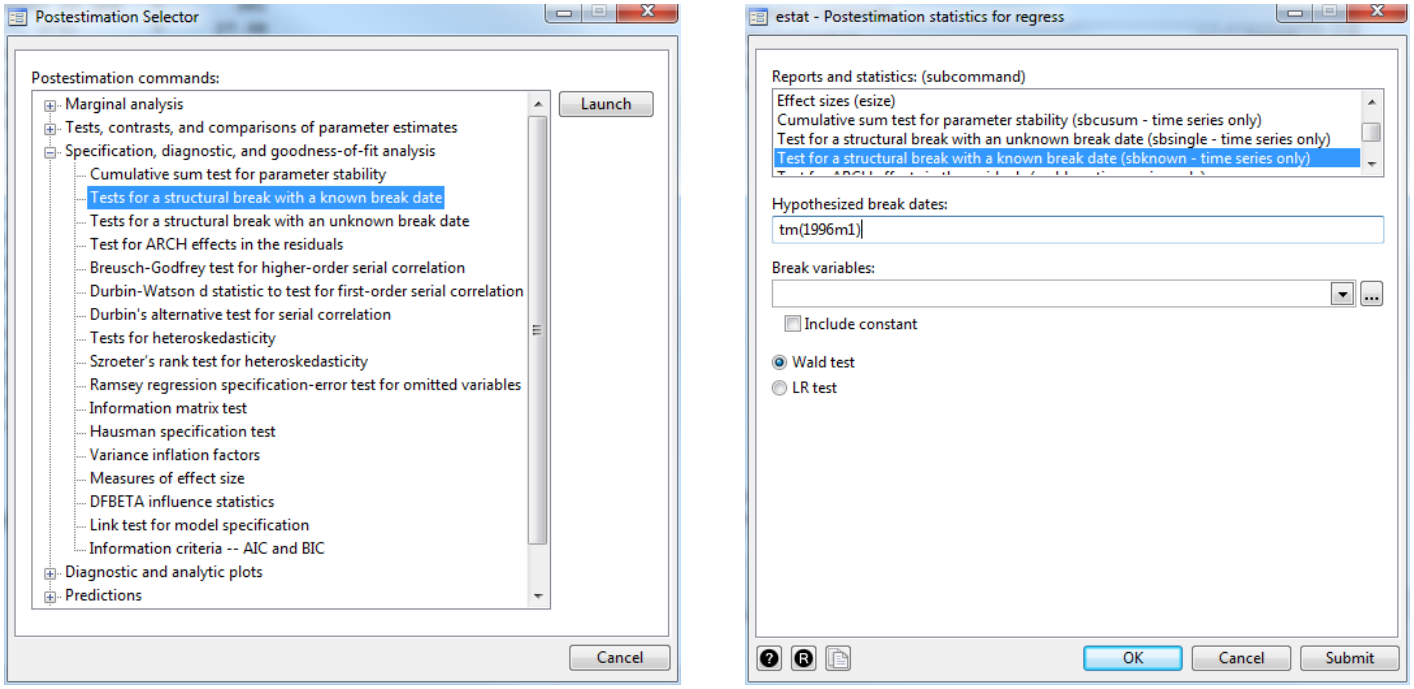


Figure 50: Specifying a Test for Structural Breaks with a Known Break Date

Let us assume that we want to test whether a breakpoint occurred in January 1996, which is roughly in the middle of the sample period. Thus, we specify **Hypothesized break dates: tm(1996m1)**. Note that the 'tm()' is used to tell Stata that the term in the brackets is formatted as a monthly date variable. In the box entitled **Break variables** we could select specific variables of the model to be included in the test. By default, all coefficients are tested. As we do not have any priors as to which variables might be subject to a structural break and which are not, we leave this box empty, and simply press **OK** to generate the following test statistics.

[47] The corresponding command for this regression is: regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm.

```
. estat sbknown, break(tm(1996m1))

Wald test for a structural break: Known break date

                                 Number of obs   =        383
Sample:       1986m5 -  2018m3
Break date:   1996m1
Ho: No structural break

             chi2(8)      =   15.3223
             Prob > chi2  =    0.0532

Exogenous variables:           ersandp dprod dcredit dinflation dmoney dspread rterm
Coefficients included in test: ersandp dprod dcredit dinflation dmoney dspread rterm _cons

.
```

The output presents the statistics of a Wald test of whether the coefficients in the Microsoft regression vary between the two subperiods, i.e., before and after 1996m1. The null hypothesis is one of no structural break. We find that the χ^2 value is 15.3223 and that the corresponding p-value is 0.0532. Thus, we can reject the null hypothesis that the parameters are constant across the two subsamples at the 10% level.

Often, the date when the structural break occurs is not known in advance. Stata offers a variation of the above test that does not require us to specify the break date but tests for each possible break date in the sample. This test can be accessed via the 'Postestimation Selector' as **Test for a structural break with an unknown break date** (second option, see Figure 50, left panel). When selecting this option a new specification window appears where we can specify the test for a structural break (Figure 51).

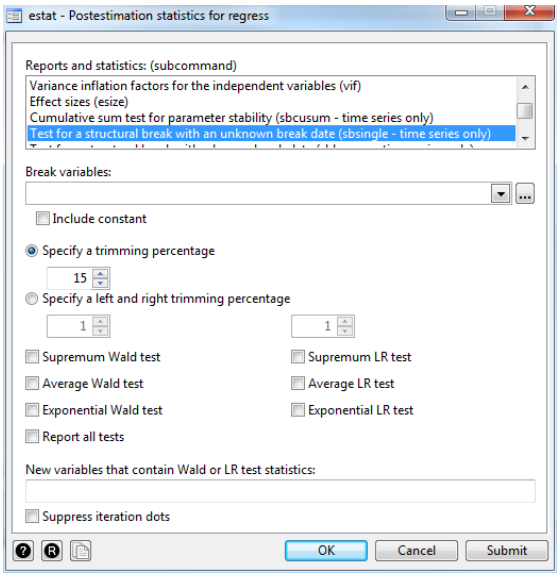


Figure 51: Specifying a Test for Structural Breaks with an Unknown Break Date

However, for now we keep the default specifications and simply press **OK** to generate the test statistics below.

```
. estat sbsingle
        1         2         3         4         5
..................................................    50
..................................................   100
..................................................   150
..................................................   200
..................................................   250
..................

Test for a structural break: Unknown break date

                             Number of obs =       383

Full sample:                  1986m5 -  2018m3
Trimmed sample:               1991m3 -  2013m6
Estimated break date:         2001m5
Ho: No structural break

     Test            Statistic          p-value
-------------------------------------------------
     swald            32.4649           0.0021
-------------------------------------------------
Exogenous variables:           ersandp dprod dcredit dinflation dmoney dspread rterm
Coefficients included in test: ersandp dprod dcredit dinflation dmoney dspread rterm _cons

.
```

Again the null hypothesis is one of no structural breaks. The test statistic and the corresponding p-value suggest that we can reject the null hypothesis that the coefficients are stable over time, confirming that our model has a structural break. The test suggests May 2001 as the break date.

Another way of testing whether the parameters are stable with respect to any break dates is to use one of the tests based on recursive estimation. Unfortunately, there is no built-in function in Stata that automatically produces plots of the recursive coefficient estimates together with standard error bands. In order to visually investigate the parameter stability, we have to run recursive estimations and save the parameter estimates (in a new file). Then we can plot these data series.

To do so, we first select **Statistics/Time series/Rolling-window and recursive estimation**. In the specification window that appears we are first asked to specify the **Stata command to run** which in our case is the base line regression command that we have been using in the previous models (Figure 52, upper panel). So we just type: **regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm**

Next we need to specify the parameters that we would like to be saved from the recursive regressions. As we want to obtain both the **Model coefficients** and the **SE of model coefficients**, i.e., the standard errors, we check both boxes. We also need to specify the window over which we would like to estimate the recursive regressions, i.e., the number of observations over which the estimation will start. We specify the **window** to be **11**, although some other starting point could have been chosen. As we want to estimate recursive regressions, that is Stata gradually adds one further observation to the data subset, we need to check the box **Use recursive samples**.

As a default for recursive regressions, Stata replaces the data in our workfile with the recursive estimates. As we want to keep the data in our workfile, we need to tell Stata to save the recursive estimates in a new file. To do so, we click on the tab **Options** and select **Save results to file** (Figure 52, lower panel). We name the new file **recursive_estimates.dta**. Stata gives us several further options, e.g., to specify the steps of the recursive estimations or the start and end date; but we leave the specifications as they are and press **OK** to generate the recursive estimates.

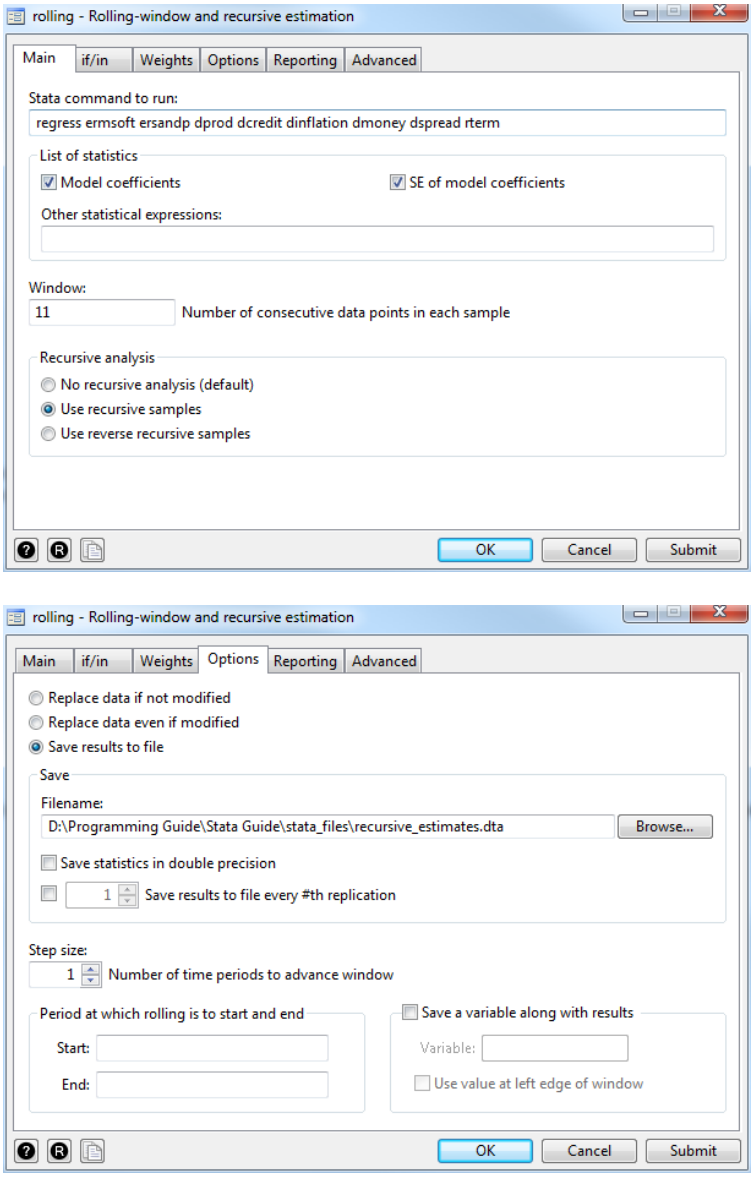


Figure 52: Specifying Recursive Regressions

As you can see from the *Output* window, Stata does not report the regression results for each of the 373 recursive estimations, but produces a '**.**' for each regression. To access the recursive estimates, we would need to **open the new workfile 'recursive_estimates.dta'** that should be stored in the same folder as the original workfile 'macro.dta'. You can open it by double-clicking on the file. The 'recursive_estimates.dta' workfile contains several data series: **start** and **end** contain the start date and the end date over which the respective parameters have been estimated; **_b_ersandp**, for example, contains the recursive coefficient estimates for the excess S&P500 returns, while **_se_ersandp** contains the corresponding standard errors of the coefficient estimates.

In order to visually investigate the parameter stability over the recursive estimations, it is best to generate a time-series plot of the recursive estimates. Assume that we would like to generate such a plot for the recursive estimates of **ersandp**. We would like to plot the actual recursive coefficients together with standard error bands. So first, we need to generate data series for the standard error bands. We decide to generate two series: one for a deviation of 2 standard errors above the coefficient estimate (**_b_ersandp_plus2SE**) and one for a deviation of 2 standard errors below the coefficient estimate (**_b_ersandp_minus2SE**). To do so, we use the following two commands to generate the two series:

generate _b_ersandp_plus2SE = _b_ersandp + 2*_se_ersandp

generate _b_ersandp_minus2SE = _b_ersandp - 2*_se_ersandp

Once we have generated the new variables, we can plot them together with the actual recursive coefficients of 'ersandp'. We click on **Graphics/Time-series graphs/Line plots**. In the graph specification window we click on **Create...** to specify the first data series which comprises the recursive coefficient estimates for 'ersandp' (**_b_ersandp**). In the dialogue box named **Y variable** we type in (**_b_ersandp**) and click on **Line properties** to format this particular data series (Figure 53, top two panels).

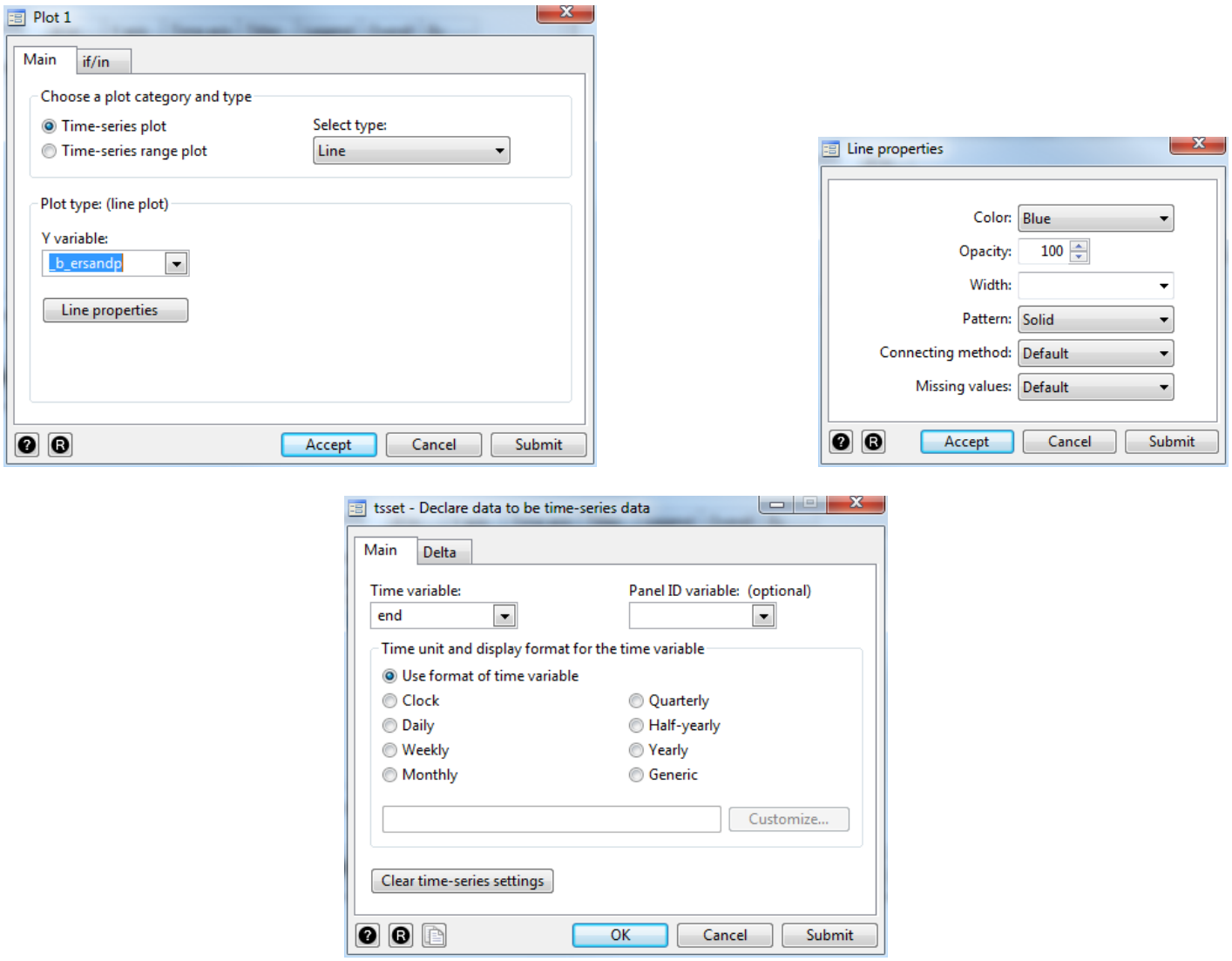


Figure 53: Generating a Plot for the Parameter Stability Test

We change the **Color** to **Blue** and the **Pattern** to **Solid** and press **Accept**. Then we press **Accept** again to return to the main graph specification window. We now click on **Create...** to specify the next series to be included in the graph which is the positive 2 standard error series (**_b_ersandp_plus2SE**). We select this variable from the drop-down menu and click on **Line properties**. For this data series, we select the **Color** to be **Red** and the **Pattern** to be **Dash** and click **Accept** twice to return to the main specification window. We generate **Plot 3** of the negative 2 standard error series (**_b_ersandp_minus2SE**) in a similar way as the previous series, again selecting **Color: Red** and **Pattern: Dash**. Finally, we need to specify the time variable by clicking on **Time settings...** in the

main specification window. In the auxiliary window that appears we define the **Time variable** to be **end** and click **OK** (Figure 53, bottom panel). Now that we have specified all graph characteristics we simply need to press **OK** to generate the graph.

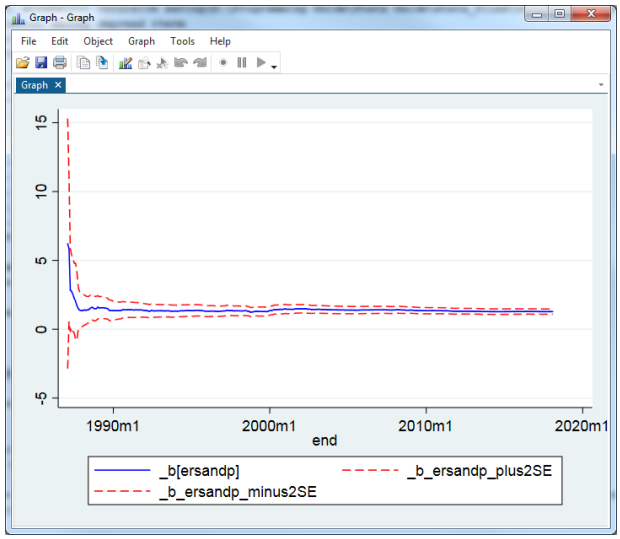


Figure 54: Plot of the Parameter Stability Test

What do we observe from the graph in Figure 54? The coefficients of the first couple of subsamples seem to be relatively unstable with large standard error bands while they seem to stabilise after a short period of time and only show small standard error bands. This pattern is to be expected as it takes some time for the coefficients to stabilise since the first few sets are estimated using very small samples. Given this, the parameter estimates are remarkably stable. We can repeat this process for the recursive estimates of the other variables to see whether they show similar stability over time.

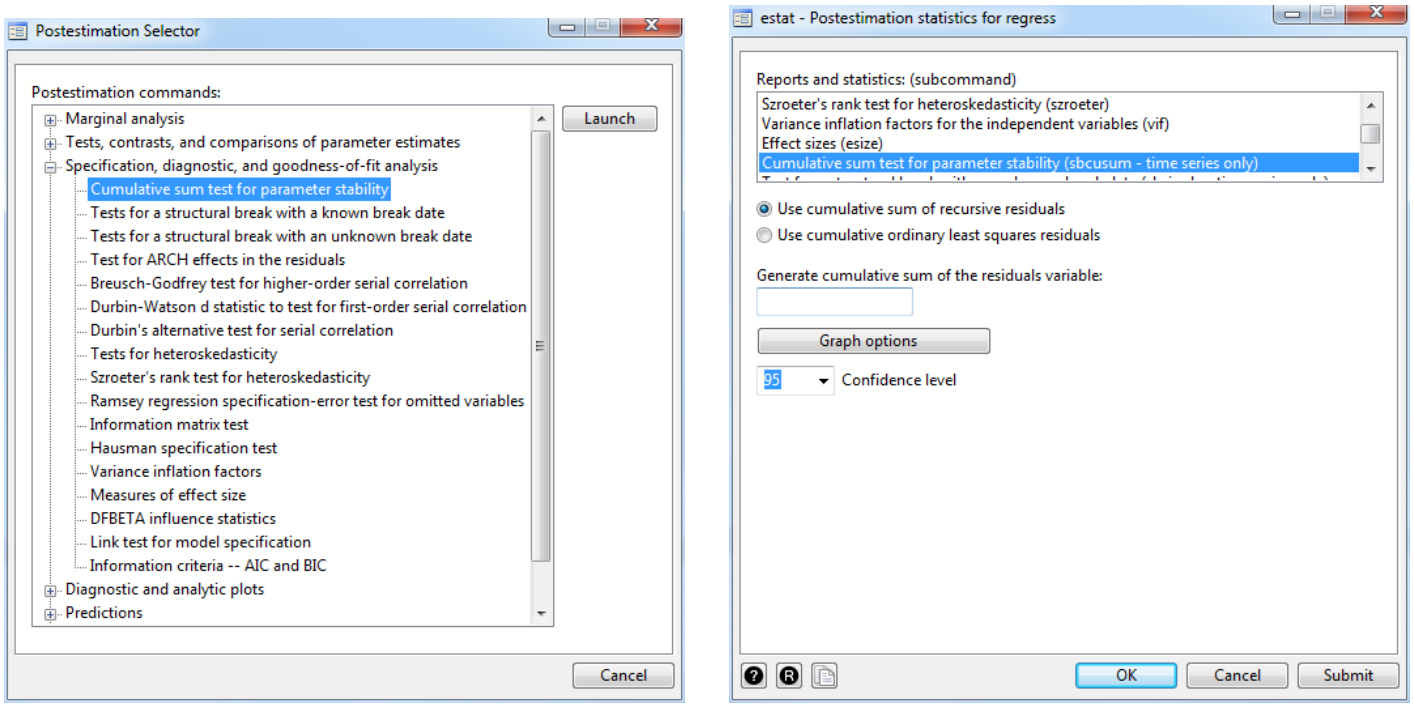


Figure 55: CUSUM Test for Parameter Stability Test

Another option is to run a so called CUSUM test. Therefore, we reopen the **'macro.dta'** file and run

the linear regression again. Afterwards we open the Postestimation Selector (Figure 55, left panel) and launch the command **Specification, diagnostic, and goodness-of-fit-analysis/Cumulative sum test for parameter stability**. We keep the default settings as they are presented in the right panel of Figure 55 and click **OK**.

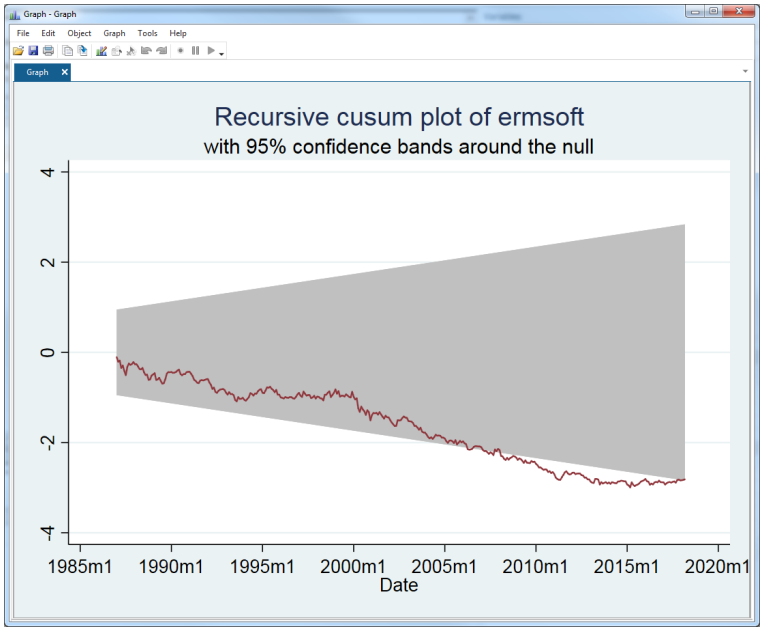


Figure 56: CUSUM Plot

The graph in Figure 56 will pop up. For most of the time the test statistic is within the 95% confidence bands which means we cannot reject the null hypothesis of stable parameters. However, the values are close to the critical value and in the period around 2010 we reject the null hypothesis.

11 Constructing ARMA Models

Reading: Brooks (2019, Sections 6.4 – 6.7)

This example uses the monthly UK house price series which was already incorporated in a Stata workfile in section 2 (**'ukhp.dta'**). So first we re-load the workfile into Stata. There are a total of 326 monthly observations running from February 1991 (recall that the January observation was 'lost' in constructing the lagged value) to March 2018 for the percentage change in house price series. The objective of this exercise is to build an ARMA model for the house price changes. Recall that there are three stages involved: identification, estimation, and diagnostic checking. The first stage is carried out by looking at the autocorrelation and partial autocorrelation coefficients to identify any structure in the data.

11.1 Estimating Autocorrelation Coefficients

To generate a table of autocorrelations, partial correlations and related test statistics we click on **Statistics/Time series/Graphs** and select the option **Autocorrelations & partial autocorrelations**. In the specification window that appears we select **dhp** as the variable for which we want to generate the above statistics and specify that we want to use **12** lags as the **specified number of autocorrelations** (Figure 57).

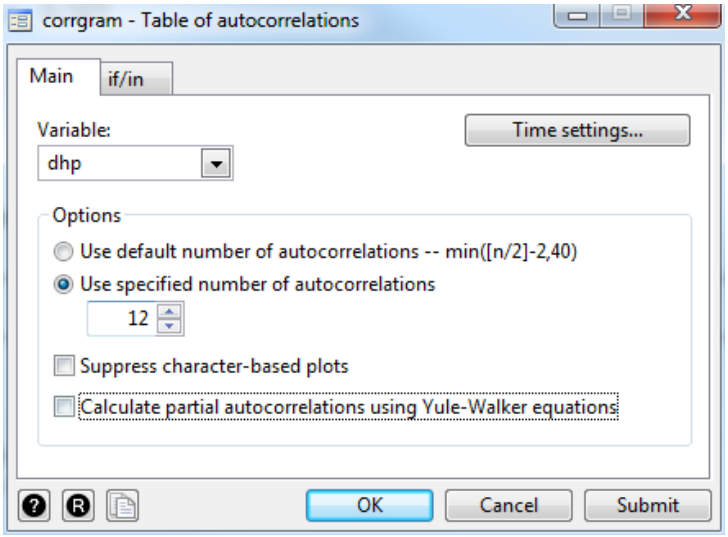


Figure 57: Generating a Correlogram

By clicking **OK**, the correlogram appears in the Stata *Output* window, as given below.[48]

[48]Note that the graphs for the AC and PAC values are very small. You can generate individual graphs for the AC and PAC including confidence bands by selecting **Statistics/Time series/Graphs/Correlogram (ac)** and **Statistics/-Time series/Graphs/Partial correlogram (pac)**, respectively.

```
. corrgram dhp, lags(12)

                                                -1       0       1 -1       0       1
 LAG       AC       PAC      Q      Prob>Q  [Autocorrelation]  [Partial Autocor]
-------------------------------------------------------------------------------------
 1       0.3575   0.3575   42.056  0.0000
 2       0.4181   0.3332    99.75  0.0000
 3       0.2311   0.0134   117.44  0.0000
 4       0.1824  -0.0208   128.48  0.0000
 5       0.1264   0.0070    133.8  0.0000
 6       0.1306   0.0579    139.5  0.0000
 7       0.0608  -0.0290   140.74  0.0000
 8       0.0964   0.0366   143.87  0.0000
 9       0.1605   0.1439   152.56  0.0000
 10      0.1366   0.0396   158.87  0.0000
 11      0.2545   0.1471   180.86  0.0000
 12      0.2994   0.1845   211.38  0.0000

.
```

It is clearly evident that the series is quite persistent given that it is already in percentage change form. The autocorrelation function dies away rather slowly. Only the first two partial autocorrelation coefficients appear strongly significant. The numerical values of the autocorrelation and partial autocorrelation coefficients at lags 1–12 are given in the columns (2) and (3) of the output, with the lag length given in column (1).[49]

Remember that as a rule of thumb, a given autocorrelation coefficient is classed as significant if it is outside a $\pm 1.96 \times 1/(T)^{1/2}$ band, where T is the number of observations. In this case, it would imply that a correlation coefficient is classed as significant if it is bigger than approximately 0.11 or smaller than –0.11. The band is of course wider when the sampling frequency is monthly, as it is here, rather than daily where there would be more observations. It can be deduced that the first six autocorrelation coefficients (then nine through twelve) and the first two partial autocorrelation coefficients (then nine, eleven, and twelve) are significant under this rule. Since the first acf coefficient is highly significant, the joint test statistic presented in column (4) rejects the null hypothesis of no autocorrelation at the 1% level for all numbers of lags considered. It could be concluded that a mixed ARMA process might be appropriate, although it is hard to precisely determine the appropriate order given these results. In order to investigate this issue further, information criteria are now employed.

11.2 Using Information Criteria to Decide on Model Orders

An important point to note is that books and statistical packages often differ in their construction of the test statistic. For example, the formulae given in (Brooks, 2019) for Akaike's and Schwarz's Information Criteria are

$$\text{AIC} = \ln(\hat{\sigma}^2) + \frac{2k}{T} \tag{2}$$

$$\text{SBIC} = \ln(\hat{\sigma}^2) + \frac{k}{T}\ln(T) \tag{3}$$

where $\hat{\sigma}^2$ is the estimator of the variance of regressions disturbances u_t, k is the number of parameters and T is the sample size. When using the criterion based on the estimated standard errors, the model with the lowest value of AIC and SBIC should be chosen. On the other hand, Stata uses a formulation

[49] The right-hand side of the output measures the coefficients on a scale from –1 to 1 and gives a quick indication which lags are significantly large.

of the test-statistic based on maximum likelihood estimation. The corresponding Stata formulae are

$$\text{AIC}_L = -2 \cdot \ln(L) + 2k \tag{4}$$
$$\text{SBIC}_L = -2 \cdot \ln(L) + k \ln(T), \tag{5}$$

where $\ln(L)$ is the maximised log-likelihood of the model.

Unfortunately, this modification is not benign, since it affects the relative strength of the penalty term compared with the error variance, sometimes leading different packages to select different model orders for the same data and criterion!

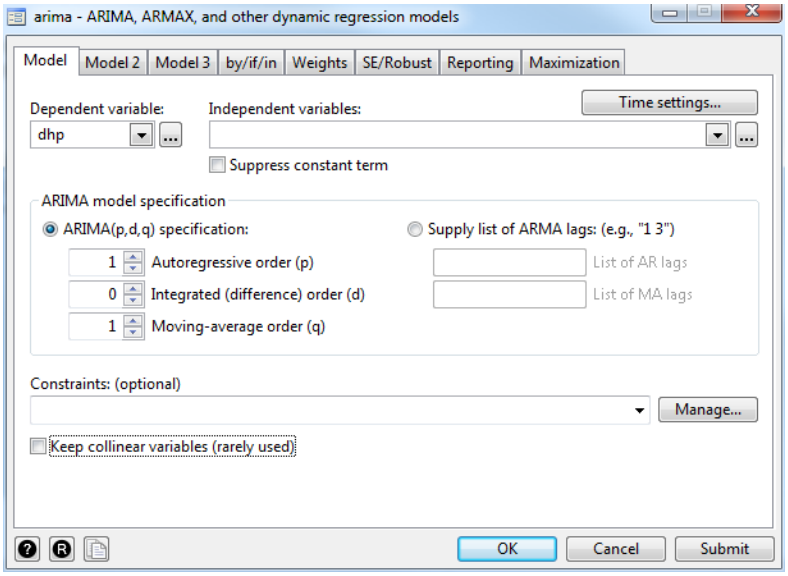


Figure 58: Specifying an ARMA(1,1) Model

Suppose it is thought that ARMA models from order (0,0) to (5,5) are plausible for the house price changes. This would entail considering 36 models (ARMA(0,0), ARMA(1,0), ARMA(2,0), ..., ARMA(5,5)), i.e., from zero up to 5 lags in both the autoregressive and moving average terms.

In Stata, this can be done by separately estimating each of the models and noting down the value of the information criteria in each case. We can do this in the following way. From the Stata main menu, we click on **Statistics/Time series** and select **ARIMA and ARMAX models**. In the specification window that appears we select **Dependent variable: dhp** and leave the dialogue box for the independent variables empty as we only want to include autoregressive and moving-average terms but no other explanatory variables. We are then asked to specify the ARMA model. There are three boxes in which we can type in the number of either the autoregressive order (p), the integrated (difference) order (d) or the moving-average order (q). As we want to start with estimating an **ARMA(1,1)** model, i.e., a model of **autoregressive order 1** and **moving-average order 1**, we specify this in the respective boxes (Figure 58). An equivalent formulation would be:

$$\Delta HP_t = \mu + \varphi \Delta HP_{t-1} + \theta \epsilon_{t-1} + \epsilon_t, \tag{6}$$

with mean μ, AR coefficient φ and MA coefficient θ. We click **OK** and Stata generates the following estimation output.

```
. arima dhp, arima(1,0,1)

(setting optimization to BHHH)
Iteration 0:   log likelihood = -468.79263
Iteration 1:   log likelihood = -463.50762
Iteration 2:   log likelihood = -462.78694
Iteration 3:   log likelihood = -462.71518
Iteration 4:   log likelihood = -462.71056
(switching optimization to BFGS)
Iteration 5:   log likelihood = -462.71002
Iteration 6:   log likelihood = -462.70995
Iteration 7:   log likelihood = -462.70995

ARIMA regression

Sample:  1991m2 - 2018m3                        Number of obs     =        326
                                                Wald chi2(2)      =     378.32
Log likelihood =   -462.71                      Prob > chi2       =     0.0000

------------------------------------------------------------------------------
             |                 OPG
         dhp |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
dhp          |
       _cons |   .4285885   .1447873     2.96   0.003     .1448105    .7123664
-------------+----------------------------------------------------------------
ARMA         |
          ar |
          L1. |   .8223724   .0547301    15.03   0.000     .7151034    .9296414
             |
          ma |
          L1. |  -.5417254   .0858423    -6.31   0.000    -.7099731   -.3734777
-------------+----------------------------------------------------------------
       /sigma |   .9999633   .0340591    29.36   0.000     .9332086    1.066718
------------------------------------------------------------------------------
Note: The test of the variance against zero is one sided, and the two-sided
      confidence interval is truncated at zero.

.
```

In theory, the output would be discussed in a similar way to the simple linear regression model discussed in Section 3 of this guide. However, in reality it is very difficult to interpret the parameter estimates in the sense of, for example, saying 'a 1 unit increase in x leads to a β unit increase in y'. In part because the construction of ARMA models is not based on any economic or financial theory, it is often best not to even try to interpret the individual parameter estimates, but rather to examine the plausibility of the model as a whole, and to determine whether it describes the data well and produces accurate forecasts (if this is the objective of the exercise, which it often is). Note also that the header of the Stata output for ARMA models states the number of iterations that have been used in the model estimation process. This shows that, in fact, an iterative numerical optimisation procedure has been employed to estimate the coefficients. Stata also automatically adjusts the computation of standard errors according to the optimisation technique used. As we use the Berndt–Hall-Hall-Hausman algorithm which is the default method ('bhhh'), the reported standard errors are Outer product gradient (OPG) errors.

In order to generate the information criteria corresponding to the ARMA(1,1) model we open the 'Postestimation Selector' (**Statistics/Postestimation**) and select **Specification, diagnostic, and goodness-of-fit analysis** (Figure 59, left panel). In the specification window, the correct subcommand **information criteria (ic)** is already pre-selected (Figure 59, right panel). We can specify the number of observations for calculating the SBIC (Schwartz criterion), though we keep the default option which is all 268 observations.

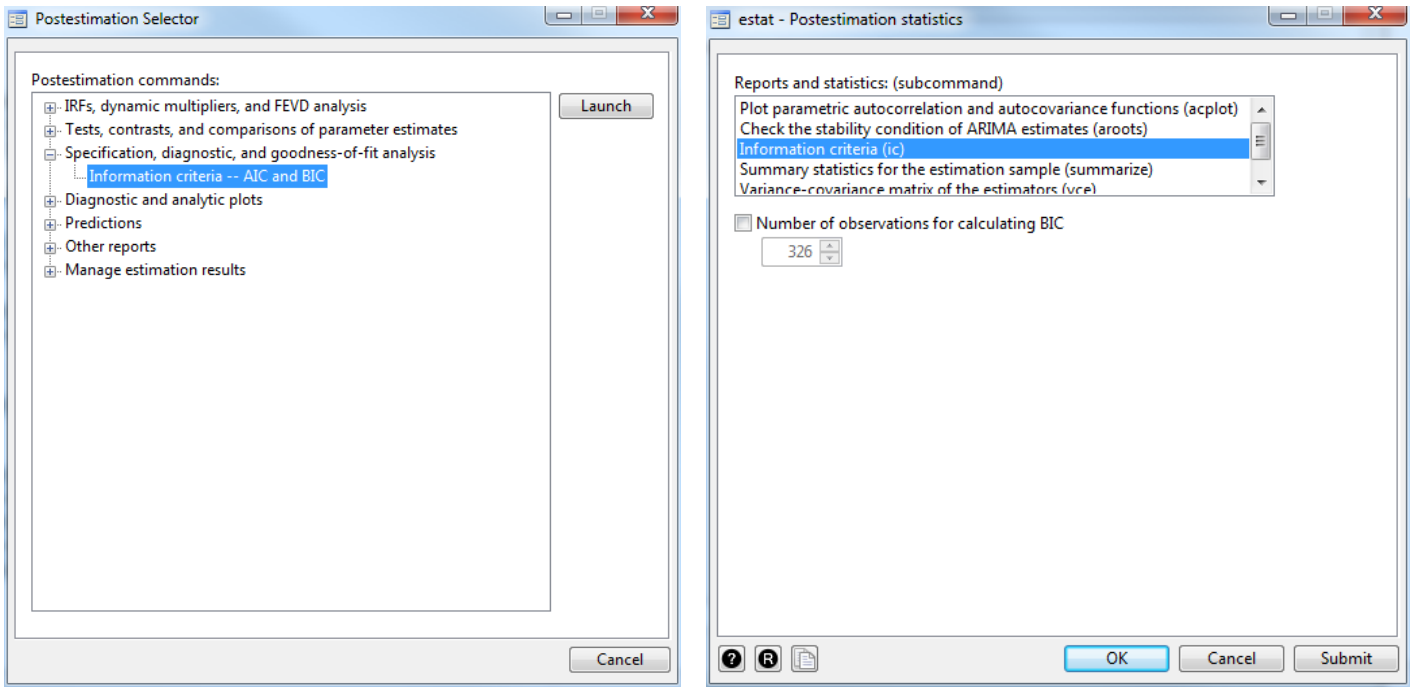


Figure 59: Generating Information Criteria for the ARMA(1,1) Model

By clicking **OK** the following test statistics are generated.

```
. estat ic

Akaike's information criterion and Bayesian information criterion

-----------------------------------------------------------------------------
       Model |        Obs  ll(null)  ll(model)      df         AIC        BIC
-------------+---------------------------------------------------------------
           . |        326         .   -462.71       4    933.4199   948.5675
-----------------------------------------------------------------------------
               Note: N=Obs used in calculating BIC; see [R] BIC note.

.
```

We see that the AIC has a value of 933.42 and the BIC a value of 948.57. However, by themselves these two statistics are relatively meaningless for our decision as to which ARMA model to choose. Instead, we need to generate these statistics for the competing ARMA models and then select the model with the lowest information criterion.

To check that the process implied by the model is stationary and invertible, it is useful to look at the inverses of the AR and MA roots of the characteristic equation. If the inverse roots of the AR polynomial all lie inside the unit circle, the process is stationary, invertible, and has an infinite-order moving-average (MA) representation. We can test this by selecting **Diagnostic and analytic plots/Check stability condition of estimates** in the 'Postestimation Selector' (Figure 60, upper left panel). In the specification window that appears, we tick the box **Label eigenvalues with the distance from the unit circle** and click **OK** (Figure 60, upper right panel). From the test output we see that the inverted roots for both the AR and MA parts lie inside the unit circle and have a distance from the circle of 0.178 and 0.458, respectively (Figure 60, bottom panel). Thus the conditions of stationarity and invertibility, respectively, are met.

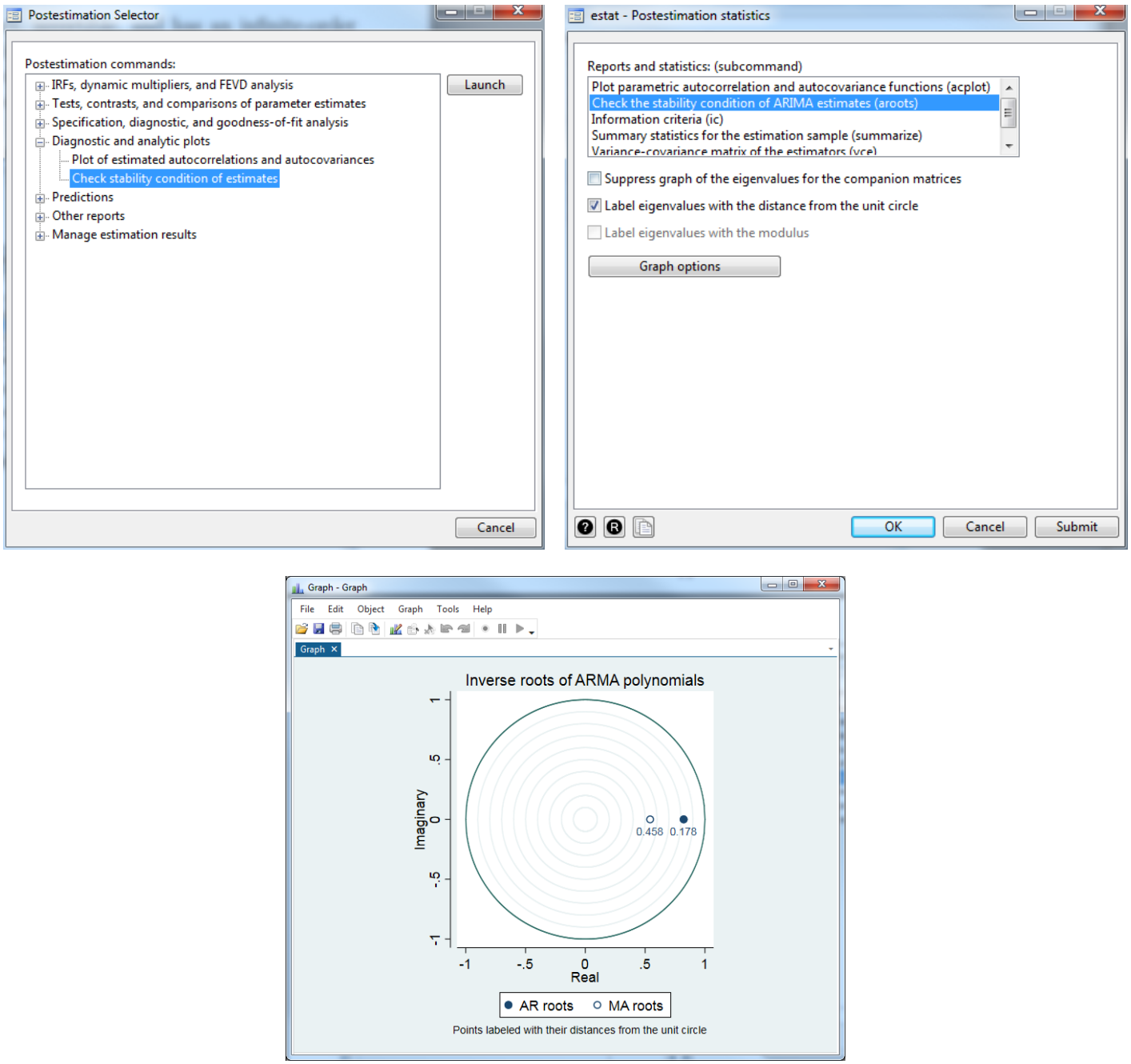


Figure 60: Testing the Stability Condition for the ARMA(1,1) Estimates

Repeating these steps for the other ARMA models would give all of the required values for the information criteria. To give just one more example, in the case of an ARMA(5,5), the following would be typed in the ARMA specification window (Figure 61).[50]

[50]For more information on how to specify ARMA and ARIMA models in Stata, refer to the respective entry in the Stata manual.

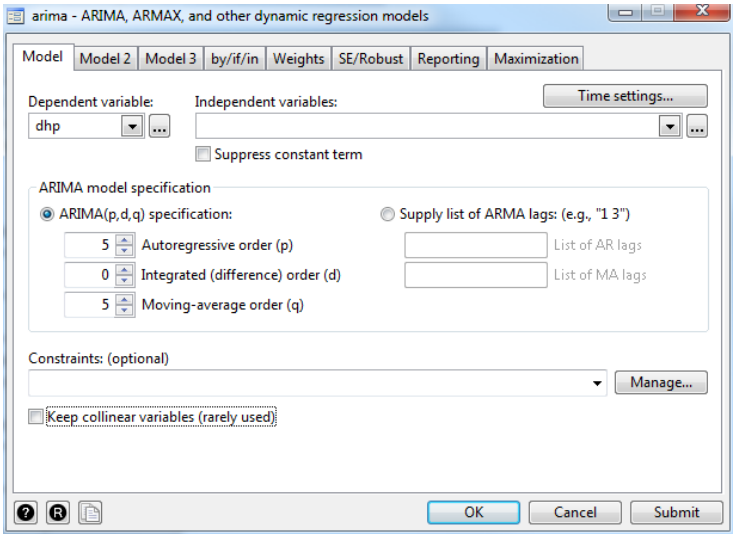


Figure 61: Specifying an ARMA(5,5) Model

Again, we need to generate the information criteria by selecting **Specification, diagnostic, and goodness-of-fit analysis**. Table 2 reports values of the information criteria for all the competing ARMA models, calculated via Stata.[51]

Table 2: Information Criteria for ARMA(p,q) Models

AIC

p/q	0	1	2	3	4	5
0	1001.2637	977.40202	935.50826	931.45446	930.29495	929.95052
1	958.79144	933.4199	925.64874	923.78884	924.49989	929.30722
2	922.46005	924.40804	926.2427	924.07266	925.97743	922.58878
3	924.40159	924.63688	926.35538	925.95741	925.4296	923.84556
4	926.26105	926.27666	925.35727	929.26543	**914.85569**	918.0929
5	928.24536	925.9181	929.4992	921.68207	916.10621	919.801

SBIC

p/q	0	1	2	3	4	5
0	1008.8375	988.76271	950.65585	950.38895	953.01633	956.4588
1	970.15214	948.56749	944.58323	946.51022	951.00817	959.6024
2	**937.60764**	943.34253	948.96408	950.58094	956.27261	956.67086
3	943.33607	947.35826	952.86366	956.25259	959.51168	961.71453
4	948.98243	952.78495	955.65245	963.34751	952.72467	959.74877
5	954.75364	956.21327	963.58128	959.55105	957.76208	965.24377

So which model actually minimises the two information criteria? In this case, the criteria choose different models: *AIC* selects an ARMA(4,4), while *SBIC* selects the smaller ARMA(2,0) model. These chosen models are highlighted in bold in the table. It will always be the case that *SBIC* selects a model that is at least as small (i.e., with fewer or the same number of parameters) as *AIC*, because the former criterion has a stricter penalty term. This means that *SBIC* penalises the incorporation of additional

[51]Note that these results can be calculated quicker using a do-File, which is a small program. We will discuss do-Files later in this guide.

terms more heavily. Many different models provide almost identical values of the information criteria, suggesting that the chosen models do not provide particularly sharp characterisations of the data and that a number of other specifications would fit the data almost as well.

12 Forecasting Using ARMA Models

Reading: Brooks (2019, Section 6.8)

Suppose that an AR(2) model selected for the house price percentage changes series was estimated using observations February 1991–December 2015, leaving 27 remaining observations to construct forecasts for and to test forecast accuracy (for the period January 2016–March 2018).

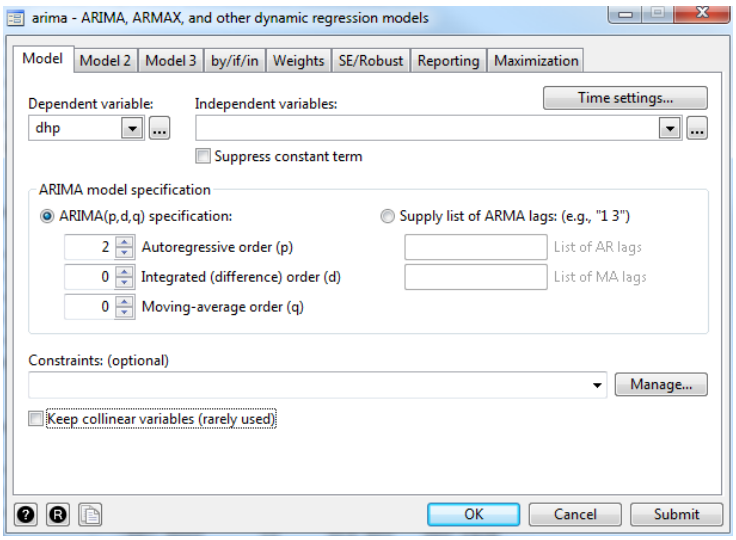


Figure 62: Specifying an ARMA(2,0) Model

Let us first estimate the ARMA(2,0) model for the time period 1991m2 – 2015m12. The specification window for estimating this model resembles Figure 62. We select **2** as the **Autoregressive order (p)** and leave the other model parts as zero. As we only want to estimate the model over a subperiod of the data, we next select the tab **by/if/in** (Figure 63). In the dialogue box **If: (expression)** we type in **month<=tm(2015m12)**.

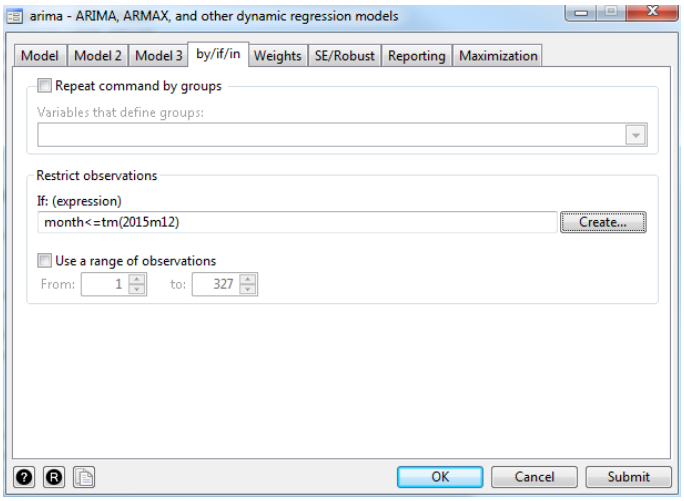


Figure 63: Restricting the Sample to the Subperiod 1991m2 - 2015m12

Next, we click **OK** and the following regression output should appear in the *Output* window.

```
. arima dhp if month <= tm(2015m12), arima(2,0,0)

(setting optimization to BHHH)
Iteration 0:   log likelihood = -427.95559
Iteration 1:   log likelihood = -427.95229
Iteration 2:   log likelihood = -427.95193
Iteration 3:   log likelihood = -427.95189
Iteration 4:   log likelihood = -427.95188

ARIMA regression

Sample:  1991m2 - 2015m12                       Number of obs     =        299
                                                Wald chi2(2)      =      91.31
Log likelihood = -427.9519                      Prob > chi2       =     0.0000

------------------------------------------------------------------------------
             |                 OPG
         dhp |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
dhp          |
       _cons |   .4417418   .1395683     3.17   0.002      .168193    .7152907
-------------+----------------------------------------------------------------
ARMA         |
          ar |
         L1. |   .2353281   .0470072     5.01   0.000     .1431957    .3274604
         L2. |   .3405194   .0451424     7.54   0.000      .252042    .4289968
-------------+----------------------------------------------------------------
      /sigma |   1.011752    .037069    27.29   0.000     .9390976    1.084406
------------------------------------------------------------------------------
Note: The test of the variance against zero is one sided, and the two-sided
      confidence interval is truncated at zero.

.
```

Now that we have fitted the model we can produce the forecasts for the period 2016m1 to 2018m3. There are two methods available in Stata for constructing forecasts: dynamic and static. The option **Dynamic** calculates multi-step forecasts starting from the first period in the forecast sample. **Static** forecasts imply a sequence of one-step-ahead forecasts, rolling the sample forwards one observation after each forecast.

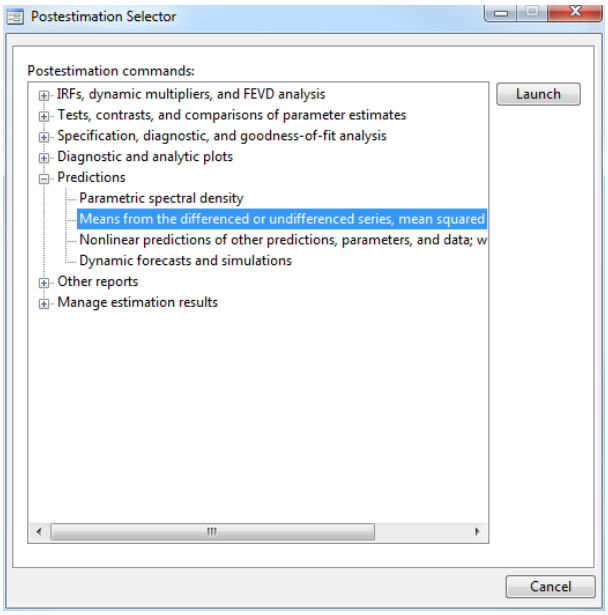


Figure 64: Postestimation Selector to Generate Predictions Based on an ARMA Model

We start with generating static forecasts. These forecasts can be generated by opening the 'Postestimation Selector' and choosing **Predictions/Means from the differenced or undifferenced series, mean squared errors, residuals, etc.** (Figure 64).

In the 'predict' specification window, we are first asked to name the variable that will contain the predictions (Figure 65, upper left panel). We choose to name the static forecasts **dhpf_stat**. As we want to create predicted/fitted values, we keep the default option under **Produce** which is **Values for mean equation**. If we change to the **Options** tab now, the window should resemble that in Figure 65, upper right panel. We see that the option **One-step prediction** is selected as default, so we do not need to make any changes at this stage. We simply click **OK** and we should find the new series appearing in our **Variables** window.

We create the dynamic forecasts in a similar way. First we open the 'predict' specification window again, and name the series that should contain the dynamic forecasts **dhpf_dyn**. Then we change to the **Options** tab. Here we select the option **Switch to dynamic predictions at time:** and we specify the time as **tm(2016m1)**, i.e., the start of the forecast period. By clicking **OK**, we generate the series of dynamic forecasts (Figure 65, bottom panel).

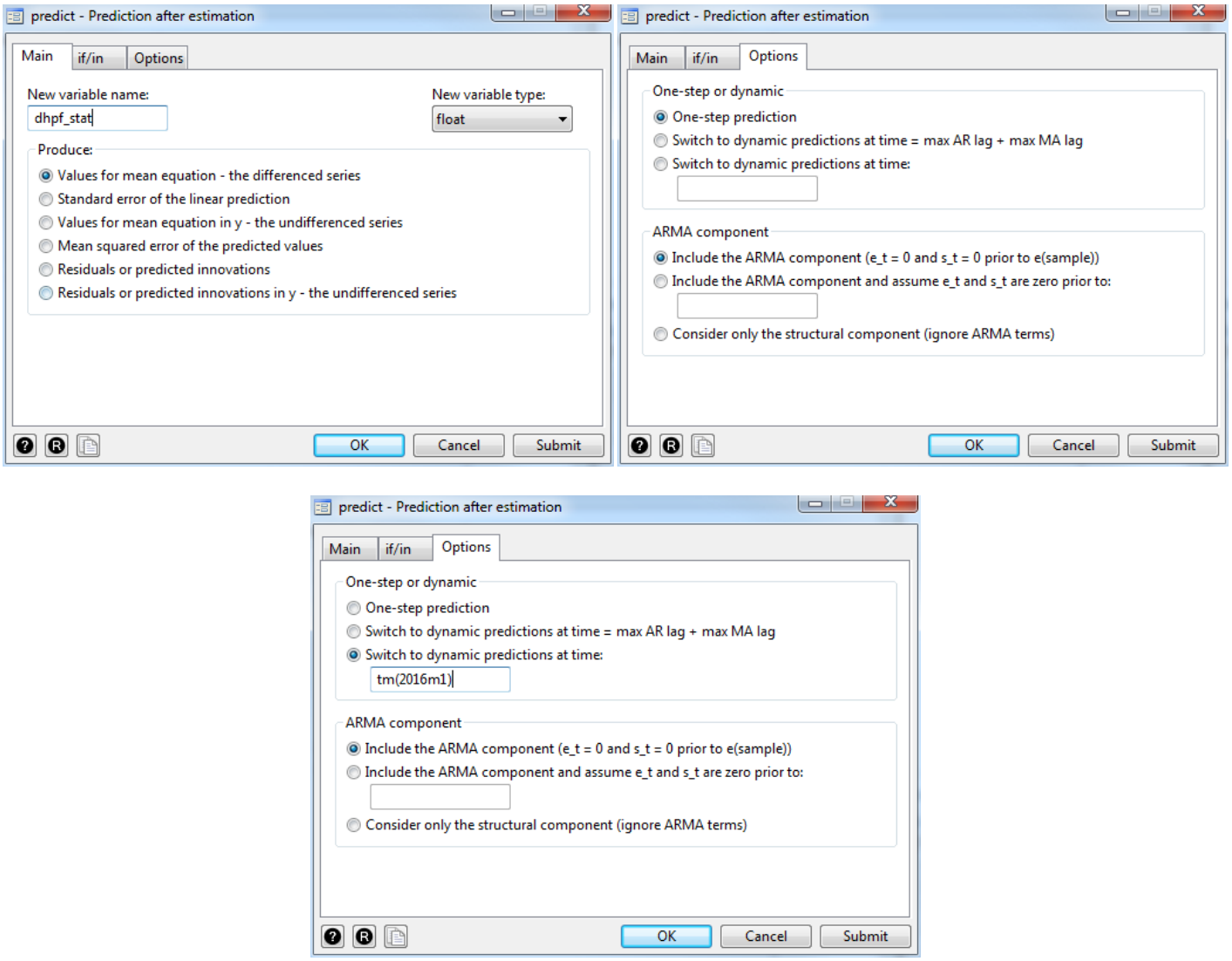


Figure 65: Generating Static and Dynamic Forecasts

To spot differences between the two forecasts and to compare them to the actual values of the changes in house prices that were realised over this period, it is useful to create a graph of the three series. To

do so, we click on **Graphics/Time-series graphs/Line plots**. We click on **Create...** and select the variable 'dhp'. We can format this plot by clicking on **Line properties**. We want this series to be plotted in blue so we select this colour from the drop-down box. We then return to the main specification window and create another plot of the series 'dhpf_stat'. Let us format this series as **Red** and **Dash**. Finally, we create a plot for 'dhpf_dyn' for which we choose the format **Green** and **Dash**. As we only want to observe the values for the forecast period, we change to the **if/in** tab and restrict the observations to those beyond December 2015 by typing **month>tm(2010m12)** into the dialogue box. If the graph is correctly specified, it should look like Figure 66.

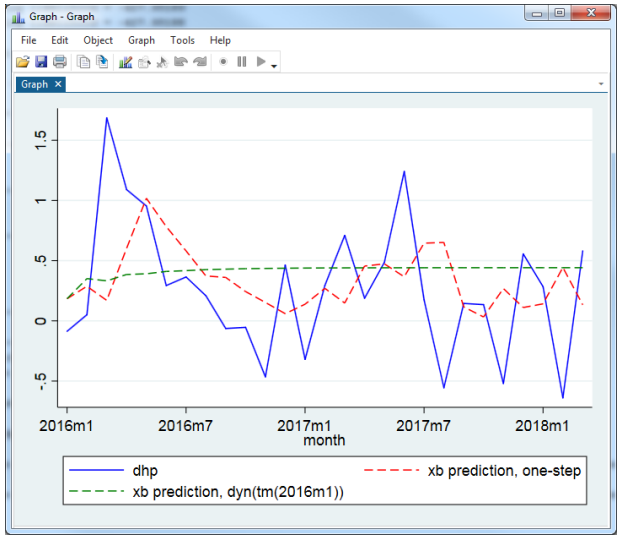


Figure 66: Graph Comparing the Static and Dynamic Forecasts with the Actual Series

Let us have a closer look at the graph. For the dynamic forecasts, it is clearly evident that the forecasts quickly converge upon the long-term unconditional mean value as the horizon increases. Of course, this does not occur with the series of 1-step-ahead forecasts which seem to more closely resemble the actual 'dhp' series.

A robust forecasting exercise would of course employ a longer out-of-sample period than the two years or so used here, would perhaps employ several competing models in parallel, and would also compare the accuracy of the predictions by examining the forecast error measures, such as the square root of the mean squared error (RMSE), the MAE, the MAPE, and Theil's U-statistic. Unfortunately, there is no built-in function in Stata to compute these statistics so they would need to be created manually by generating new data series for each of the statistics.[52]

[52] You can find the formulae to generate the forecast error statistics in Brooks (2019).

13 Estimating Exponential Smoothing Models

Reading: Brooks (2019, Section 6.9)

Stata allows us to estimate exponential smoothing models as well. To do so, we click on **Statistics/Time series/Smoothers/univariate forecasters** and then select **Single-exponential smoothing**. As you can see from the other options under **Smoothers/univariate forecasters**, there is a variety of smoothing methods available, including single and double, or various methods to allow for seasonality and trends in the data. However, since single-exponential smoothing is the only smoothing method discussed in the textbook, we will focus on this.

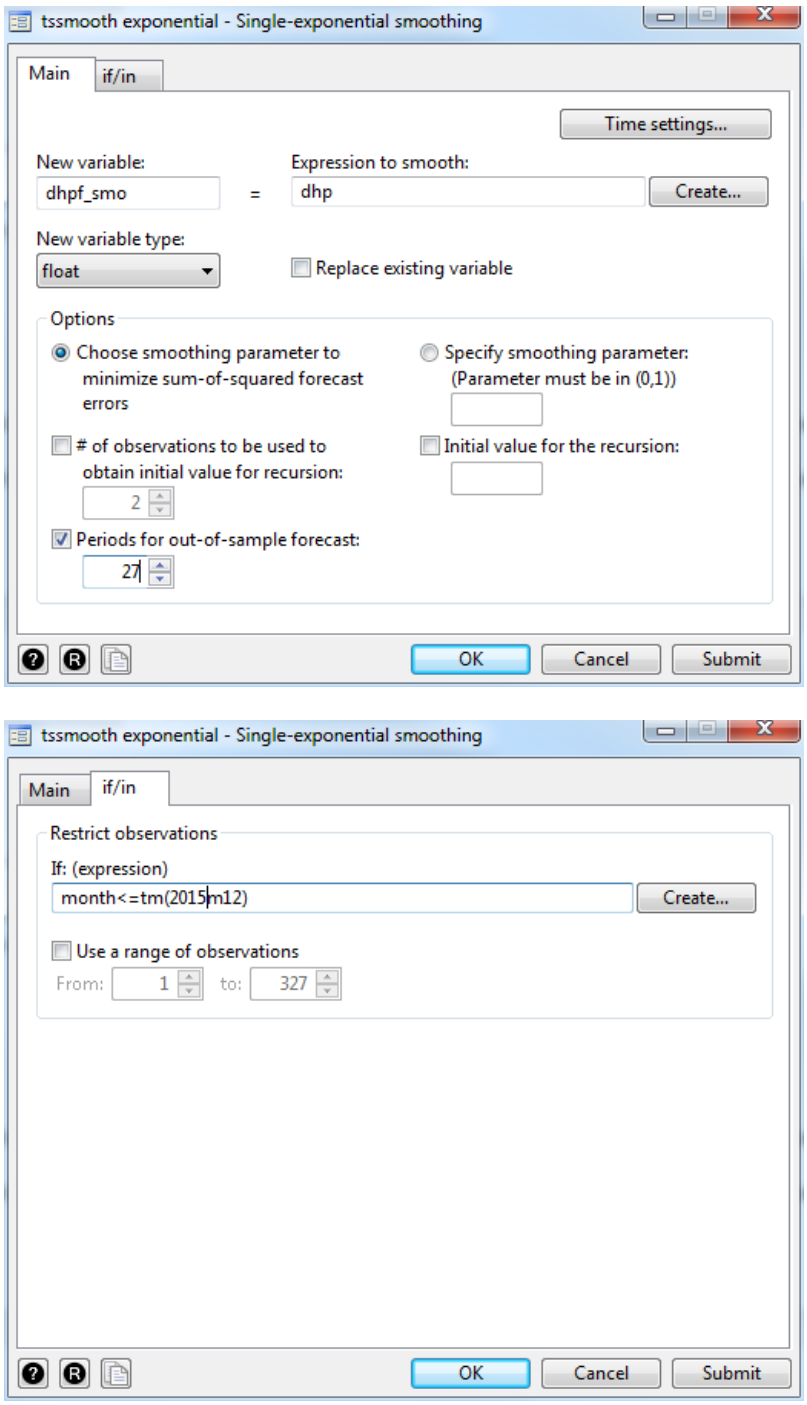


Figure 67: Specifying an Exponential Smoothing Model

In the specification window that appears, we first have to name the new variable that will contain the smoothed forecasts (Figure 67, upper panel). We type in **dhpf_smo** as **New Variable**. Next, we specify that the **Expression to smooth** is **dhp**. Our estimation period is 1991m1 - 2015m12 which we define by changing to the **if/in** tab and **Restrict observations** to **month<=tm(2015m12)** as shown in Figure 67, lower panel. This leaves 27 observations for out-of-sample forecasting. We return to the main tab and specify the **Periods for out-of-sample forecast: 27**. Clicking **OK** will give the results shown below.

```
. tssmooth exponential dhpf_smo = dhp if month<=tm(2015m12), forecast(27)

computing optimal exponential  coefficient (0,1)

optimal exponential coefficient =       0.2383
sum-of-squared residuals        =     335.38452
root mean squared error         =     1.0590974

.
```

The output includes the value of the estimated smoothing coefficient (0.2383 in this case), together with the sum-of-squared residuals (RSS) for the in-sample estimation period and the root mean squared error (RMSE) for the whole forecast. The final in-sample smoothed value will be the forecast for those 27 observations (which in this case would be 0.2501414). You can find these forecasts by changing to the *Data Editor* view and scrolling down to the observations for 2016m1 - 2018m3.[53]

14 Simultaneous Equations Modelling

Reading: Brooks (2019, Sections 7.5 – 7.9)

What is the relationship between inflation and stock returns? Clearly, they ought to be simultaneously related given that the rate of inflation will affect the discount rate applied to cashflows and therefore the value of equities, but the performance of the stock market may also affect consumer demand and therefore inflation through its impact on householder wealth (perceived or actual).

This simple example uses the same macroeconomic data as used previously **('macro.dta')** to estimate this relationship simultaneously. Suppose (without justification) that we wish to estimate the following model, which does not allow for dynamic effects or partial adjustments and does not distinguish between expected and unexpected inflation:

$$inflation_t = \alpha_0 + \alpha_1 returns_t + \alpha_2 dcredit_t + \alpha_3 dprod_t + \alpha_4 dmoney + u_{1t} \quad (7)$$

$$returns_t = \beta_0 + \beta_1 dprod_t + \beta_2 dspread_t + \beta_3 inflation_t + \beta_4 rterm_t + u_{2t} \quad (8)$$

where 'returns' are stock returns – see Brooks (2019) for details.

It is evident that there is feedback between the two equations since the *inflation* variable appears in the *stock returns* equation and vice versa. Two-stage least squares (2SLS) is therefore the appropriate technique to use. To do this we need to specify a list of instruments, which would be all of the variables from the reduced form equation. In this case, the reduced form equations would be:

$$inflation = f(constant, dprod, dspread, rterm, dcredit, rterm, dmoney) \quad (9)$$

$$returns = g(constant, dprod, dspread, rterm, dcredit, rterm, dmoney) \quad (10)$$

[53] Note that these values are also dependent on the initial value of the smoothed series. By default Stata uses the mean calculated over the first half of the sample. The upper panel of Figure 67 gives the option to set a specific initial value or to use the mean over a selected number of observations.

For this example we will be using the 'macro.dta' file. We can access the 2SLS estimator by clicking on **Statistics/Endogenous covariates** and selecting **Linear regression with endogenous covariates**. In the specification window, we first need to define the **Dependent variable: inflation**. In the dialog box to the right we input all exogenous variables, i.e., **dprod dcredit dmoney** thus omitting the endogenous variable 'rsandp' from the list of regressors, as we type this variable in the **Endogenous variables** dialogue box (Figure 68, left panel).

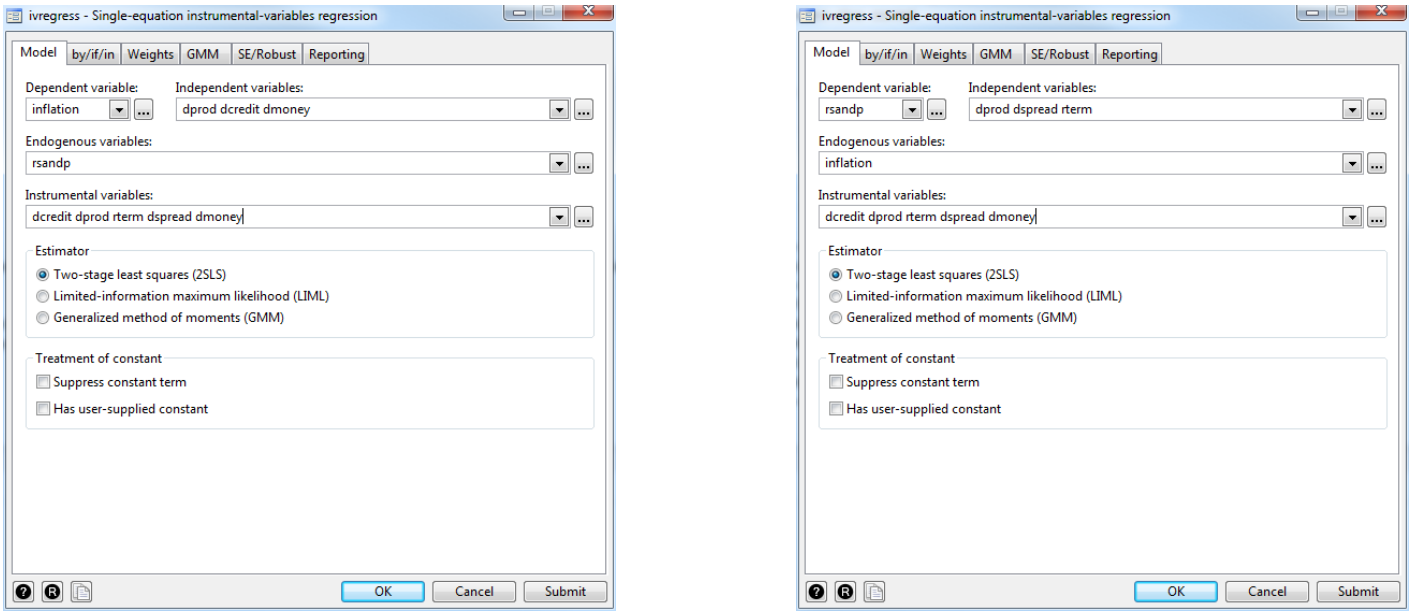


Figure 68: Specifying the 2SLS Model for the Inflation (left) and Returns (right) Equation

Finally, we are asked to define the list of **Instrumental variables**, which are **dcredit dprod rterm dspread dmoney**, i.e., the list of variables from the reduced form equation. The default estimator is already defined to be **Two-stage least squares (2SLS)** so we do not need to make any changes here. Similarly, the dialog box for the 'rsandp' equation would be specified as in Figure 68, right panel. Clicking **OK** generates the regression results. The output below shows the results form the inflation equation first, followed by the results from the returns equation.

```
. ivregress 2sls inflation dprod dcredit dmoney (rsandp = dcredit dprod rterm dspread dmoney)

Instrumental variables (2SLS) regression          Number of obs   =        384
                                                  Wald chi2(4)    =      21.78
                                                  Prob > chi2     =     0.0002
                                                  R-squared       =          .
                                                  Root MSE        =     .53922

------------------------------------------------------------------------------
   inflation |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      rsandp |   .1036744   .0333443     3.11   0.002     .0383207    .1690281
       dprod |   .0308597   .0499977     0.62   0.537     -.067134    .1288535
     dcredit |  -.0051825   .0019043    -2.72   0.007    -.0089149     -.00145
      dmoney |  -.0027873   .0010555    -2.64   0.008    -.0048561   -.0007186
       _cons |   .2129317    .036854     5.78   0.000     .1406991    .2851643
------------------------------------------------------------------------------
Instrumented:  rsandp
Instruments:   dprod dcredit dmoney rterm dspread

. ivregress 2sls rsandp dprod dspread rterm (inflation = dcredit dprod rterm dspread dmoney)

Instrumental variables (2SLS) regression          Number of obs   =        384
                                                  Wald chi2(4)    =      14.64
                                                  Prob > chi2     =     0.0055
                                                  R-squared       =          .
                                                  Root MSE        =     4.3623

------------------------------------------------------------------------------
      rsandp |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
   inflation |  -3.959215   2.816838    -1.41   0.160    -9.480117    1.561687
       dprod |  -.1590636   .4018596    -0.40   0.692     -.946694    .6285667
     dspread |  -11.73333   3.755187    -3.12   0.002    -19.09336   -4.373297
       rterm |  -.3258591   1.049526    -0.31   0.756    -2.382892    1.731174
       _cons |   1.480152   .6425509     2.30   0.021     .2207754    2.739529
------------------------------------------------------------------------------
Instrumented:  inflation
Instruments:   dprod dspread rterm dcredit dmoney

.
```

The results show that the stock index returns are a positive and significant determinant of inflation (changes in the money supply negatively affect inflation), while inflation has a negative effect on the stock market, albeit not significantly so.

It may also be of relevance to conduct a Hausman test for the endogeneity of the inflation and stock return variables. To do this, we estimate the reduced form equations and save the residuals. To simplify this step we directly type in the regression command in the **Command** window. Let us start with the inflation regression:

regress inflation dprod dspread rterm dcredit dmoney

Now we create a series of fitted values using the **predict** command. We select **Statistics/Postestimation/Predictions/Predictions and their SEs, leverage statistics, distance statistics, etc.**. In the 'predict' specification window, we call the fitted value series **inflation_fit** and we make sure that the first option **Linear prediction (xb)** is selected. By clicking **OK** the new series of fitted values should appear in the *Variables* window. We create **rsandp_fit** in a similar way. First we estimate the reduced from equation

regress rsandp dprod dspread rterm dcredit dmoney

and then we generate the fitted values using the **predict** command. Finally, we estimate the structural equations (separately), adding the fitted values from the relevant reduced form equations. The two regression commands are as follows. For the inflation equation:

regress inflation dprod dcredit dmoney rsandp rsandp_fit

and for the stock returns equation:

regress rsandp dprod dspread rterm inflation inflation_fit

The results of these regressions are presented below.

```
. regress inflation dprod dcredit dmoney rsandp rsandp_fit

      Source |       SS           df       MS      Number of obs   =       384
-------------+----------------------------------   F(5, 378)       =     14.57
       Model |  6.38090111         5  1.27618022   Prob > F        =    0.0000
    Residual |  33.1091953       378  .087590464   R-squared       =    0.1616
-------------+----------------------------------   Adj R-squared   =    0.1505
       Total |  39.4900964       383  .103107301   Root MSE        =    .29596

------------------------------------------------------------------------------
   inflation |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      dprod |   .0308597    .027442     1.12   0.261    -.0230983    .0848178
    dcredit |  -.0051825   .0010452    -4.96   0.000    -.0072377   -.0031273
    dmoney  |  -.0027873   .0005793    -4.81   0.000    -.0039264   -.0016482
     rsandp |   -.002599    .003549    -0.73   0.464    -.0095773    .0043793
rsandp_fit  |   .1062734   .0186425     5.70   0.000     .0696175    .1429293
      _cons |   .2129317   .0202279    10.53   0.000     .1731584     .252705
------------------------------------------------------------------------------

. regress rsandp dprod dspread rterm inflation inflation_fit

      Source |       SS           df       MS      Number of obs   =       384
-------------+----------------------------------   F(5, 378)       =      3.15
       Model |  288.887965         5  57.777593   Prob > F        =    0.0085
    Residual |  6943.84171       378  18.3699516   R-squared       =    0.0399
-------------+----------------------------------   Adj R-squared   =    0.0272
       Total |  7232.72967       383  18.8844117   Root MSE        =     4.286

------------------------------------------------------------------------------
      rsandp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      dprod |  -.1590636   .3948316    -0.40   0.687    -.9354051    .6172778
    dspread |  -11.73333   3.689514    -3.18   0.002    -18.98787   -4.478787
      rterm |  -.3258591   1.031171    -0.32   0.752    -2.353409    1.701691
  inflation |  -.5707632   .7616511    -0.75   0.454    -2.068367    .9268405
inflation_fit |  -3.388452   2.870468    -1.18   0.239    -9.032538    2.255633
      _cons |   1.480152   .6313136     2.34   0.020     .2388258    2.721479
------------------------------------------------------------------------------

.
```

The conclusion is that the inflation fitted value term is not significant in the stock return equation and so inflation can be considered exogenous for stock returns. Thus it would be valid to simply estimate this equation (minus the fitted value term) on its own using OLS. But the fitted stock return term is significant in the inflation equation, suggesting that stock returns are endogenous.

15 The Generalised Method of Moments

Reading: Brooks (2019, Section 14.4)

Instead of the Two-Stage Least Squares Method, the instrumental variables approach can also be realised using the Generalised Method of Moments (GMM). We therefore follow the same steps as in the previous section, but in the specification windows (Figure 69) we choose the option **Generalized method of moments (GMM)** for 'Estimator', but leaving all other options as before in Figure 68. Clicking **OK** yields the results on the next page.

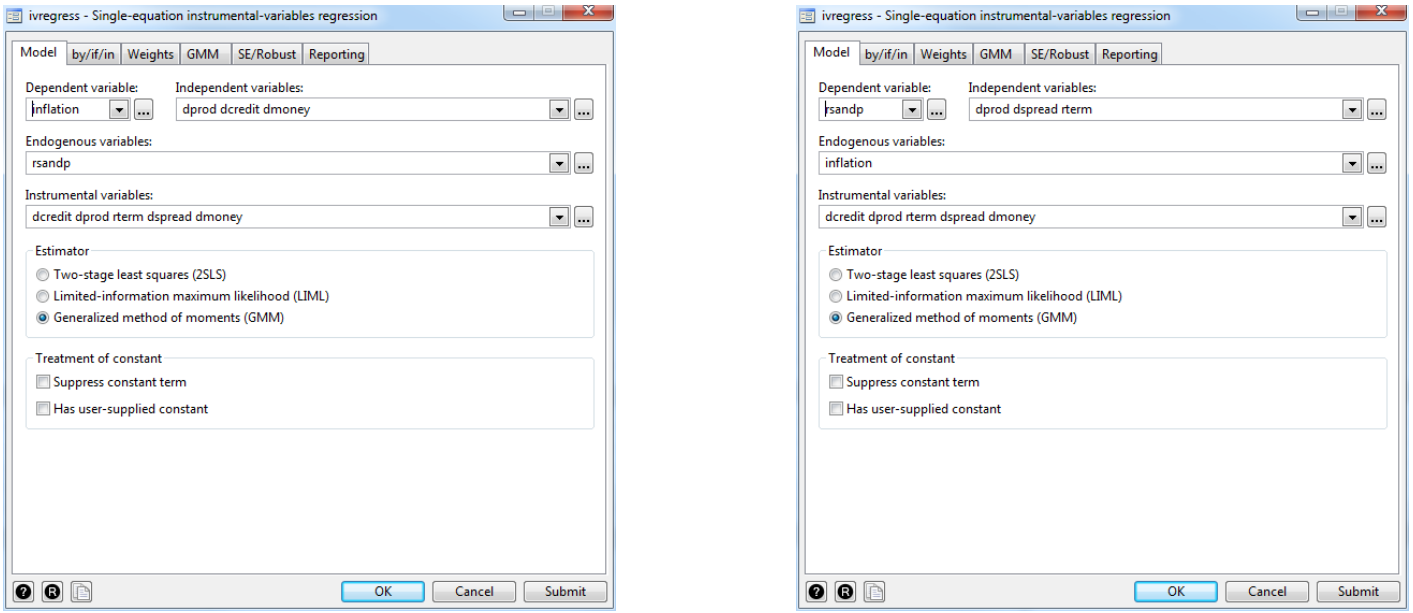


Figure 69: Specifying the Instrumental Variables Approach with GMM Estimator

Note that the results are qualitatively similar to those from the previous section, but show small differences. A look into the tab 'GMM' of the specification window in Figure 69 reveals that under default settings the GMM estimator weighs the observations and hence influences the coefficient estimates. This weighting matrix is optimal under the default setting of heteroscedastic error terms with the GMM estimator. Therefore, the deviations from the previous section are only due to specific settings.

```
. ivregress gmm inflation dprod dcredit dmoney (rsandp = dcredit dprod rterm dspread dmoney)

Instrumental variables (GMM) regression          Number of obs   =        384
                                                 Wald chi2(4)    =      18.19
                                                 Prob > chi2     =     0.0011
                                                 R-squared       =          .
GMM weight matrix: Robust                        Root MSE        =     .52816

------------------------------------------------------------------------------
             |               Robust
   inflation |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      rsandp |   .1005179   .0408627     2.46   0.014     .0204285    .1806072
       dprod |   .0267983   .0685692     0.39   0.696    -.1075949    .1611914
     dcredit |  -.0047856   .0016896    -2.83   0.005    -.0080972   -.001474
      dmoney |  -.0025196   .0010495    -2.40   0.016    -.0045765   -.0004626
       _cons |   .2017619   .0409705     4.92   0.000     .1214612    .2820626
------------------------------------------------------------------------------
Instrumented:  rsandp
Instruments:   dprod dcredit dmoney rterm dspread

. ivregress gmm rsandp dprod dspread rterm (inflation = dcredit dprod rterm dspread dmoney)

Instrumental variables (GMM) regression          Number of obs   =        384
                                                 Wald chi2(4)    =       5.33
                                                 Prob > chi2     =     0.2549
                                                 R-squared       =          .
GMM weight matrix: Robust                        Root MSE        =     4.3624

------------------------------------------------------------------------------
             |               Robust
      rsandp |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
   inflation |  -3.960072   2.776275    -1.43   0.154     -9.40147    1.481327
       dprod |  -.1590209   .6601477    -0.24   0.810    -1.452887    1.134845
     dspread |  -11.73469    5.19395    -2.26   0.024    -21.91464   -1.554733
       rterm |  -.3259688   1.195958    -0.27   0.785    -2.670003    2.018066
       _cons |   1.480315   .5916739     2.50   0.012     .3206551    2.639974
------------------------------------------------------------------------------
Instrumented:  inflation
Instruments:   dprod dspread rterm dcredit dmoney

.
```

15.1 OLS as a Special Case of GMM

In the previous subsection we used the Generalised Method of Moments to estimate an instrumental variables regression. However, we can also specify the moment conditions separately and customise the GMM estimator. For example, we can also estimate the OLS regression from the CAPM model in Equation 1 with the Generalised Method of Moments estimator `gmm`. If we click **Statistics/Endogenous covariates/Generalised method of moments estimation**, the window in the left panel of Figure 70 will pop up. By clicking **Create...** we can specify the moment equations necessary to estimate the model. For the moment restrictions equivalent to the ordinary least squares method we hence want to state that errors and explanatory variables are uncorrelated.

$$\mathbb{E}(Xu) = 0 \Leftrightarrow \mathbb{E}(X(y - \beta' X - \alpha)) = 0 \tag{11}$$

We achieve this by typing `erford -- {b0} -- {b1}*ersandp` into the window (Figure 70, right panel). Using the curly brackets around the two coefficients `b0` and `b1`, we also specify that two coefficients have to be estimated. Further, we choose `ersandp` as the independent variable.

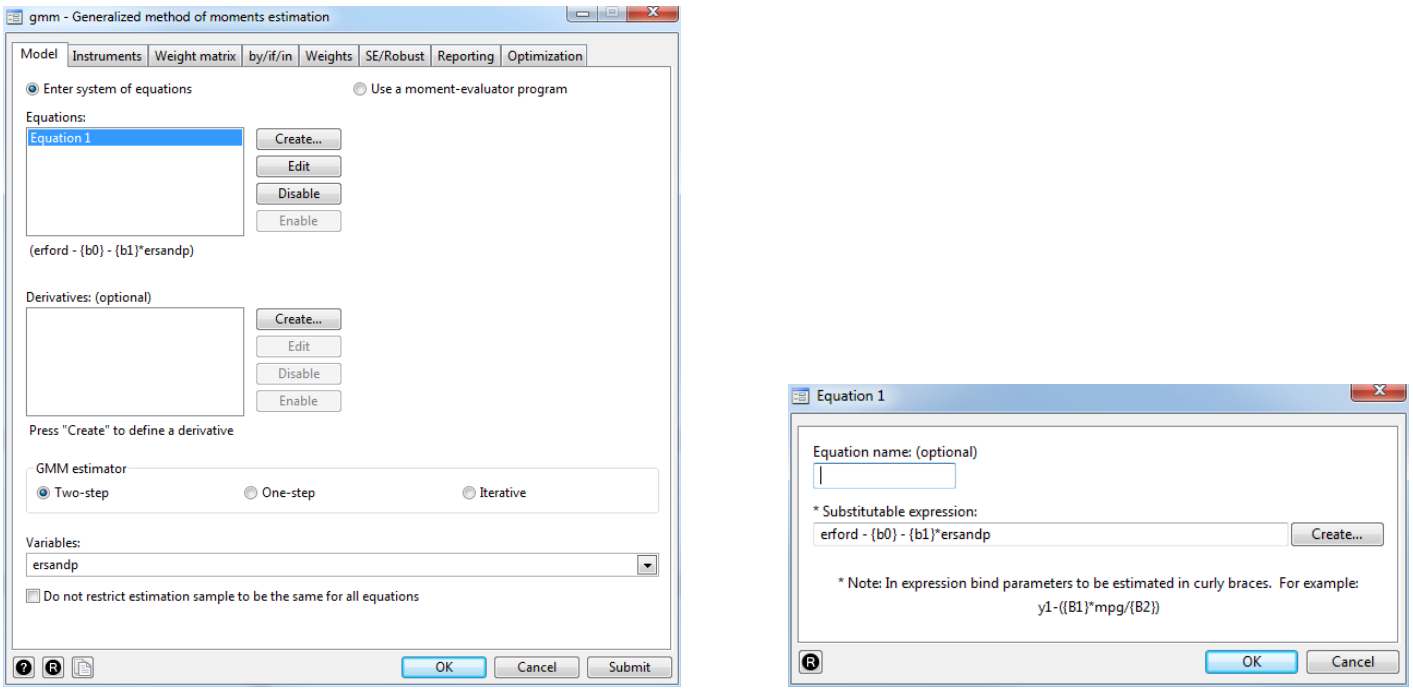


Figure 70: Estimating the CAPM Using the Generalised Method of Moments

As we do not plan to use instrumental variables in this example, we need to set the instrumental variables equal to the independent variables that we want to use. Therefore, we switch to the Tab 'Instruments' (Figure 71, left panel) and click **Create...** to choose `ersandp` as our instrument (Figure 71, right panel). Note that we do not need to specify the constant term as it is included automatically.

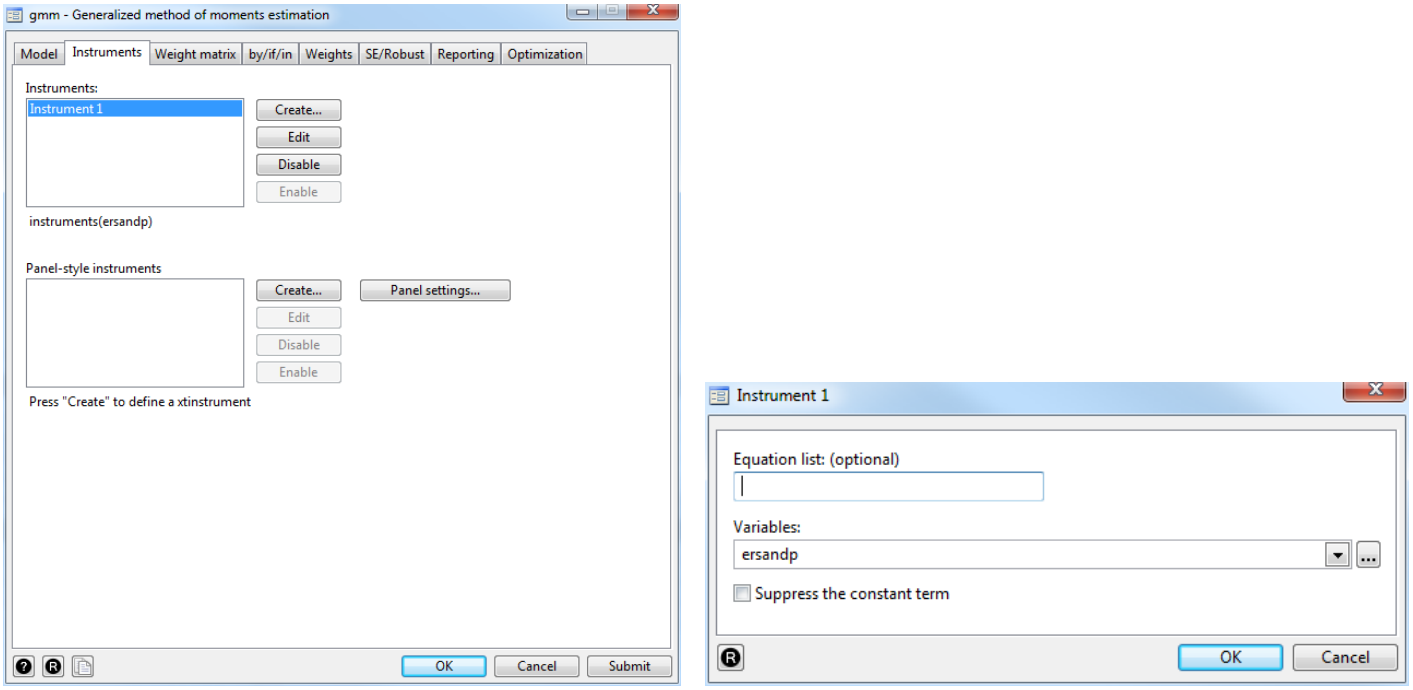


Figure 71: Estimating the CAPM Using the Generalised Method of Moments

After having specified our moment equation and the instrumental variables we can click **OK** and will be presented with the output below.

```
. gmm (erford - {b1}*ersandp- {b0}), instruments(ersandp)

Step 1
Iteration 0:   GMM criterion Q(b) =  60.820212
Iteration 1:   GMM criterion Q(b) =  1.513e-24
Iteration 2:   GMM criterion Q(b) =  6.876e-33

Step 2
Iteration 0:   GMM criterion Q(b) =  4.918e-35
Iteration 1:   GMM criterion Q(b) =  4.918e-35

note: model is exactly identified

GMM estimation

Number of parameters =   2
Number of moments    =   2
Initial weight matrix: Unadjusted                 Number of obs   =        193
GMM weight matrix:     Robust

------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          /b1 |   1.889755    .420315     4.50   0.000     1.065953    2.713558
          /b0 |  -.9559846   .8204317    -1.17   0.244    -2.564001    .6520321
------------------------------------------------------------------------------
Instruments for equation 1: ersandp _cons

.
```

Comparing the coefficients, we see that the GMM estimator finds the exact same solution. The differences in standard errors stem from the fact that the default setting uses robust standard errors, which we could adjust for in the OLS regression, too. Further, the standard errors are scaled by $\sqrt{\frac{193}{191}}$, accounting for a small sample bias.

16 Vector Autoregressive (VAR) Models

Reading: Brooks (2019, Section 7.10)

In this section, a VAR is estimated in order to examine whether there are lead–lag relationships between the returns to three exchange rates against the US dollar – the Euro, the British Pound and the Japanese Yen. The data are daily and run from 14 December 1998 to 3 July 2018, giving a total of 7,142 observations. The data are contained in the Excel file 'currencies.xls'. First, we import the dataset into Stata and **tsset Date**. Next, we construct a set of continuously compounded percentage returns called **'reur', 'rgbp' and 'rjpy'** using the following set of commands, respectively:

generate reur=100*(ln(EUR/L.EUR))

generate rgbp=100*(ln(GBP/L.GBP))

generate rjpy=100*(ln(JPY/L.JPY))

VAR estimation in Stata can be accomplished by clicking on **Statistics/Multivariate time series** and then **Vector autoregression (VAR)**. The VAR specification window appears as in Figure 72.

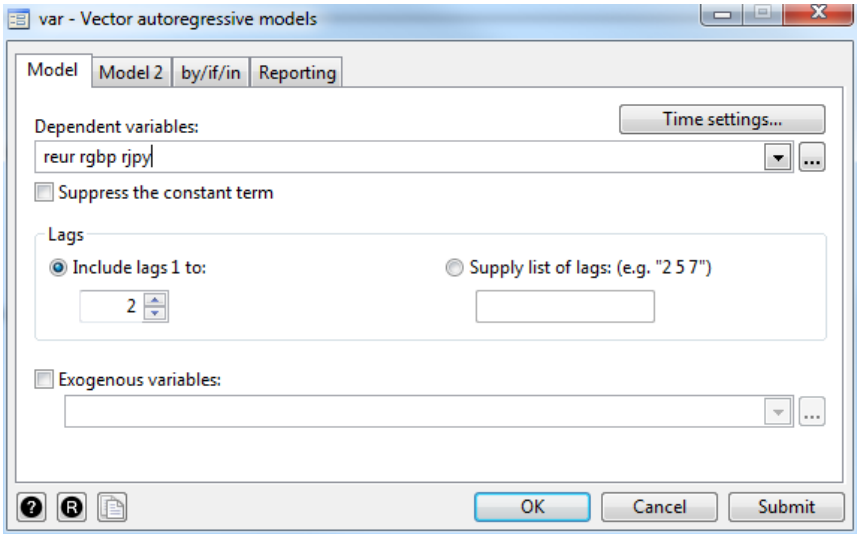


Figure 72: Specifying a VAR Model

We define the **Dependent variables** to be **reur rgbp rjpy**. Next we need to specify the number of lags to be included for each of these variables. The default is two lags, i.e., the first lag and the second lag. Let us keep the default setting for now and estimate this VAR(2) model by clicking **OK**. The regression output appears as on the next page.

```
. var reur rgbp rjpy, lags(1/2)

Vector autoregression

Sample:  12/17/1998 - 7/3/2018                  Number of obs     =      7,139
Log likelihood =  -10960.34                     AIC               =   3.076435
FPE            =   .0043517                     HQIC              =   3.083395
Det(Sigma_ml)  =   .0043261                     SBIC              =   3.096654

Equation           Parms      RMSE     R-sq      chi2     P>chi2
------------------------------------------------------------------
reur                   7     .459944   0.0182   132.6653   0.0000
rgbp                   7     .412427   0.0408   303.8789   0.0000
rjpy                   7     .466694   0.0208   151.8156   0.0000
------------------------------------------------------------------

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
reur         |
        reur |
         L1. |   .1474966     .01567     9.41   0.000     .1167839    .1782092
         L2. |  -.0118082   .0156552    -0.75   0.451    -.0424918    .0188753
             |
        rgbp |
         L1. |  -.0183557   .0170283    -1.08   0.281    -.0517307    .0150192
         L2. |   .0066229   .0170237     0.39   0.697    -.0267429    .0399886
             |
        rjpy |
         L1. |  -.0070978   .0121141    -0.59   0.558    -.0308409    .0166453
         L2. |  -.0054273   .0121137    -0.45   0.654    -.0291697    .0183151
             |
       _cons |   .0001369   .0054414     0.03   0.980    -.0105281    .0108018
-------------+----------------------------------------------------------------
rgbp         |
        reur |
         L1. |  -.0252705   .0140512    -1.80   0.072    -.0528103    .0022692
         L2. |   .0469266   .0140378     3.34   0.001     .019413     .0744403
             |
        rgbp |
         L1. |   .2213618   .0152691    14.50   0.000     .1914348    .2512888
         L2. |  -.0677941   .0152649    -4.44   0.000    -.0977128   -.0378754
             |
        rjpy |
         L1. |   -.039016   .0108625    -3.59   0.000    -.0603062   -.0177258
         L2. |   .0032869   .0108622     0.30   0.762    -.0180027    .0245765
             |
       _cons |   .0028257   .0048792     0.58   0.563    -.0067375    .0123888
-------------+----------------------------------------------------------------
rjpy         |
        reur |
         L1. |   .0410613      .0159     2.58   0.010     .0098978    .0722247
         L2. |  -.0188915   .0158849    -1.19   0.234    -.0500254    .0122424
             |
        rgbp |
         L1. |  -.0708457   .0172783    -4.10   0.000    -.1047105   -.0369809
         L2. |   .0249075   .0172735     1.44   0.149    -.0089479     .058763
             |
        rjpy |
         L1. |   .1324572   .0122918    10.78   0.000     .1083656    .1565488
         L2. |   .0149565   .0122915     1.22   0.224    -.0091343    .0390474
             |
       _cons |  -.0004126   .0055213    -0.07   0.940    -.0112341    .0104089
------------------------------------------------------------------------------

.
```

At the top of the table, we find information for the model as a whole, including information criteria, while further down we find coefficient estimates and goodness-of-fit measures for each of the equations

separately. Each regression equation is separated by a horizontal line.

We will shortly discuss the interpretation of the output, but the example so far has assumed that we know *the appropriate lag length* for the VAR. However, in practice, the first step in the construction of any VAR model, once the variables that will enter the VAR have been decided, will be to determine the appropriate lag length. This can be achieved in a variety of ways, but one of the easiest is to employ a multivariate information criterion. In Stata, this can be done by clicking on **Statistics/Multivariate time series/VAR diagnostics and tests** and selecting the first option **Lag-order selection statistics (preestimation)**. In the specification window we define the **Dependent variables: reur rgbp rjpy** (Figure 73).

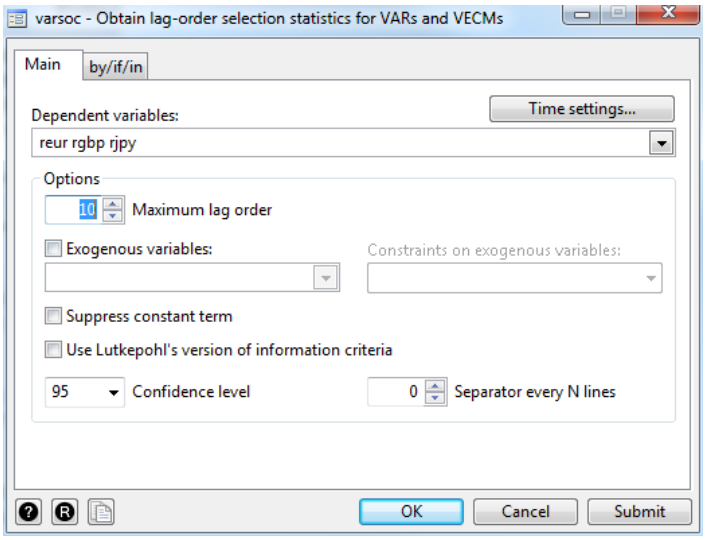


Figure 73: Selecting the VAR Lag Order Length

Then we are asked to specify the **Maximum Lag order** to entertain including in the model, and for this example, we arbitrarily select **10**. By clicking **OK** we should be able to observe the following output.

```
. varsoc reur rgbp rjpy, maxlag(10)

   Selection-order criteria
   Sample:  12/25/1998 - 7/3/2018              Number of obs      =      7131
```

lag	LL	LR	df	p	FPE	AIC	HQIC	SBIC
0	-11292.3				.004769	3.16794	3.16893	3.17083
1	-10970.4	643.81	9	0.000	.004368	3.08018	3.08416	3.09174*
2	-10947.8	45.178	9	0.000	.004351	3.07637	3.08333*	3.0966
3	-10937.4	20.867	9	0.013	.00435	3.07596	3.08592	3.10488
4	-10923.9	26.861*	9	0.001	.004344*	3.07472*	3.08766	3.11231
5	-10916.5	14.827	9	0.096	.004346	3.07517	3.09109	3.12142
6	-10911.3	10.337	9	0.324	.004351	3.07624	3.09515	3.13117
7	-10904.5	13.753	9	0.131	.004353	3.07684	3.09873	3.14044
8	-10899	10.968	9	0.278	.004358	3.07782	3.10271	3.1501
9	-10891.7	14.631	9	0.102	.00436	3.0783	3.10616	3.15925
10	-10888.6	6.1095	9	0.729	.004367	3.07996	3.11082	3.16959

```
   Endogenous:  reur rgbp rjpy
    Exogenous:  _cons

.
```

Stata presents the values of various information criteria and other methods for determining the lag order. In this case, the Akaike (AIC) and Hannan–Quinn (HQIC) criteria select a lag length of four and

two as optimal, while Schwarz's (SBIC) criterion chooses a VAR(1). Let us **estimate a VAR(1)** and examine the results. Does the model look as if it fits the data well? Why or why not?

Next, we run a Granger causality test. We click **Statistics/Multivariate time series/VAR diagnostics and tests** and select the first option **Granger causality test**. As we want to run the Granger causality test on the most recently estimated VAR(1), we can simply press **OK**.

```
. vargranger

  Granger causality Wald tests
```

Equation	Excluded	chi2	df	Prob > chi2
reur	rgbp	1.1876	2	0.552
reur	rjpy	.62657	2	0.731
reur	ALL	1.7658	4	0.779
rgbp	reur	12.895	2	0.002
rgbp	rjpy	12.935	2	0.002
rgbp	ALL	29.02	4	0.000
rjpy	reur	7.3267	2	0.026
rjpy	rgbp	17.137	2	0.000
rjpy	ALL	17.399	4	0.002

```
.
```

The results show only modest evidence of lead-lag interactions between the series. Since we have estimated a tri-variate VAR, three panels are displayed, with one for each dependent variable in the system. There is causality from EUR to GBP and from JPY to GBP that is significant at the 1% level. We also find significant causality at the 5% level from EUR to JPY and GBP to JPY, but no causality from any of the currencies to EUR. These results might be interpreted as suggesting that information is incorporated slightly more quickly in the pound-dollar rate and yen-dollar rates than into the euro-dollar rate.

After fitting a VAR, one hypothesis of interest is that all the variables at a given lag are jointly zero. We can test this in Stata using a Wald lag-exclusion test. To do so, we select **Statistics/Multivariate time series/VAR diagnostics and tests/Wald lag-exclusion statistics** and simply click **OK**. The following statistics will appear for testing the null hypothesis that the coefficient estimates of the lagged variables are jointly zero.

```
. varwle
```

Equation: reur

lag	chi2	df	Prob > chi2
1	131.8321	3	0.000
2	1.007423	3	0.799

Equation: rgbp

lag	chi2	df	Prob > chi2
1	295.9339	3	0.000
2	20.63448	3	0.000

Equation: rjpy

lag	chi2	df	Prob > chi2
1	143.5058	3	0.000
2	3.524408	3	0.318

Equation: All

lag	chi2	df	Prob > chi2
1	682.4493	9	0.000
2	45.46146	9	0.000

```
.
```

Stata obtains these test statistics for each of the three equations separately (first three panels) and for all three equations jointly (last panel). Based on the high χ^2 values for all four panels, we have strong evidence that we can reject the null that no lags should be excluded.

To obtain the impulse responses for the estimated model, we click on **Statistics/Multivariate time series/Basic VAR**. We are then presented with a specification window, as in Figure 74.

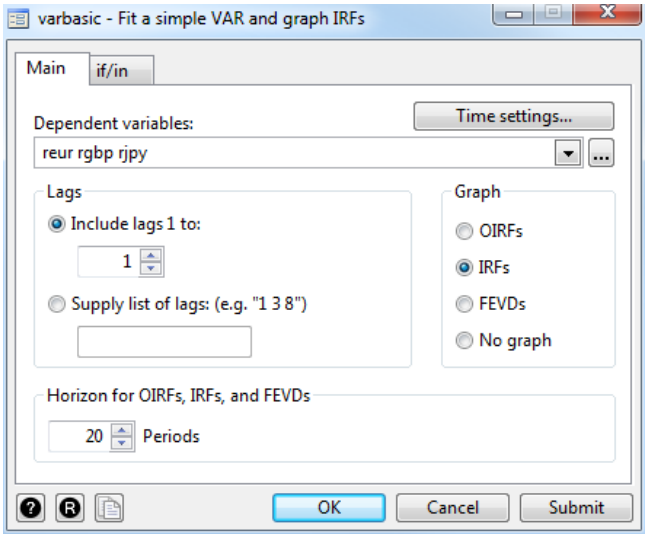


Figure 74: Generating Impulse Responses for the VAR(1) Model

We define the **Dependent variables: reur rgbp rjpy** and select a VAR model with one lag of each variable. We can do this either by specifying **Include lags 1 to: 1** or by **Supply list of lags: 1**. We then specify that we want to generate a **Graph** for the **IRFs**, that is the impulse response functions. Finally, we need to select the number of periods over which we want to generate the IRFs. We arbitrarily select **20** and press **OK**. You can see that Stata re-estimates the VAR(1) but additionally it creates a set of graphs of the implied IRFs (Figure 75).

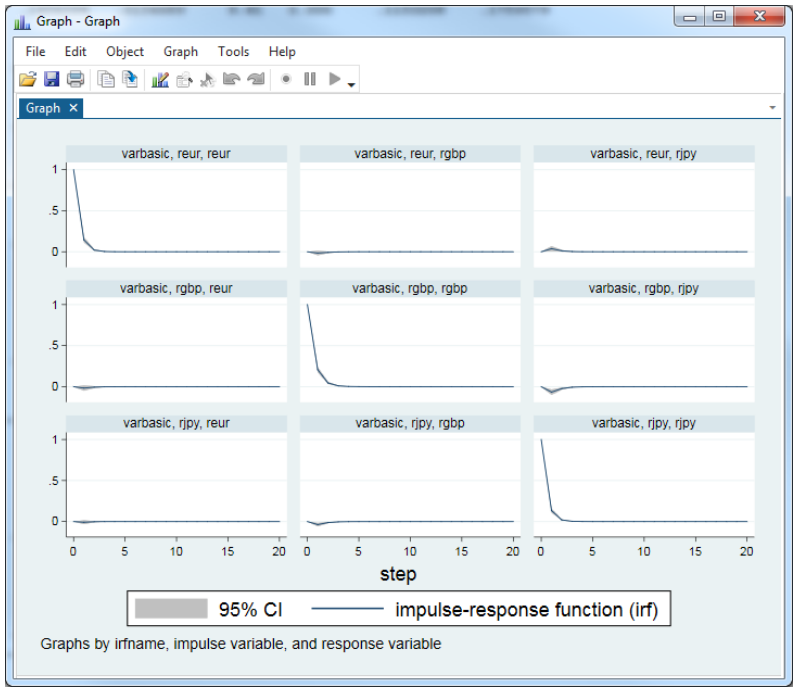


Figure 75: Graphs of Impulse Response Functions (IRFs) for the VAR(1) Model

As one would expect given the parameter estimates and the Granger causality test results, only a few linkages between the series are established here. The responses to the shocks are very small, except for the response of a variable to its own shock, and they die down to almost nothing after the first lag.

Note that plots of the variance decompositions can also be generated using the **varbasic** command. Instead of specifying the IRFs in the **Graph** selection, we choose **FEVDs**, that is the forecast-error variance decompositions. A similar plot for the variance decompositions would appear as in Figure 76.

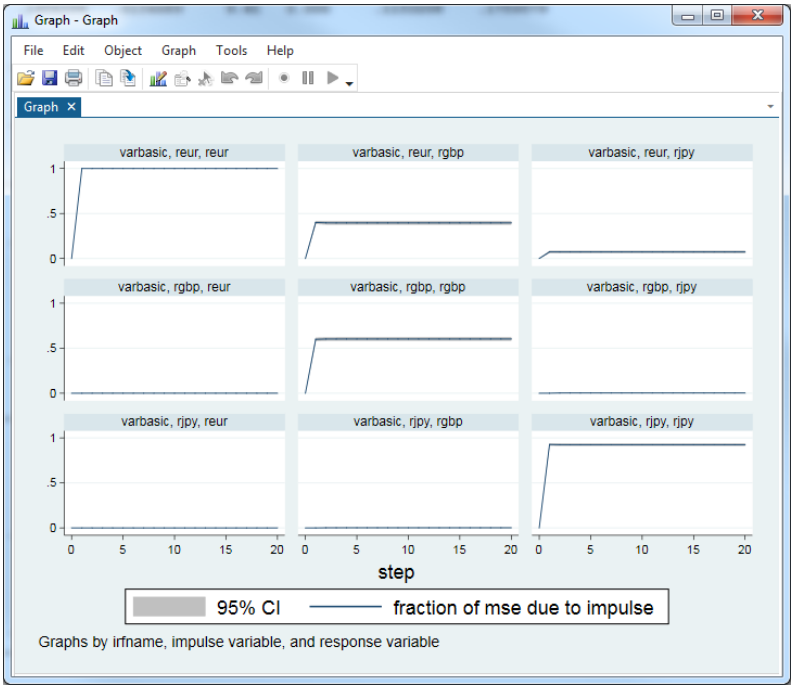


Figure 76: Graphs of FEVDs for the VAR(1) Model

There is little again that can be seen from these variance decomposition graphs apart from the fact that the behaviour is observed to settle down to a steady state very quickly. To illustrate how to interpret the FEVDs, let us have a look at the effect that a shock to the euro rates has on the other two rates and itself, which are shown in the first row of the FEVD plot. Interestingly, while the percentage of the errors that is attributable to own shocks is 100% in the case of the euro rate (top left graph), for the pound, the euro series explains around 43% of the variation in returns (top middle graph), and for the yen, the euro series explains around 7% of the variation.

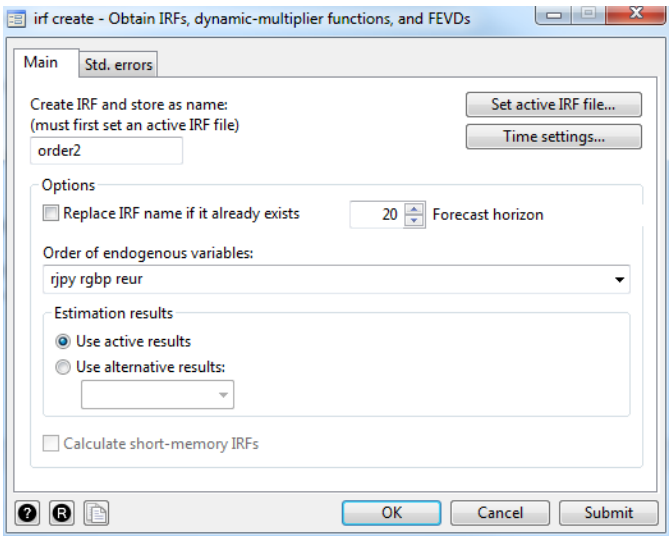


Figure 77: Generating FEVDs for an Alternative Ordering

We should remember that the ordering of the variables has an effect on the impulse responses and variance decompositions, and when, as in this case, theory does not suggest an obvious ordering of the series, some sensitivity analysis should be undertaken. Let us assume we would like to test how sensitive

the FEVDs are to a different way of ordering. We first click on **Statistics/Multivariate time series** and select **IRF and FEVD analysis/Obtain IRFs, dynamic-multiplier functions, and FEVDs**.

In the specification window that appears, we first need to **Set active IRF file...** which we do by simply clicking on the respective button and then pressing **OK** (Figure 77). You can see that in the folder where the current workfile is stored a new file appears named **_varbasic.irf** which will store all the new IRF results. However, you do not need to concern yourself with this file as Stata will automatically obtain the necessary information from the files if needed. Next we name the new IRF as **order2**. To make our results comparable, we choose **20 Forecast horizon**. Finally, we need to select the order of the variables. In this test, we choose the reverse order to that used previously, which is **rjpy rgbp reur** and click **OK** to generate the IRFs.

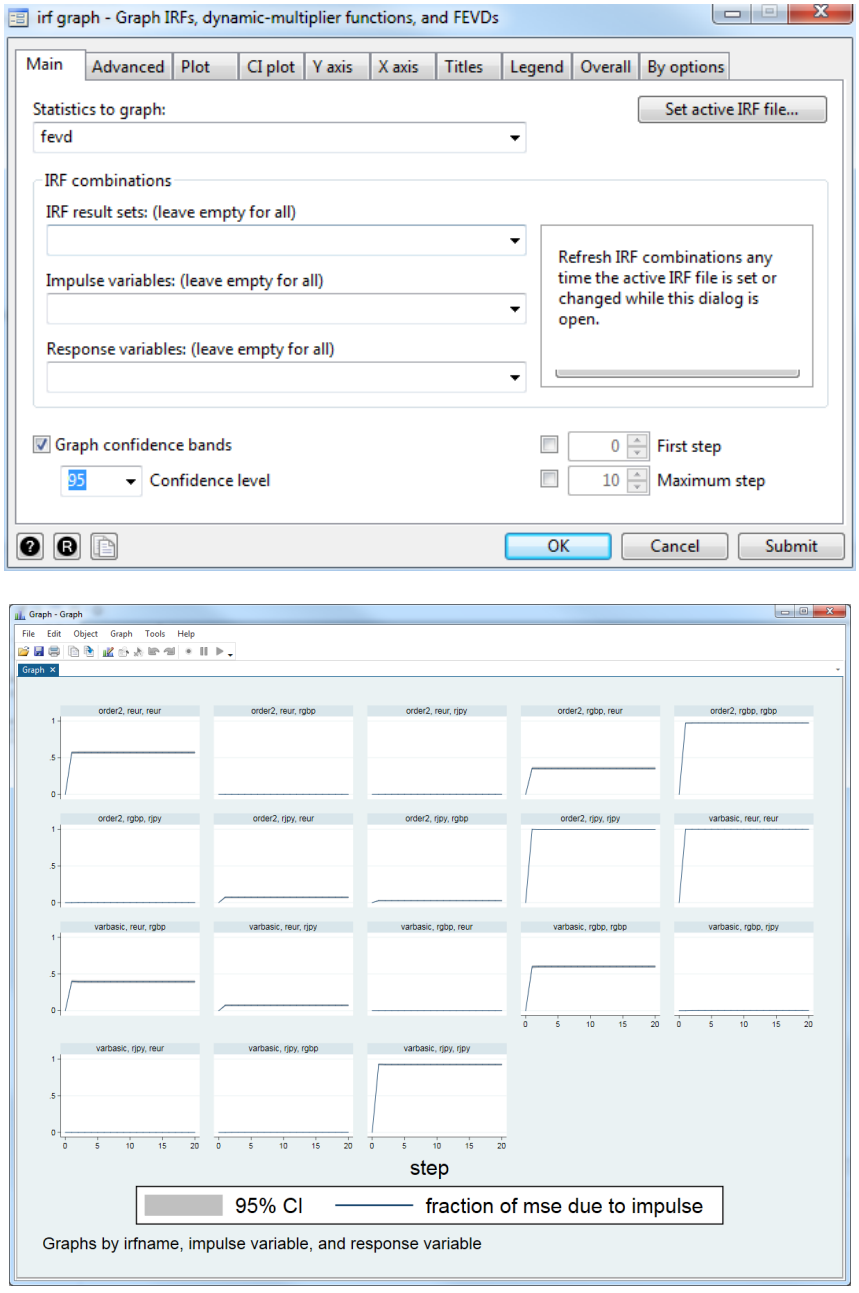


Figure 78: Graphs for FEVDs with an Alternative Ordering

To inspect and compare the FEVDs for this ordering and the previous one, we can create graphs of the FEVDs by selecting **Statistics/Multivariate time series/IRF and FEVD analysis/Graphs by**

impulse or response. A specification window as shown in Figure 78, upper panel, appears. We only need to specify the **Statistics to graph** which is **Cholesky forecast-error variance decomposition (fevd)** and click **OK**. We can now compare the FEVDs of the reverse order with those of the previous ordering (Figure 78, lower panel). Note that the FEVDs for the original order are denoted by 'varbasic' and the ones for the reverse ordering by 'order2'.

17 Testing for Unit Roots

Reading: Brooks (2019, Section 8.1)

In this section, we focus on how we can test whether a data series is stationary or not using Stata. This example uses the same data on UK house prices as employed previously (**'ukhp.dta'**). Assuming that the data have been loaded, and the variables are defined as before, we want to conduct a unit root test on the **HP** series. We click on **Statistics/Time series/Tests** and can select from a number of different unit root tests. To start with, we choose the first option **Augmented Dickey-Fuller unit-root test**. In the test specification window, we select **Variable: hp** and select **10 Lagged differences** to be included in the test (Figure 79).

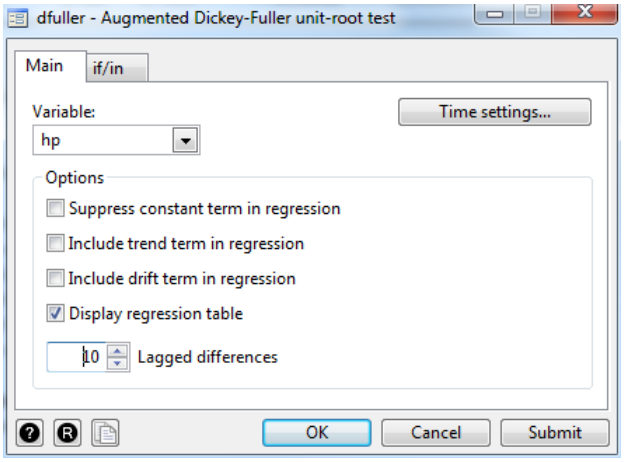


Figure 79: Specifying an Augmented Dickey–Fuller Test for Unit Roots

We can also select whether we would like to show the regression results from the auxiliary regression by checking the box **Display regression table**. We press **OK** and the following test statistics are reported in the *Output* window on the next page.

In the upper part of the output we find the actual test statistics for the null hypothesis that the series 'hp' has a unit root. Clearly, the test statistic (-0.402) is not more negative than the critical value, so the null hypothesis of a unit root in the house price series cannot be rejected. The remainder of the output presents the estimation results. Since one of the independent variables in this regression is non-stationary, it is not appropriate to examine the coefficient standard errors or their t-ratios in the test regression.

```
. dfuller hp, regress lags(10)

Augmented Dickey-Fuller test for unit root         Number of obs   =       316

                                 ------------ Interpolated Dickey-Fuller ------------
                 Test         1% Critical       5% Critical      10% Critical
               Statistic           Value             Value            Value
------------------------------------------------------------------------------
 Z(t)            -0.402            -3.455            -2.877           -2.570
------------------------------------------------------------------------------
MacKinnon approximate p-value for Z(t) = 0.9098

------------------------------------------------------------------------------
        D.hp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          hp |
         L1. |  -.0004905   .0012214    -0.40   0.688    -.0028939    .0019129
          LD.|   .3177639   .0573463     5.54   0.000     .2049181    .4306098
         L2D.|   .2950393   .0601096     4.91   0.000     .1767558    .4133229
         L3D.|   .0371026   .0624682     0.59   0.553    -.0858223    .1600275
         L4D.|  -.0215644   .0622258    -0.35   0.729    -.1440123    .1008835
         L5D.|  -.0289773    .062346    -0.46   0.642    -.1516615     .093707
         L6D.|   .0685895    .062339     1.10   0.272    -.0540811    .1912601
         L7D.|  -.1199061   .0624339    -1.92   0.056    -.2427634    .0029512
         L8D.|  -.0272475      .0631    -0.43   0.666    -.1514155    .0969206
         L9D.|    .124201   .0605096     2.05   0.041     .0051303    .2432717
        L10D.|   .0273032   .0582814     0.47   0.640    -.0873829    .1419893

       _cons |   228.4731   167.5132     1.36   0.174     -101.159    558.1051
------------------------------------------------------------------------------

.
```

Now we repeat all of the above steps for the **first difference of the house price series**. To do so we open the **dfuller** specification window again but instead of typing in HP in the **Variable** box we type **d.hp**. The **d.** is a time-series operator that tells Stata to use first differences of the respective series instead of levels. The output would appear as in the table below.

```
. dfuller d.hp, regress lags(10)

Augmented Dickey-Fuller test for unit root         Number of obs   =       315

                                 ---------- Interpolated Dickey-Fuller ----------
                 Test         1% Critical       5% Critical      10% Critical
               Statistic           Value             Value            Value
------------------------------------------------------------------------------
 Z(t)            -3.272            -3.455            -2.877           -2.570
------------------------------------------------------------------------------
MacKinnon approximate p-value for Z(t) = 0.0162

------------------------------------------------------------------------------
       D2.hp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          hp |
         LD. |  -.2744116   .0838714    -3.27   0.001    -.4394558   -.1093673
         LD2.|  -.4121368   .0900273    -4.58   0.000    -.5892947   -.2349788
       L2D2. |  -.1386534    .090044    -1.54   0.125    -.3158443    .0385374
       L3D2. |  -.0997717   .0860123    -1.16   0.247    -.2690289    .0694854
       L4D2. |  -.0999252   .0820118    -1.22   0.224    -.2613101    .0614596
       L5D2. |  -.1378524   .0798138    -1.73   0.085     -.294912    .0192072
       L6D2. |  -.0673726   .0768307    -0.88   0.381    -.2185619    .0838168
       L7D2. |  -.1864522   .0738854    -2.52   0.012    -.3318456   -.0410588
       L8D2. |  -.2157062   .0714922    -3.02   0.003    -.3563902   -.0750222
       L9D2. |  -.1473323   .0691703    -2.13   0.034    -.2834473   -.0112174
      L10D2. |  -.1803642   .0574064    -3.14   0.002      -.29333   -.0673985

       _cons |   140.8339   78.15428     1.80   0.073    -12.95995    294.6278
------------------------------------------------------------------------------

.
```

We find that the null hypothesis of a unit root can be rejected for the differenced house price series at the 5% level.[54]

For completeness, we run a unit root test on the **dhp** series (levels, not differenced), which are the percentage changes rather than the absolute differences in prices. We should find that these are also stationary (at the 5% level for a lag length of 10, test statistic –3.14).

As mentioned above, Stata presents a large number of options for unit root tests. We could for example include a trend term or a drift term in the ADF regression. Alternatively, we can use a completely different test setting – for example, instead of the Dickey–Fuller test, we could run the Phillips–Perron test for stationarity. Among the options available in Stata, we only focus on one further unit root test that is strongly related to the Augmented Dickey–Fuller test presented above, namely the Dickey-Fuller GLS test (**dfgls**). It can be accessed in Stata via **Statistics/Time series/DF-GLS test for a unit root**. 'dfgls' performs a modified Dickey-Fuller t-test for a unit root in which the series has been transformed by a generalised least-squares regression.

Several empirical studies have shown that this test has significantly greater power than the previous versions of the augmented Dickey-Fuller test. Another advantage of the 'dfgls' test is that it does not require knowledge of the optimal lag length before running it but it performs the test for a series of models that include 1 to k lags.

[54] If we decrease the number of added lags we find that the null hypothesis is rejected even at the 1% significance level.

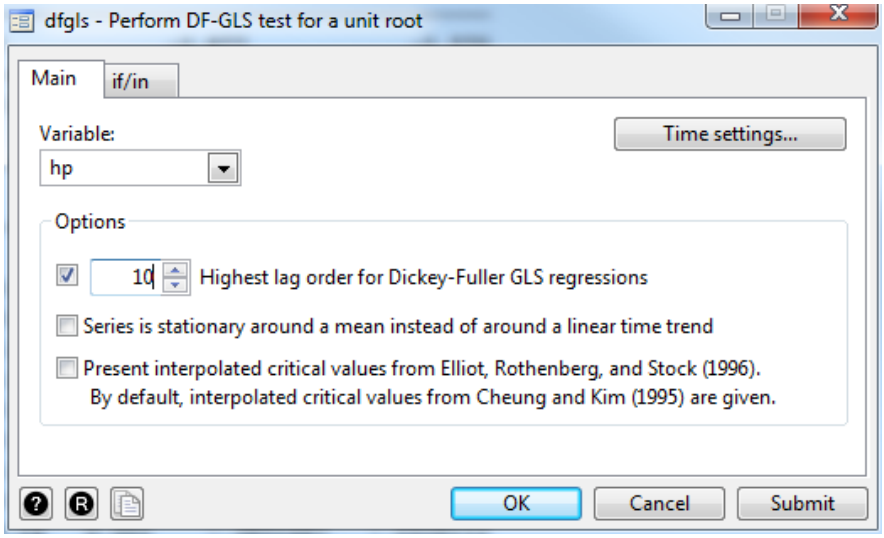


Figure 80: Specifying the Dickey–Fuller GLS Test

In the specification window, shown in Figure 80, we select **Variable: hp** and specify the **Highest lag order for Dickey-Fuller GLS regressions** to be **10**. We then press **OK** and Stata generates the following output.

```
. dfgls hp, maxlag(10)

DF-GLS for hp                                          Number of obs =   316

              DF-GLS tau      1% Critical       5% Critical      10% Critical
  [lags]     Test Statistic       Value             Value             Value
------------------------------------------------------------------------------
    10           -1.682          -3.480            -2.851            -2.567
    9            -1.635          -3.480            -2.857            -2.573
    8            -1.426          -3.480            -2.863            -2.578
    7            -1.388          -3.480            -2.868            -2.583
    6            -1.493          -3.480            -2.873            -2.587
    5            -1.413          -3.480            -2.878            -2.592
    4            -1.462          -3.480            -2.883            -2.596
    3            -1.492          -3.480            -2.888            -2.601
    2            -1.450          -3.480            -2.893            -2.605
    1            -1.070          -3.480            -2.897            -2.609

Opt Lag (Ng-Perron seq t) =  9 with RMSE  1162.444
Min SC   =  14.20163 at lag  2 with RMSE  1180.265
Min MAIC =  14.17316 at lag  2 with RMSE  1180.265

.
```

We see a series of test-statistics for models with different lag lengths, from 10 lags to 1 lag. Below the table we find information criteria in order to select the optimal lag length. The last two criteria, the modified Schwartz criterion (Min SC) and the modified Akaike information criterion (Min MAIC), both select an optimal lag length of 2.

18 Cointegration Tests and Modelling Cointegrated Systems

Reading: Brooks (2019, Sections 8.3–8.11)

In this section, we will examine the S&P500 spot and futures series contained in the **'SandPhedge.dta'** workfile (that were discussed in section 3) for cointegration using Stata. We start with a test for cointegration based on the Engle-Granger approach where the residuals of a regression of the spot price on the futures price are examined. First, we **create two new variables**, for the log of the spot series and the log of the futures series, and call them **lspot** and **lfutures**, respectively.[55] Then we run the following OLS regression:

regress lspot lfutures

Note that is is not valid to examine anything other than the coefficient values in this regression as the two series are non-stationary. Let us have a look at both the fitted and the residual series over time. As explained in previous sections, we can use the **predict** command to generate series of the fitted values and the residuals. For brevity, only the commands to create these two series are presented here:[56]

predict lspot_fit, xb
predict resid, residuals

Next we generate a graph of the actual, fitted and residual series by clicking on **Graphics/Time-series graphs/Line plots** or simply typing in the command: **twoway (tsline lspot, lcolor(blue)) (tsline lspot_fit, lcolor(green)) (tsline resid, yaxis(2) lcolor(red))**

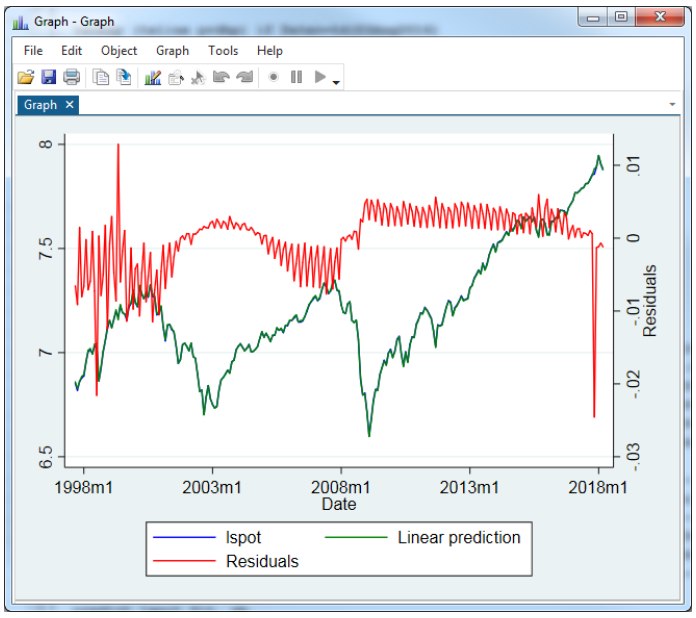


Figure 81: Actual, Fitted and Residual Plot

[55] We use the commands **gen lspot=ln(Spot)** and **gen lfutures=ln(Futures)** to generate the two series. Note that it is common to run a regression of the log of the spot price on the log of the futures rather than a regression in levels; the main reason for using logarithms is that the differences of the logs are returns, whereas this is not true for the levels.

[56] If you prefer to generate these series using the Stata menu you can select **Statistics/Postestimation/Predictions/Predictions and their SEs, leverage statistics, distance statistics, etc.** and specify each series using the specification window.

Note that we have created a second y-axis for their values as the residuals are very small and we would not be able to observe their variation if they were plotted in the same scale as the actual and fitted values.[57] The plot should appear as in Figure 81.

You will see a plot of the levels of the residuals (red line), which looks much more like a stationary series than the original spot series (the blue line corresponding to the actual values of y). Note how close together the actual and fitted lines are – the two are virtually indistinguishable and hence the very small right-hand scale for the residuals.

Let us now perform an ADF Test on the residual series '**resid**'. As we do not know the optimal lag length for the test we use the **DF-GLS** test by clicking on **Statistics/Time series/Tests/DF-GLS test for a unit root** and specifying a **12** lags as the **Highest lag order for Dickey-Fuller GLS regressions**. The output should appear as below.

```
. dfgls resid, maxlag(12)

DF-GLS for resid                                        Number of obs =   234

                DF-GLS tau      1% Critical       5% Critical      10% Critical
  [lags]     Test Statistic        Value             Value             Value
------------------------------------------------------------------------------
    12           -1.458           -3.480            -2.833            -2.552
    11           -1.366           -3.480            -2.842            -2.561
    10           -1.614           -3.480            -2.851            -2.569
     9           -1.634           -3.480            -2.860            -2.576
     8           -1.149           -3.480            -2.868            -2.584
     7           -1.775           -3.480            -2.876            -2.591
     6           -1.580           -3.480            -2.884            -2.598
     5           -1.056           -3.480            -2.891            -2.605
     4           -2.481           -3.480            -2.898            -2.611
     3           -2.616           -3.480            -2.905            -2.617
     2           -2.647           -3.480            -2.911            -2.623
     1           -5.902           -3.480            -2.917            -2.628

Opt Lag (Ng-Perron seq t) =  9 with RMSE  .0023463
Min SC   =  -11.8767 at lag  9 with RMSE  .0023463
Min MAIC = -11.98865 at lag 11 with RMSE  .0023328

.
```

The three information criteria at the bottom of the test output all suggest at least 9 lags. Let us focus on an optimal lag length of 9. For nine lags we have a test statistic of (-1.634) which is not more negative than the critical values, even at the 10% level. Thus, the null hypothesis of a unit root in the test regression residuals cannot be rejected and we would conclude that the two series are not cointegrated. This means that the most appropriate form of the model to estimate would be one containing only first differences of the variables as they have no long-run relationship.

If instead we had found the two series to be cointegrated, an error correction model (ECM) could have been estimated, as there would be a linear combination of the spot and futures prices that would be stationary. The ECM would be the appropriate model in that case rather than a model in pure first difference form because it would enable us to capture the long-run relationship between the series as well as their short-run association. We could estimate an error correction model by running the following regression

regress rspot rfutures L.resid

The corresponding estimation output is presented below.

[57] When using the menu to create the graph you can add the second axis by ticking the box **Add a second y axis on the right** next to the **Y variable** box when defining the Plot for the residuals.

```
. regress rspot rfutures L.resid

      Source |       SS           df       MS      Number of obs   =       246
-------------+----------------------------------   F(2, 243)       =  14725.21
       Model |  4562.88525         2  2281.44262   Prob > F        =    0.0000
    Residual |  37.6490867       243  .154934513   R-squared       =    0.9918
-------------+----------------------------------   Adj R-squared   =    0.9917
       Total |  4600.53433       245  18.7776912   Root MSE        =    .39362

------------------------------------------------------------------------------
       rspot |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    rfutures |   .9847711   .0057811   170.34   0.000     .9733837    .9961585
             |
       resid |
         L1. |  -55.06016   5.784954    -9.52   0.000    -66.45521    -43.6651
             |
       _cons |   .0093306   .0252095     0.37   0.712    -.0403266    .0589877
------------------------------------------------------------------------------

.
```

While the coefficient on the error correction term shows the expected negative sign, indicating that if the difference between the logs of the spot and futures prices is positive in one period, the spot price will fall during the next period to restore equilibrium, and vice versa, the size of the coefficient is not really plausible as it would imply a large adjustment. Given that the two series are not cointegrated, the results of the ECM need to be interpreted with caution and a model of the form

regress rspot rfutures L.rspot L.rfutures

would be more appropriate. Note that we can either include or exclude the lagged terms and either form would be valid from the perspective that all of the elements in the equation are stationary.

Before moving on, we should note that this result is not an entirely stable one – for instance, if we run the regression containing no lags (i.e., the pure Dickey–Fuller test) or on a subsample of the data, we should find that the unit root null hypothesis should be rejected, indicating that the series are cointegrated. We thus need to be careful about drawing a firm conclusion in this case.

18.1 The Johansen Test for Cointegration

Although the Engle–Granger approach is evidently very easy to use, as outlined above, one of its major drawbacks is that it can estimate only up to one cointegrating relationship between the variables. In the spot-futures example, there can be at most one cointegrating relationship since there are only two variables in the system. But in other situations, if there are more variables, there can potentially be more than one linearly independent cointegrating relationship. Thus, it is appropriate instead to examine the issue of cointegration within the Johansen VAR framework.

The application we will now examine centres on whether the yields on Treasury bills of different maturities are cointegrated. For this example we will use the **'fred.dta'** workfile which we created in section 9.[58] It contains six interest rate series corresponding to 3 and 6 months, and 1, 3, 5, and 10 years.[59] Each series has a name in the file starting with the letters 'GS'. The first step in any cointegration analysis is to ensure that the variables are all non-stationary in their levels form, so **confirm that this**

[58] We use the provided variable **daten** to set the time-series by typing **gen month=mofd(daten)** and **tsset month, monthly** into the Command window.

[59] The **vec intro** entry in the Stata Manual provides a good overview of estimating VECM models in Stata. It illustrates the process of testing for cointegration and estimating a VECM based on an example.

is the case for each of the six series, by running a unit root test on each one using the **dfgls** command with a maximum lag length of 12.[60]

Before specifying the VECM using the Johansen method, it is often very useful to graph the variables to see how they are behaving over time and with respect to each other. This will also help us to select the correct option for the VECM specification, e.g., if the series appear to follow a linear trend. To generate a graph of all variables we use the well-known time-series line plot:

twoway (tsline GS3M) (tsline GS6M) (tsline GS1) (tsline GS3) (tsline GS5) (tsline GS10)

and Figure 82 should appear. Note that we dropped the labels to make the legend easier readable. We can do this by typing **label variable GS3M ""** for the GS3M series and others respectively.

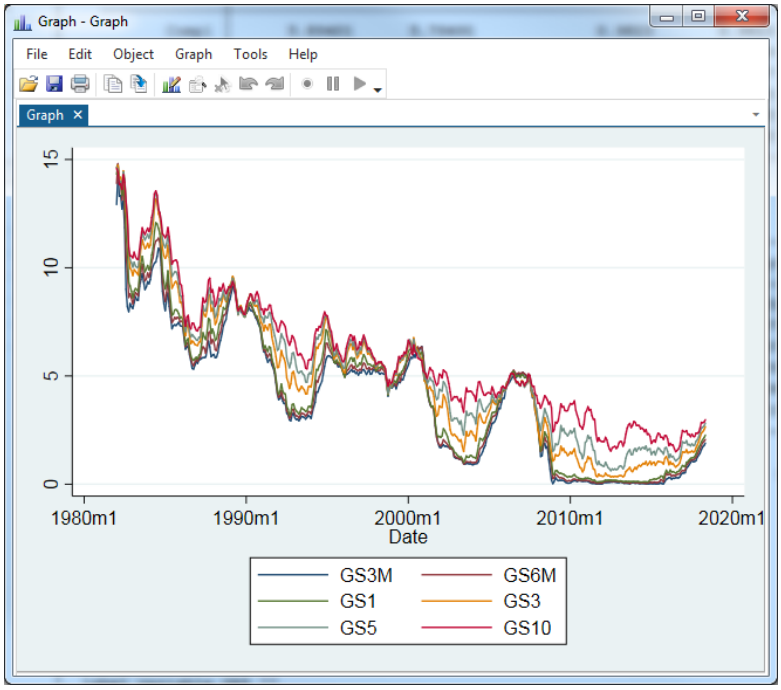


Figure 82: Graph of the Six U.S. Treasury Interest Rates

We see that the series generally follow a linear downward trend, though some series show stronger inter-temporal variation with large drops than other series. Additionally, while all series seem to be related in some way, we find that the plots of some rates resemble each other more strictly than others, e.g., the GS3M, GS6M and GS1 rates.

To test for cointegration or fit cointegrating VECMs, we must also specify how many lags to include in the model. To select the optimal number of lags we can use the methods implemented in Stata's **varsoc** command. To access this test, we click on **Statistics/Multivariate time series/VEC diagnostics and tests/Lag-order selection statistics (preestimation)**. However, as the models we are planning to estimate are very large, we need to increase the maximum number of variables that Stata allows in a model, the so-called matsize. The default matsize is 400, while the maximum matsize in Stata/MP and Stata/SE is 11,000. We simply set matsize to its maximum number using the command: **set matsize 11000**

[60] Note that for the 6-month, 5-year, and 10-year rates, the unit root test is rejected for the optimal lag length based on the Schwarz criterion. However, for the sake of this example we will continue using all six rates.

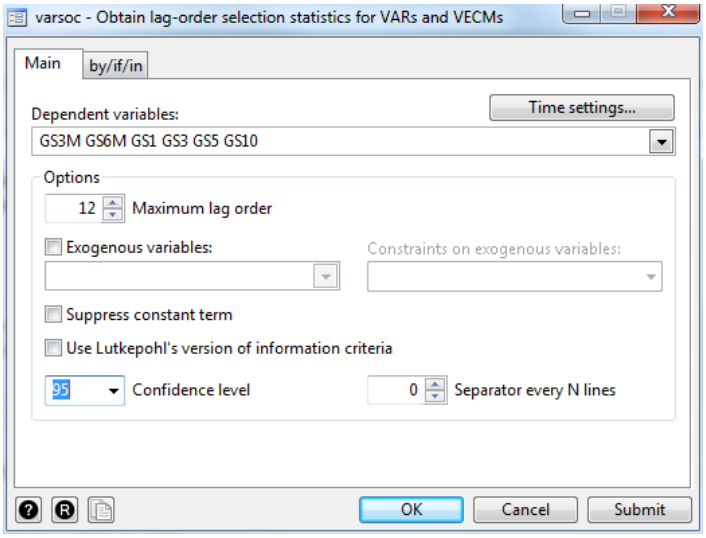


Figure 83: Specifying the Lag-Order Selection Test

Once this is done, we can run the lag-order selection test. In the specification window for the test, we first define all **six interest rates as Dependent variables** and then select a **Maximum lag order of 12** (Figure 83). We then click **OK** and the test output should appear as below.

```
. varsoc  GS3M GS6M GS1 GS3 GS5 GS10, maxlag(12)

   Selection-order criteria
   Sample:  1983m1 - 2018m5                  Number of obs      =       425
```

lag	LL	LR	df	p	FPE	AIC	HQIC	SBIC
0	-30.3943				4.8e-08	.171267	.193867	.228473
1	2901.96	5864.7	36	0.000	5.8e-14	-13.4586	-13.3004	-13.0582
2	3022.52	241.12	36	0.000	3.9e-14*	-13.8566*	-13.5628*	-13.1129*
3	3055.77	66.515	36	0.001	3.9e-14	-13.8436	-13.4143	-12.7567
4	3092.04	72.534	36	0.000	3.9e-14	-13.8449	-13.2799	-12.4148
5	3117.16	50.246	36	0.058	4.1e-14	-13.7937	-13.0931	-12.0203
6	3156.83	79.333	36	0.000	4.1e-14	-13.811	-12.9748	-11.6943
7	3187.06	60.453	36	0.007	4.2e-14	-13.7838	-12.812	-11.3239
8	3224.11	74.105	36	0.000	4.2e-14	-13.7888	-12.6814	-10.9857
9	3245.33	42.443	36	0.213	4.5e-14	-13.7192	-12.4762	-10.5729
10	3273.02	55.373*	36	0.021	4.7e-14	-13.6801	-12.3015	-10.1905
11	3297.73	49.43	36	0.067	4.9e-14	-13.627	-12.1128	-9.79418
12	3320.03	44.593	36	0.154	5.3e-14	-13.5625	-11.9127	-9.38646

```
   Endogenous:  GS3M GS6M GS1 GS3 GS5 GS10
    Exogenous:  _cons

.
```

The four information criteria provide conclusive results regarding the optimal lag length. All four criteria FPE, AIC, HQIC, and SBIC suggest an optimal lag length of 2. Note, that this might not always be the case, and it is then up to the reader to decide on the appropriate lag length. In the framework of this example, we follow the information criteria and select a lag length of 2.

The next step of fitting a VECM is determining the number of cointegrating relationships using a VEC rank test. The corresponding Stata command is **vecrank**. The tests for cointegration implemented in 'vecrank' are based on Johansen's method by comparing the log likelihood functions for a model that contains the cointegrating equation(s) and a model that does not. If the log likelihood of the unconstrained model that includes the cointegrating equations is significantly different from the log

likelihood of the constrained model that does not include the cointegrating equations, we reject the null hypothesis of no cointegration.

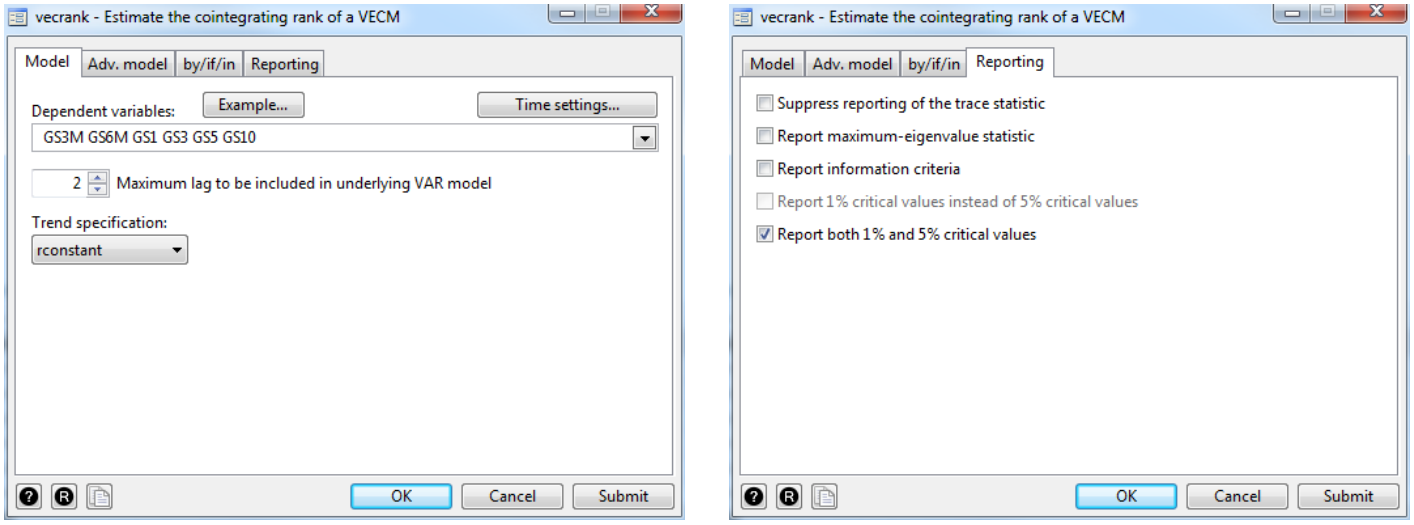


Figure 84: Testing for the Number of Cointegrating Relationships

To access the VEC rank test, we click on **Statistics/Multivariate time series** and select **Cointegrating rank of a VECM**. In the specification window (Figure 84, left panel) we define the list of **Dependent variables: GS3M GS6M GS1 GS3 GS5 GS10**. Next, we set the **Maximum lag to be included in underlying VAR model** to **2** as determined in the previous step. We set the **Trend specification** to **rconstant**, since we want to allow for the cointegration relationship to be stationary around a constant mean, but we do not want linear time trends in the levels of the data.

Switching to the reporting tab (Figure 84, right panel) , we add the 1% critical values to our output and click **OK**. The following table should appear in the Output window.

```
. vecrank GS3M GS6M GS1 GS3 GS5 GS10, trend(rconstant) levela

                    Johansen tests for cointegration
Trend: rconstant                                     Number of obs =     435
Sample:  1982m3 - 2018m5                                   Lags =       2
------------------------------------------------------------------------------
maximum                                          trace     5% critical  1% critical
  rank    parms       LL       eigenvalue  statistic      value        value
    0       36     2739.7012                 247.7512     102.14       111.01
    1       48     2786.4373     0.19336     154.2789      76.07        84.45
    2       58     2820.3046     0.14419      86.5444      53.12        60.16
    3       66     2845.9453     0.11120      35.2630*1    34.91        41.07
    4       72     2855.2577     0.04191      16.6381*5    19.96        24.60
    5       76     2859.5438     0.01951       8.0660       9.42        12.97
    6       78     2863.5768     0.01837
------------------------------------------------------------------------------
.
```

The first column in the table shows the rank of the VECM that is been tested, or in other words the number of cointegrating relationships for the set of interest rates. The second and third columns report the number of smoothing parameters and the log-likelihood values, respectively. In the fourth column we find the ordered eigenvalues, from highest to lowest. We find the λ_{trace} statistics in the fifth column, together with the corresponding critical values for the 5% level and the 1% level in the sixth and seventh column. The first row of the table tests the null hypothesis of no cointegrating vectors, against

the alternative hypothesis that the number of cointegrating equations is strictly larger than the number assumed under the null hypothesis, i.e., larger than zero. The test statistic of 247.75 considerably exceeds the critical value (111.01) and so the null of no cointegrating vector is rejected. If we then move to the next row, the test statistic (154.28) again exceeds the critical value so that the null of at most one cointegrating vector is also rejected. This continues, and we also reject the null of at most two cointegrating vectors, but we stop at the next row, where we do not reject the null hypothesis of at most three cointegrating vectors at the 1% level. If we use a significance level of 5%, we would still reject this hypothesis and would only stop in the next row.

Besides the λ_{trace} statistic, we can also employ an alternative statistic, the maximum-eigenvalue statistic (λ_{max}). In contrast to the trace statistic, the maximum-eigenvalue statistic assumes a given number of r cointegrating relations under the null hypothesis and tests this against the alternative that there are $r+1$ cointegrating equations. We can generate the results for this alternative test by going back to the 'vecrank' specification window, changing to the **Reporting** tab and checking the box **Report maximum-eigenvalue statistic**. We leave everything else unchanged and click **OK**.

The test output should now report the results for the λ_{trace} statistics in the first panel and those for the λ_{max} statistics in the panel below. We find that the results from the λ_{max} test confirm our previous results and show that we cannot reject the null hypothesis of three cointegrating relations against the alternative of four cointegrating relations between the interest rates. It is therefore reasonable to assume a cointegration rank of three.

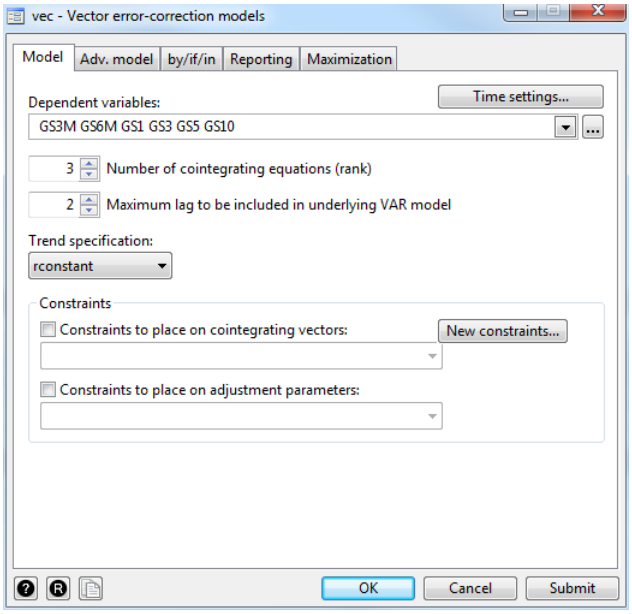


Figure 85: Specifying the VECM

Now that we have determined the lag length, trend specification and the number of cointegrating relationships, we can fit the VECM model. To do so, we click on **Statistics/Multivariate time series** and select **Vector error-correction model (VECM)**. In the VECM specification window, we first specify the six interest rates as the **Dependent variables** and then select **3** as the **Number of cointegrating equations (rank)** and **2** again as the **Maximum lag to be included in underlying VAR model** (Figure 85). As in the previous specification, we keep the default **Trend specification: rconstant** as well as all other default specifications and simply press **OK**. The output should appear as on the following pages.

```
. vec GS3M GS6M GS1 GS3 GS5 GS10, trend(rconstant) rank(3) vsquish

Vector error-correction model

Sample:  1982m3 - 2018m5                        Number of obs     =       435
                                                AIC               =  -12.78136
Log likelihood =  2845.945                      HQIC              =  -12.53731
Det(Sigma_ml)  =  8.37e-14                      SBIC              =  -12.16303

Equation           Parms      RMSE     R-sq      chi2     P>chi2
------------------------------------------------------------------
D_GS3M                 9     .235601   0.2643   152.3184   0.0000
D_GS6M                 9     .229521   0.3086   189.2563   0.0000
D_GS1                  9     .237308   0.2810   165.7268   0.0000
D_GS3                  9     .257038   0.2348   130.1366   0.0000
D_GS5                  9     .252906   0.2288   125.7885   0.0000
D_GS10                 9     .238754   0.2151   116.2124   0.0000
------------------------------------------------------------------

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
D_GS3M       |
        _ce1 |
         L1. |   .1963815    .120564     1.63   0.103    -.0399196    .4326826
        _ce2 |
         L1. |  -.5795111   .2530223    -2.29   0.022    -1.075426   -.0835966
        _ce3 |
         L1. |   .1858437   .2166587     0.86   0.391    -.2387995     .610487
       GS3M  |
         LD. |   .6082763   .1572071     3.87   0.000     .3001561    .9163966
       GS6M  |
         LD. |  -.7207148   .3165273    -2.28   0.023    -1.341097   -.1003328
        GS1  |
         LD. |   .5403262   .2967651     1.82   0.069    -.0413227    1.121975
        GS3  |
         LD. |   -.267814   .3169369    -0.85   0.398    -.8889989    .3533709
        GS5  |
         LD. |   .3052805   .3784928     0.81   0.420    -.4365517    1.047113
       GS10  |
         LD. |  -.1189049   .2055196    -0.58   0.563    -.5217158    .2839061
-------------+----------------------------------------------------------------
D_GS6M       |
        _ce1 |
         L1. |    .497106   .1174527     4.23   0.000     .2669029    .7273091
        _ce2 |
         L1. |  -1.004557   .2464927    -4.08   0.000    -1.487674   -.5214402
        _ce3 |
         L1. |   .3359907   .2110676     1.59   0.111    -.0776941    .7496756
       GS3M  |
         LD. |   .5081543   .1531502     3.32   0.001     .2079855    .8083231
       GS6M  |
         LD. |  -.7434922   .3083589    -2.41   0.016    -1.347865   -.1391198
        GS1  |
         LD. |   .7122247   .2891067     2.46   0.014     .1455859    1.278864
        GS3  |
         LD. |  -.6416528    .308758    -2.08   0.038    -1.246807   -.0364983
        GS5  |
         LD. |   .8164138   .3687254     2.21   0.027     .0937253    1.539102
       GS10  |
         LD. |  -.2669406   .2002159    -1.33   0.182    -.6593566    .1254754
------------------------------------------------------------------------------
```

D_GS1						
_ce1						
L1.	.4991636	.1214377	4.11	0.000	.26115	.7371772
_ce2						
L1.	-.8510384	.2548559	-3.34	0.001	-1.350547	-.35153
_ce3						
L1.	.1736022	.2182288	0.80	0.426	-.2541184	.6013228
GS3M						
LD.	.2946149	.1583464	1.86	0.063	-.0157383	.604968
GS6M						
LD.	-.4648259	.3188211	-1.46	0.145	-1.089704	.1600521
GS1						
LD.	.580247	.2989157	1.94	0.052	-.0056171	1.166111
GS3						
LD.	-.7109028	.3192337	-2.23	0.026	-1.336589	-.0852162
GS5						
LD.	1.032654	.3812357	2.71	0.007	.2854461	1.779863
GS10						
LD.	-.3194938	.207009	-1.54	0.123	-.7252239	.0862364
D_GS3						
_ce1						
L1.	.6350292	.1315342	4.83	0.000	.377227	.8928314
_ce2						
L1.	-1.061773	.2760448	-3.85	0.000	-1.602811	-.520735
_ce3						
L1.	.4252023	.2363725	1.80	0.072	-.0380793	.8884839
GS3M						
LD.	.0947646	.1715114	0.55	0.581	-.2413915	.4309207
GS6M						
LD.	-.4058679	.3453281	-1.18	0.240	-1.082699	.2709628
GS1						
LD.	.5358639	.3237678	1.66	0.098	-.0987093	1.170437
GS3						
LD.	-.4262092	.345775	-1.23	0.218	-1.103916	.2514974
GS5						
LD.	.7808156	.4129319	1.89	0.059	-.0285161	1.590147
GS10						
LD.	-.1952593	.2242198	-0.87	0.384	-.6347221	.2442035
D_GS5						
_ce1						
L1.	.6827193	.1294198	5.28	0.000	.4290612	.9363774
_ce2						
L1.	-1.063761	.2716074	-3.92	0.000	-1.596102	-.5314207
_ce3						
L1.	.3379851	.2325729	1.45	0.146	-.1178494	.7938195
GS3M						
LD.	.1024645	.1687544	0.61	0.544	-.228288	.433217
GS6M						
LD.	-.3878152	.3397771	-1.14	0.254	-1.053766	.2781357
GS1						
LD.	.4091105	.3185633	1.28	0.199	-.2152621	1.033483
GS3						
LD.	-.391316	.3402168	-1.15	0.250	-1.058129	.2754967
GS5						
LD.	.7164748	.4062941	1.76	0.078	-.0798471	1.512797
GS10						
LD.	-.1023226	.2206156	-0.46	0.643	-.5347211	.3300759

D_GS10						
_ce1						
L1.	.7102766	.1221778	5.81	0.000	.4708125	.9497406
_ce2						
L1.	-1.121533	.256409	-4.37	0.000	-1.624085	-.6189804
_ce3						
L1.	.3709008	.2195587	1.69	0.091	-.0594264	.8012279
GS3M						
LD.	.0193887	.1593113	0.12	0.903	-.2928557	.3316331
GS6M						
LD.	-.2003951	.320764	-0.62	0.532	-.829081	.4282908
GS1						
LD.	.297745	.3007373	0.99	0.322	-.2916893	.8871793
GS3						
LD.	-.6275827	.3211791	-1.95	0.051	-1.257082	.0019168
GS5						
LD.	.8335154	.3835589	2.17	0.030	.0817537	1.585277
GS10						
LD.	-.0428679	.2082705	-0.21	0.837	-.4510705	.3653348

Cointegrating equations

Equation	Parms	chi2	P>chi2
_ce1	3	3817.164	0.0000
_ce2	3	7927.131	0.0000
_ce3	3	23667.93	0.0000

Identification: beta is exactly identified

Johansen normalization restrictions imposed

beta	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
_ce1						
GS3M	1	.	.	.	.	.
GS6M	0	(omitted)				
GS1	-4.44e-16	.	.	.	.	.
GS3	-2.499094	.3773112	-6.62	0.000	-3.23861	-1.759578
GS5	1.573232	.6761918	2.33	0.020	.2479205	2.898544
GS10	.0989393	.3311876	0.30	0.765	-.5501763	.748055
_cons	-.5346689	.1806848	-2.96	0.003	-.8888046	-.1805333
_ce2						
GS3M	-2.78e-16	.	.	.	.	.
GS6M	1	.	.	.	.	.
GS1	4.44e-16	.	.	.	.	.
GS3	-2.786098	.2713497	-10.27	0.000	-3.317934	-2.254263
GS5	2.073284	.4862948	4.26	0.000	1.120164	3.026404
GS10	-.1585894	.2381791	-0.67	0.506	-.625412	.3082331
_cons	-.4664695	.1299425	-3.59	0.000	-.7211522	-.2117869
_ce3						
GS3M	-9.37e-17	.	.	.	.	.
GS6M	5.55e-17	.	.	.	.	.
GS1	1	.	.	.	.	.
GS3	-2.768639	.1615685	-17.14	0.000	-3.085307	-2.45197
GS5	2.242124	.2895523	7.74	0.000	1.674612	2.809636
GS10	-.3987532	.1418179	-2.81	0.005	-.6767113	-.1207952
_cons	-.2485984	.0773711	-3.21	0.001	-.400243	-.0969539

.

Stata produces a large set of tables. The header of the table contains information about the sample, the fit of each equation, and statistics regarding the overall model fit. The first table contains the estimates of the short-run parameters, along with their standard errors, z-statistics, and confidence intervals. The

two coefficients on 'L.ce1', 'L.ce2' and 'L.ce3' are the parameters in the adjustment matrix α for this model. The second table contains the estimated parameters of the cointegrating vector for this model, along with their standard errors, z-statistics, and confidence intervals.

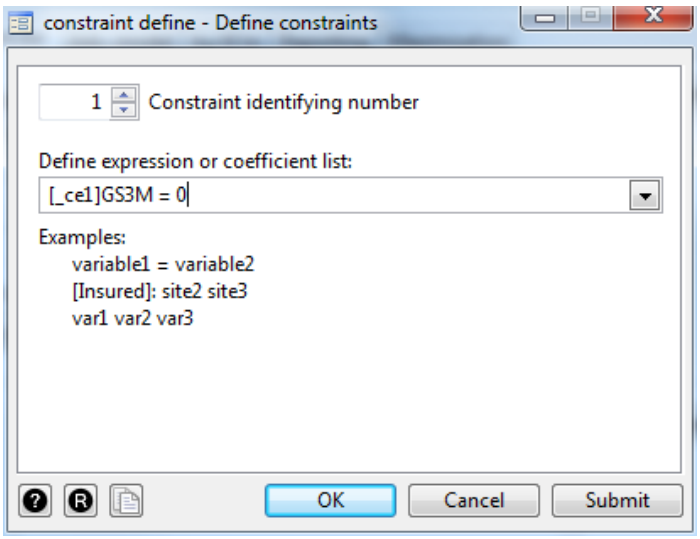


Figure 86: Defining Constraints

It is sometimes of interest to test hypotheses about either the parameters in the cointegrating vector or their loadings in the VECM. Let us assume we would like to restrict the coefficients on the rates GS3M and GS6M in the first cointegrating equation to be zero, implying that the two series do not appear in the first cointegrating equation. To do this we return to the VECM specification window, and click on **New constraints...** . A new window appears and we set the **Constraint identifying number** to **1** and **Define expression or coefficient list:** as **[_ce1]GS3M = 0** which restricts the coefficient on GS3M in the first cointegrating relationship to be zero (Figure 86). We then click **OK**.

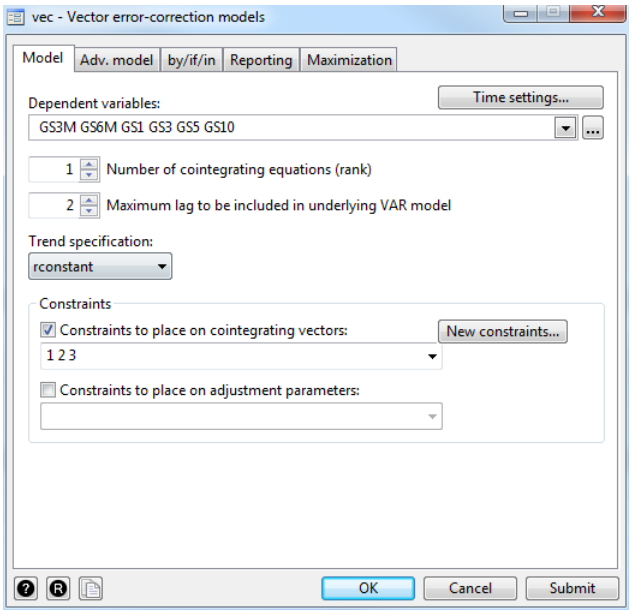


Figure 87: VECM Specification with Constraints and One Cointegrating Equation

We do the same for the GS6M series by setting the **Constraint identifying number** to **2** and **Define expression or coefficient list:** as**[_ce1]GS6M = 0** and clicking **OK**. Further, we can normalise the

effect of the 1-year rate by setting **[_ce1]GS1 = 1** as the third constraint. Once we have returned to the main VECM specification window we tick the box **Constraints to place on cointegrating vectors** and specify in the dialog box **1 2 3**, which corresponds to the three constraints we have just defined.

For this example, we are only allowing for one cointegrating relationship. Thus, we change the **Number of cointegrating equations (rank)** to **1** (Figure 87).

We are only interested in the estimates of the parameters in the cointegrating equations. We can tell Stata to suppress the estimation table for the adjustment and short-run parameters by changing to the **Reporting** tab and ticking the box **Suppress reporting of adjustment and short-run parameters**. Once all of these specifications have been executed, we press **OK** and we should find the VECM estimation output as in the table below.

```
. vec GS3M GS6M GS1 GS3 GS5 GS10, trend(rconstant) bconstraints(1 2 3) noetable nolog vsquish

Vector error-correction model

Sample:  1982m3 - 2018m5                        Number of obs     =       435
                                                AIC               =  -12.56032
Log likelihood =  2777.869                      HQIC              =  -12.39022
Det(Sigma_ml)  =  1.14e-13                      SBIC              =  -12.12936

Cointegrating equations

Equation           Parms    chi2     P>chi2
------------------------------------------
_ce1                   3   23918.86   0.0000
------------------------------------------

Identification:  beta is overidentified

 ( 1)  [_ce1]GS3M = 0
 ( 2)  [_ce1]GS6M = 0
 ( 3)  [_ce1]GS1 = 1

------------------------------------------------------------------------------
        beta |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
_ce1         |
        GS3M |          0  (omitted)
        GS6M |          0  (omitted)
         GS1 |          1          .       .       .            .           .
         GS3 |  -2.769377   .1604643  -17.26   0.000    -3.083882   -2.454873
         GS5 |   2.237347   .2875734    7.78   0.000     1.673714    2.800981
        GS10 |  -.3907743   .1408487   -2.77   0.006    -.6668326   -.1147159
       _cons |  -.2626374   .0768423   -3.42   0.001    -.4132456   -.1120293
------------------------------------------------------------------------------
LR test of identifying restrictions: chi2(2) = 17.14       Prob > chi2 = 0.000

.
```

There are two restrictions, so that the test statistic follows a χ^2 distribution with two degrees of freedom. In this case, the test statistic of 17.14 implies that we reject the restrictions at the 1% level. Thus, we would conclude that the cointegrating relationship must also include the short end of the yield curve.

19 Volatility Modelling

Reading: Brooks (2019, Chapter 9)

19.1 Testing for 'ARCH Effects' in Exchange Rate Returns

Reading: Brooks (2019, Section 9.7)

In this section we will test for 'ARCH effects' in exchange rates using the '**currrencies.dta**' dataset. First, we want to compute the Engle (1982) test for ARCH effects to make sure that this class of models is appropriate for the data. This exercise (and the remaining exercises of this section), will employ returns on daily exchange rates where there are 7,141 observations. Models of this kind are inevitably more data intensive than those based on simple linear regressions. Hence, everything else being equal, they work better when the data are sampled daily rather than at a lower frequency.

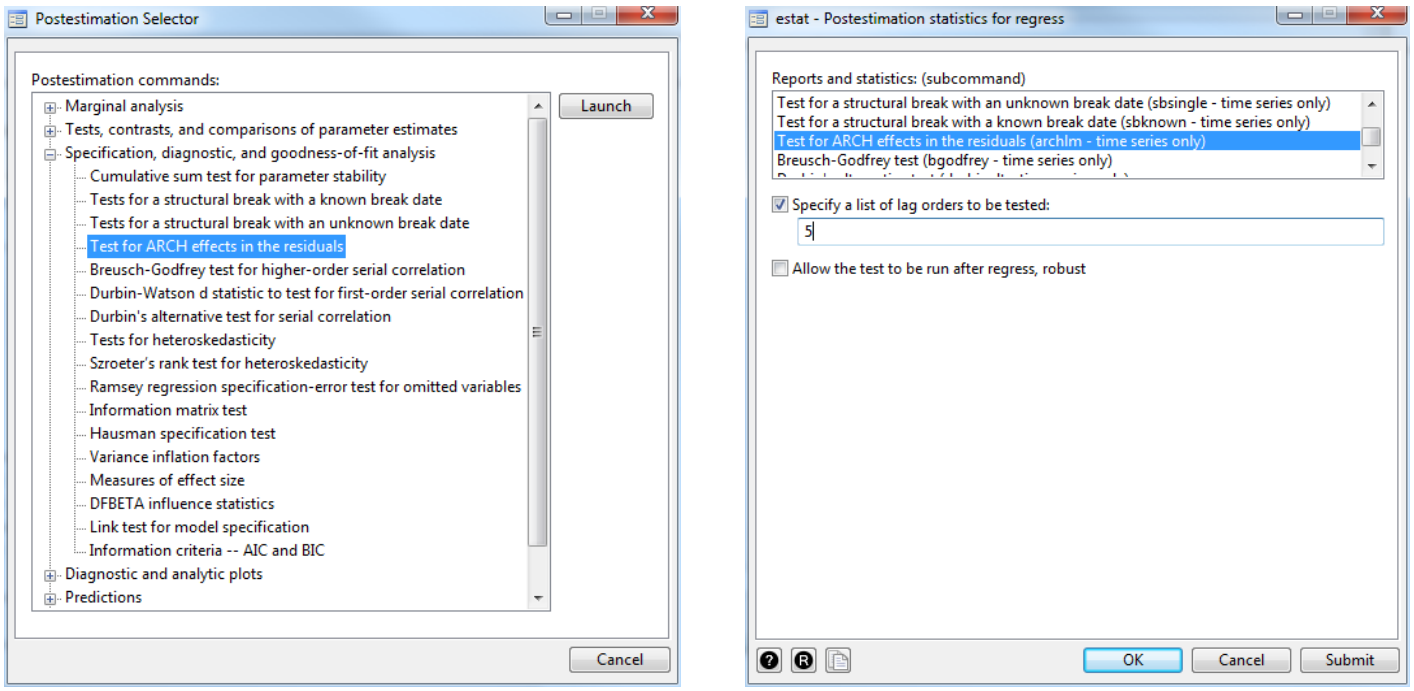


Figure 88: Testing for ARCH Effects Using Engle's Lagrange Multiplier Test

A test for the presence of ARCH in the residuals is calculated by regressing the squared residuals on a constant and p lags, where p is set by the user. As an example, assume that p is set to five. The first step is to estimate a linear model so that the residuals can be tested for ARCH. In Stata we perform these tests by fitting a constant-only model based on an OLS regression and testing for ARCH effects using Engle's Lagrange multiplier test. To do so we use the command

regress rgbp

and then click on **Statistics/Postestimation** to open the 'Postestimation Selector' and select **Specification, diagnostic, and goodness-of-fit analysis/Test for ARCH effects in the residuals** (Figure 88, left panel). In the specification window that appears we only need to **Specify a list of lag orders to be tested:** as **5** and press **OK** (Figure 88, right panel).

As can be seen from the test output, the Engle test is based on the null hypothesis that there are no ARCH effects against the alternative hypothesis that the data are characterised by (in our case) ARCH(5) disturbances.

```
. estat archlm, lags(5)
LM test for autoregressive conditional heteroskedasticity (ARCH)
```

lags(p)	chi2	df	Prob > chi2
5	252.995	5	0.0000

```
        H0: no ARCH effects      vs.  H1: ARCH(p) disturbance

.
```

The test shows a p-value of 0.0000, which is well below 0.05, suggesting the presence of ARCH effects in the pound-dollar returns.

19.2 Estimating GARCH Models

Reading: Brooks (2019, Section 9.9)

To estimate a GARCH-type model in Stata, we select **Statistics/Time series/ARCH/GARCH/ARCH and GARCH models**. In the ARCH specification window that appears we define **Dependent variable: rjpy** (Figure 89).

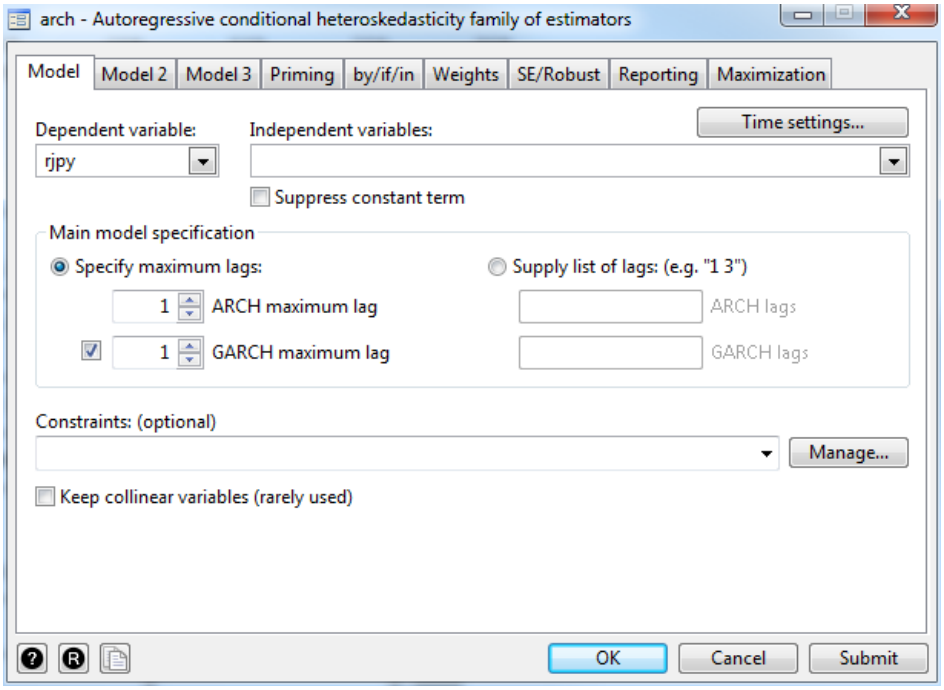


Figure 89: Specifying a GARCH(1,1) Model

We do not include any further independent variables but instead continue by specifying the **Main model specification**. Let us first **Specify maximum lags** with respect to the ARCH and GARCH terms. The default is to estimate the model with one ARCH and no GARCH. In our example we want to include one ARCH and one GARCH term (i.e., one lag of the squared errors and one lag of

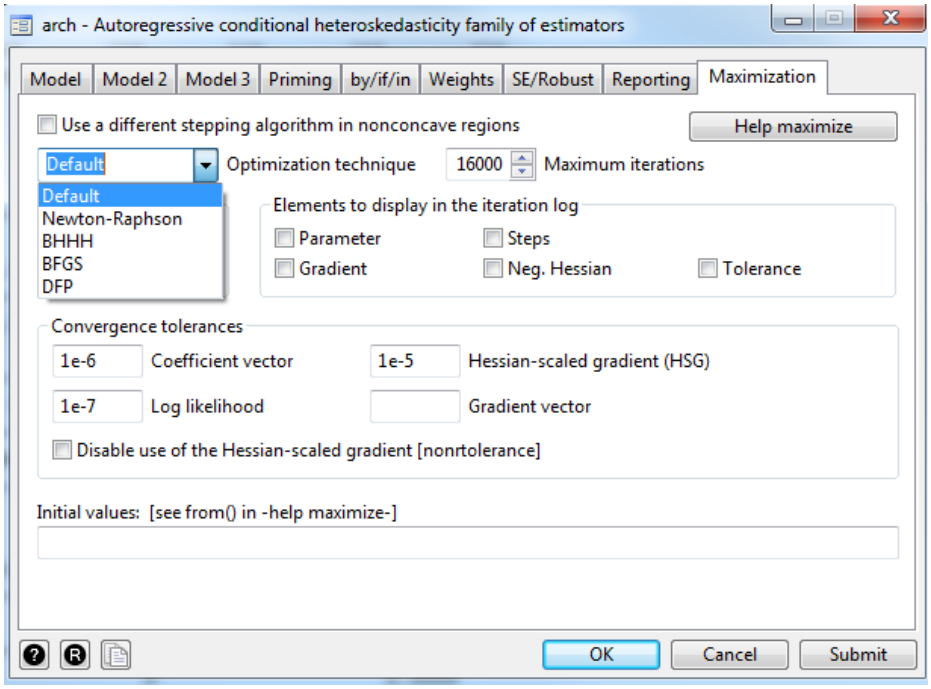


Figure 90: Optional Settings for a GARCH(1,1) Model

the conditional variance, respectively). Thus, we input **1 GARCH maximum lag**. If we wanted to include a list of non-consecutive lags, e.g., lag 1 and lag 3, we could do this by selecting **Supply list of lags** and then specifying the specific lags we want to include for the ARCH and GARCH.
The ARCH specification window provides various options of how to vary the model (Figure 90). You can have a look at the options by clicking through the various tabs. **Model 2** can be used to include ARCH-M terms (see later in this section), while **Model 3** provides different options for the assumed distribution of the errors, e.g., instead of assuming a Gaussian distribution we can specify a Student's t-distribution. In the final, tab we can specify the **Maximization** technique. Log-likelihood functions for ARCH models are often not well behaved so that convergence may not be achieved with the default estimation settings. It is possible in Stata to select the iterative algorithm (Newton–Raphson, BHHH, BFGS, DFP), to change starting values, to increase the maximum number of iterations or to adjust the convergence criteria. For example, if convergence is not achieved, or implausible parameter estimates are obtained, it is sensible to re-do the estimation using a different set of starting values and/or a different optimisation algorithm.

Estimating the GARCH(1,1) model for the yen-dollar ('rjpy') series using the instructions as listed above, and the default settings elsewhere yields the results on the following page. The coefficients on both the lagged squared residuals and lagged conditional variance terms in the conditional variance equation (i.e., the third panel in the output subtitled 'ARCH') are highly statistically significant.

Also, as is typical of GARCH model estimates for financial asset returns data, the sum of the coefficients on the lagged squared error and lagged conditional variance is very close to unity (approximately 0.99). This implies that shocks to the conditional variance will be highly persistent. This can be seen by considering the equations for forecasting future values of the conditional variance using a GARCH model. A large sum of these coefficients will imply that a large positive or a large negative return will lead future forecasts of the variance to be high for a protracted period.

The individual conditional variance coefficients are also as one would expect. The variance intercept term _cons in the 'ARCH' panel is very small, and the 'ARCH'-parameter 'L1.arch' is around 0.037 while the coefficient on the lagged conditional variance 'L1.garch' is larger at 0.956.

```
. arch rjpy, arch(1/1) garch(1/1)

(setting optimization to BHHH)
Iteration 0:   log likelihood = -4577.9248
Iteration 1:   log likelihood = -4464.8152
Iteration 2:   log likelihood =  -4367.525
Iteration 3:   log likelihood = -4351.7583
Iteration 4:   log likelihood = -4347.9926
(switching optimization to BFGS)
Iteration 5:   log likelihood = -4346.3254
Iteration 6:   log likelihood = -4345.1778
Iteration 7:   log likelihood = -4344.8833
Iteration 8:   log likelihood = -4344.8274
Iteration 9:   log likelihood = -4344.7797
Iteration 10:  log likelihood =  -4344.778
Iteration 11:  log likelihood = -4344.7779
Iteration 12:  log likelihood = -4344.7779

ARCH family regression

Sample: 12/15/1998 - 7/3/2018                   Number of obs   =      7,141
Distribution: Gaussian                          Wald chi2(.)    =          .
Log likelihood = -4344.778                      Prob > chi2     =          .

------------------------------------------------------------------------------
             |                 OPG
        rjpy |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
rjpy         |
       _cons |   .0064039   .0047412     1.35   0.177    -.0028886    .0156964
-------------+----------------------------------------------------------------
ARCH         |
        arch |
         L1. |   .0368581   .0018576    19.84   0.000     .0332174    .0404988
             |
       garch |
         L1. |   .9564137   .0023726   403.11   0.000     .9517635    .9610639
             |
       _cons |   .0016404   .0001728     9.49   0.000     .0013017     .001979
------------------------------------------------------------------------------

.
```

Stata allows for a series of postestimation commands. The following list provides a brief overview of these commands. Details can be obtained in the Stata User Manual under the entry [TS] arch postestimation:

estat	*AIC, BIC, VCE, and estimation sample summary*
estimates	*cataloging estimation results*
lincom	*point estimates, standard errors, testing, and inference for linear combinations of coefficients*
lrtest	*likelihood-ratio test*
margins	*marginal means, predictive margins, marginal effects, and average marginal effects*
marginsplot	*graph the results from margins (profile plots, interaction plots, etc.)*
nlcom	*point estimates, standard errors, testing, and inference for nonlinear combinations of coefficients*
predict	*predictions, residuals, influence statistics, and other diagnostic measures*
predictnl	*point estimates, standard errors, testing, and inference for generalized predictions*
test	*Wald tests of simple and composite linear hypotheses*
testnl	*Wald tests of nonlinear hypotheses*

19.3 GJR and EGARCH Models

Reading: Brooks (2019, Sections 9.10 – 9.13)

Since the GARCH model was developed, numerous extensions and variants have been proposed. In this section we will estimate two of them in Stata, the GJR and EGARCH models. The GJR model is a simple extension of the GARCH model with an additional term added to account for possible asymmetries. The exponential GARCH (EGARCH) model extends the classical GARCH by correcting the non-negativity constraint and by allowing for asymmetries in volatility.

We start by estimating the EGARCH model. We select **Statistics/Time series/ARCH/GARCH**. We see that there are a number of variants on the standard ARCH and GARCH model available. From the list we select **Nelson's EGARCH model**. The **arch** specification window appears and we notice that it closely resembles the arch specification window for the classical ARCH/GARCH model except that in the **Main model specification** box we are now asked to provide the maximum number of lags for the **EARCH** and **EGARCH** terms. To start with, we choose **1 EARCH** and **1 EGARCH** term to resemble the previous classic GARCH model (Figure 91).

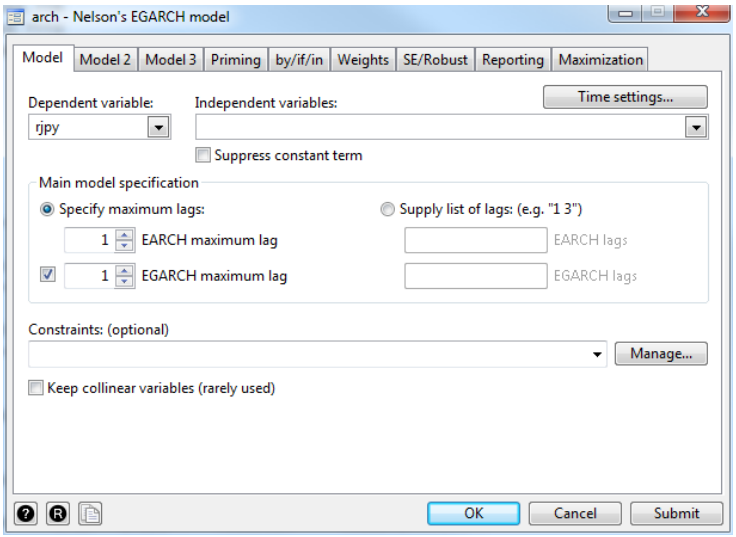


Figure 91: Estimating the EGARCH Model

After pressing **OK**, we should retrieve the output on the following page. Note that in the output we have suppressed the display of the iterations.

Looking at the results, we find that all EARCH and EGARCH terms are statistically significant. The EARCH terms represent the influence of news – lagged innovations – in the Nelson (1991) EGARCH model. The first term 'L1.earch' captures the

$$\frac{\upsilon_{t-1}}{\sqrt{\sigma^2_{t-1}}}$$

term and 'L1.earch_a' captures the

$$\frac{|\upsilon_{t-1}|}{\sqrt{\sigma^2_{t-1}}} - \sqrt{\frac{2}{\pi}}$$

term. The negative estimate on the 'L1.earch' term implies that negative shocks result in a lower next period conditional variance than positive shocks of the same sign. The result for the EGARCH asymmetry term is the opposite to what would have been expected in the case of the application of a GARCH model to a set of stock returns. But, arguably, neither the *leverage effect* or *volatility effect*

explanations for asymmetries in the context of stocks applies here. For a positive return shock, the results suggest more yen per dollar and therefore a strengthening dollar and a weakening yen. Thus, the EGARCH results suggest that a strengthening dollar (weakening yen) leads to higher next period volatility than when the yen strengthens by the same amount.

```
. arch rjpy, earch(1/1) egarch(1/1)

ARCH family regression

Sample: 12/15/1998 - 7/3/2018                     Number of obs   =      7,141
Distribution: Gaussian                            Wald chi2(.)    =          .
Log likelihood = -4322.956                        Prob > chi2     =          .

------------------------------------------------------------------------------
             |                 OPG
        rjpy |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
rjpy         |
       _cons |   .0031779   .0047107     0.67   0.500    -.0060548    .0124106
-------------+----------------------------------------------------------------
ARCH         |
       earch |
         L1. |  -.0284065   .0026653   -10.66   0.000    -.0336305   -.0231826
             |
     earch_a |
         L1. |   .1028961   .0048508    21.21   0.000     .0933887    .1124035
             |
      egarch |
         L1. |   .9864419   .0012686   777.58   0.000     .9839555    .9889283
             |
       _cons |  -.0109934    .001869    -5.88   0.000    -.0146565   -.0073302
------------------------------------------------------------------------------

.
```

Let us now test a GJR model. For this we click on **Statistics/Time series/ARCH/GARCH** and select **GJR form of threshold ARCH model**. In the GJR specification window that appears, we specify **1 ARCH maximum lag**, **1 TARCH maximum lag** and **1 GARCH maximum lag** (Figure 92), and press **OK** to fit the model.

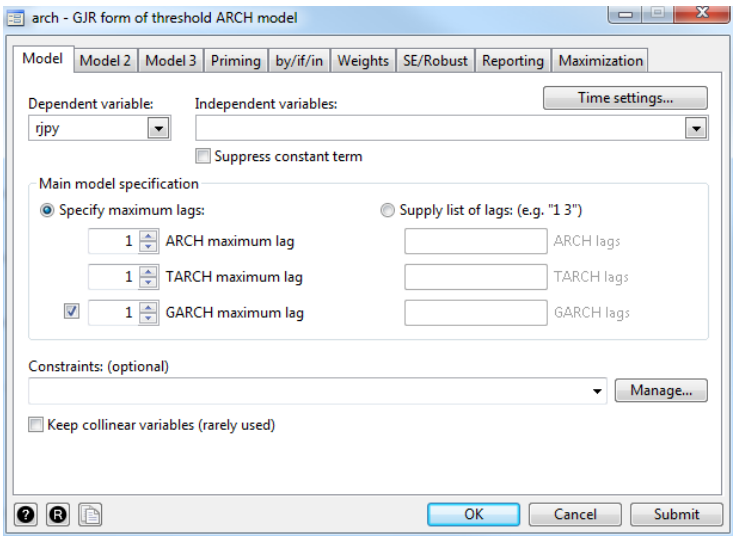


Figure 92: Estimating the GJR Model

The following GJR estimation output should appear. Note that the display of iterations is again suppressed.

```
. arch rjpy, arch(1/1) tarch(1/1) garch(1/1)
ARCH family regression

Sample: 12/15/1998 - 7/3/2018                     Number of obs   =      7,141
Distribution: Gaussian                            Wald chi2(.)    =          .
Log likelihood = -4329.717                        Prob > chi2     =          .

------------------------------------------------------------------------------
             |                 OPG
        rjpy |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
rjpy         |
       _cons |   .0035236   .0048928     0.72   0.471    -.0060661    .0131134
-------------+----------------------------------------------------------------
ARCH         |
        arch |
         L1. |   .0544101   .0030684    17.73   0.000     .0483961     .060424
             |
       tarch |
         L1. |  -.0267993   .0030914    -8.67   0.000    -.0328584   -.0207402
             |
       garch |
         L1. |   .9504746   .0027354   347.48   0.000     .9451134    .9558358
             |
       _cons |   .0020359   .0001905    10.69   0.000     .0016625    .0024094
------------------------------------------------------------------------------

.
```

Similar to the EGARCH model, we find that all ARCH, TARCH and GARCH terms are statistically significant. The 'L1.tarch' term captures the $v_{t-1}^2 I_{t-1}$ term where $I_{t-1} = 1$ if $v_{t-1}^2 < 0$ and $I_{t-1} = 0$ otherwise. We find a negative coefficient estimate on the 'L1.tarch' term, which again is not what we would expect to find according to the *leverage effect* explanation if we were modelling stock return volatilities.

19.4 GARCH-M Estimation

Reading: Brooks (2019, Section 9.15)

To estimate a GARCH-M model in Stata, we re-open the specification window for the standard GARCH model (**Statistics/Time series/ARCH/GARCH/ARCH and GARCH models**). We keep the specifications in the **Model** tab as they are, i.e., **Dependent variable: rjpy** and **1 ARCH maximum lag** and **1 GARCH maximum lag**, and change to the **Model 2** tab (Figure 93). Here we check the box **Include ARCH-in-mean term in the mean-equation specification**, which will include the contemporaneous conditional variance in the conditional mean-equation. To estimate this GARCH-M model we simply press **OK** and the following output should appear.

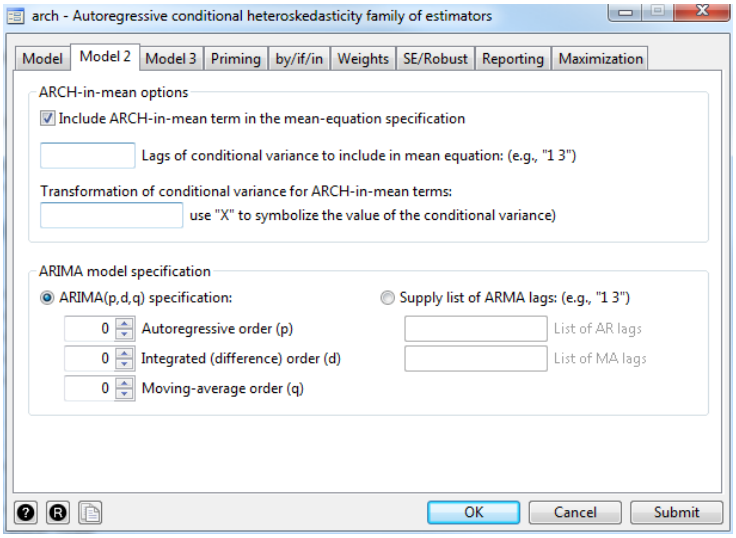


Figure 93: Specifying a GARCH-M model

```
. arch rjpy, arch(1/1) garch(1/1) archm

ARCH family regression

Sample: 12/15/1998 - 7/3/2018                     Number of obs   =      7,141
Distribution: Gaussian                            Wald chi2(1)    =       0.27
Log likelihood = -4344.629                        Prob > chi2     =     0.6057

------------------------------------------------------------------------------
             |                 OPG
        rjpy |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
rjpy         |
       _cons |   .0111954   .0102349     1.09   0.274    -.0088646    .0312555
-------------+----------------------------------------------------------------
ARCHM        |
      sigma2 |  -.0273813   .0530487    -0.52   0.606    -.1313547    .0765922
-------------+----------------------------------------------------------------
ARCH         |
        arch |
         L1. |   .036778   .0018679    19.69   0.000     .033117     .040439
             |
       garch |
         L1. |   .9565584  .0023704   403.54   0.000     .9519125    .9612043
             |
       _cons |   .0016262   .000173     9.40   0.000     .0012871    .0019653
------------------------------------------------------------------------------

.
```

In this case, the estimated volatility parameter in the mean equation (**sigma2** in the ARCHM panel) has a negative sign but is not statistically significant. We would thus conclude that, for these currency returns, there is no feedback from the conditional variance to the conditional mean.

19.5 Forecasting from GARCH Models

Reading: Brooks (2019, Section 9.18)

GARCH-type models can be used to forecast volatility. In this subsection, we will focus on generating the conditional variance forecasts using Stata. Let us assume that we want to generate forecasts based on the EGARCH model estimated earlier for the forecast period 03Aug2016 to 03Jul2018. The first step is to re-estimate the EGARCH model for the subsample running until 02Aug2016. To estimate the model we click on **Statistics/Time series/ARCH/GARCH/Nelson's EGARCH** and we input the same specifications as previously, i.e., **Dependent variable: rjpy**, **1 EARCH maximum lag**, **1 EGARCH maximum lag** (see Figure 91 above). However, now we only want to estimate the model for a subperiod of the data so we change to the **by/if/in** tab and define the following time restriction in the **If: (expression)** dialog box:

Date<=td(02Aug2016)

Then we press **OK** to fit the model. Next, we generate the conditional variance forecasts. We select **Predictions/Means from differenced or undifferenced series, conditional variances, residuals, etc.** in the 'Postestimation Selector' and are now presented with the 'predict' specification window (Figure 94).

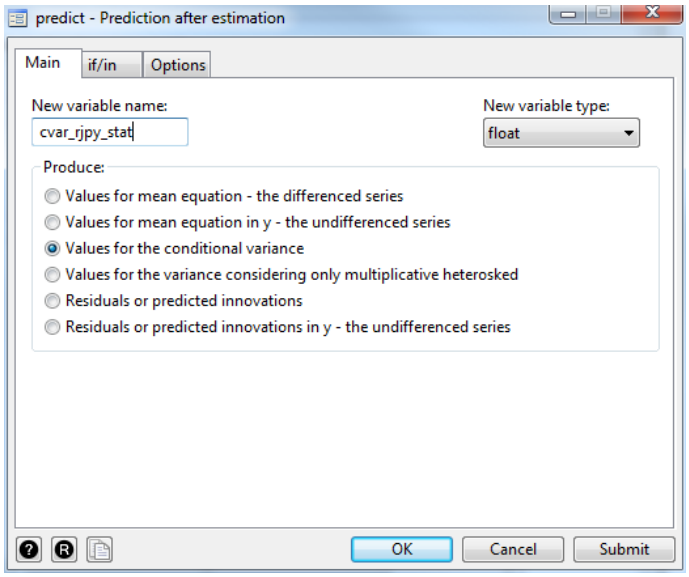


Figure 94: Generating Static Forecasts of the Conditional Variance

There is a variety of predicted values we can produce, which are listed under the **Produce:** headline. As we want to generate forecasts of the conditional variance we select the third option from the top, which is **Values for the conditional variance** (Figure 95, left panel). As was the case for the previous forecasting exercise, we can either create static (a series of rolling single-step-ahead) forecasts or dynamic (multiple-step-ahead) forecasts. Let us start with the static forecasts. We change to the **Options** tab and make sure that the option **One-step prediction** is selected. However, as this is the default specification we do not need to make any changes. Finally we have to give the new series a name. We return to the Main tab and specify **New variable name: cvar_rjpy_stat**. Then we press **OK**.

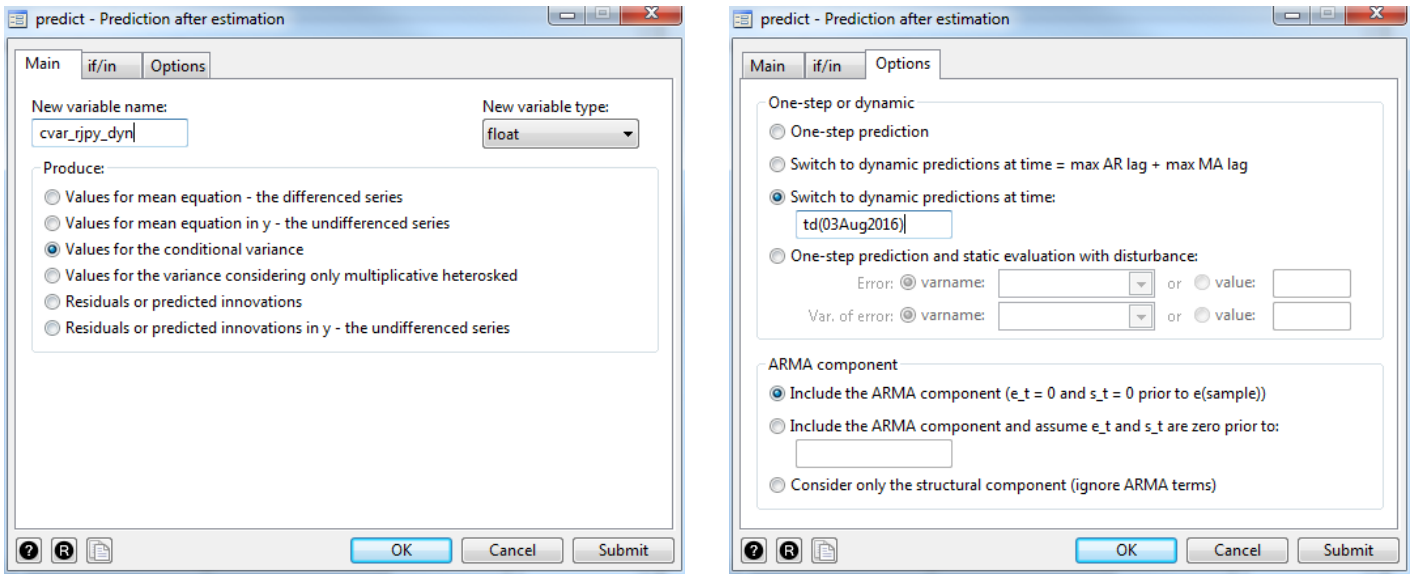


Figure 95: Generating Dynamic Forecasts of the Conditional Variances

Next we generate the dynamic conditional variance forecasts. We re-open the 'predict' specification window and change the name of the new variable to **cvar_rjpy_dyn**. We check that the **Produce** option **Values for the conditional variance** is selected and change to the **Options** tab. We now select **Switch to dynamic predictions at time** and enter the starting value **td(03Aug2016)** into the dialog box, which represents the start of the forecast period (Figure 95, right panel). Once all the changes have been made we click on **OK** and the new series should appear in the *Variables* window.

Finally we want to graphically examine the conditional variance forecasts. To generate a time-series graph of the static and dynamic forecasts, we click on **Graphics/Time series graphs/Line plots**. We create Plot 1 which contains the series **cvar_rjpy_stat** and Plot 2 comprising the series **cvar_rjpy_dyn**. As we only want to see the values for the forecast period we change to the **if/in** tab and specify **If: Date>=td(06jul2011)**. By clicking **OK**, the following graph should appear (Figure 96).

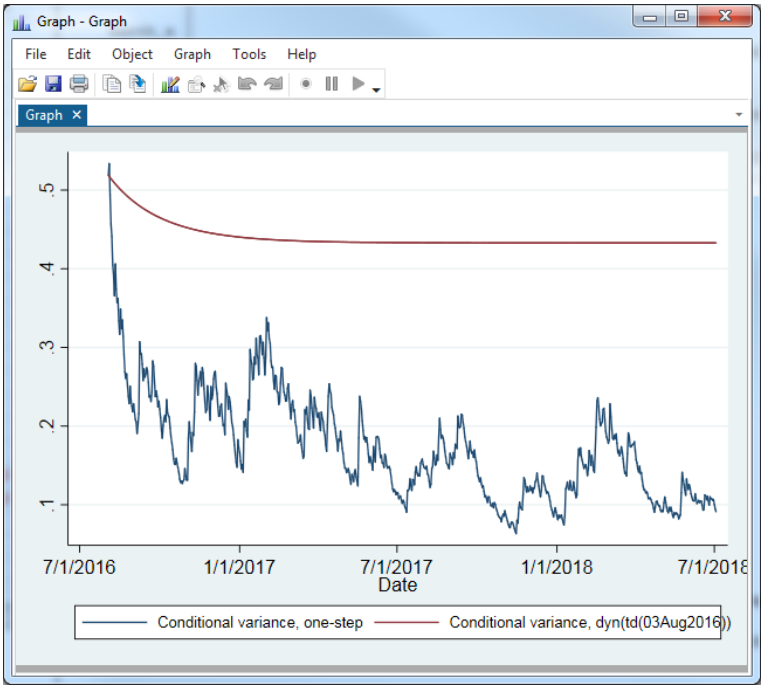


Figure 96: Graph of the Static and Dynamic Forecasts of the Conditional Variance

What do we observe? For the dynamic forecasts (red line), the value of the conditional variance starts from a historically high level at the end of the estimation period, relative to its unconditional average. Therefore, the forecasts converge upon their long-term mean value from above as the forecast horizon increases. Turning to the static forecasts (blue line), it is evident that the forecast period is characterised by a relatively low variance compared to the spikes as seen at the end of the estimation period.

Note that while the forecasts are updated daily based on new information that feeds into the forecasts, the parameter estimates themselves are not updated. Thus, towards the end of the sample, the forecasts are based on estimates almost two years old. If we wanted to update the model estimates as we rolled through the sample, we would need to write some code to do this within a loop – it would also run much more slowly as we would be estimating a lot of GARCH models rather than one.

Predictions can be similarly produced for any member of the GARCH family that is estimable with the software. For specifics of how to generate predictions after estimation of GARCH or ARCH models, refer to the corresponding postestimation commands section in the Stata Manual.

19.6 Estimation of Multivariate GARCH Models

Reading: Brooks (2019, Sections 9.20 and 9.21)

Multivariate GARCH models are in spirit very similar to their univariate counterparts, except that the former also specify equations for how the covariances move over time and are therefore by their nature inherently more complex to specify and estimate. To estimate a multivariate GARCH model in Stata, we click on **Statistics/Multivariate Time series** and we select **Multivariate GARCH**. In the 'mgarch' specification window, we are first asked to select the type of multivariate GARCH model that we would like to estimate.

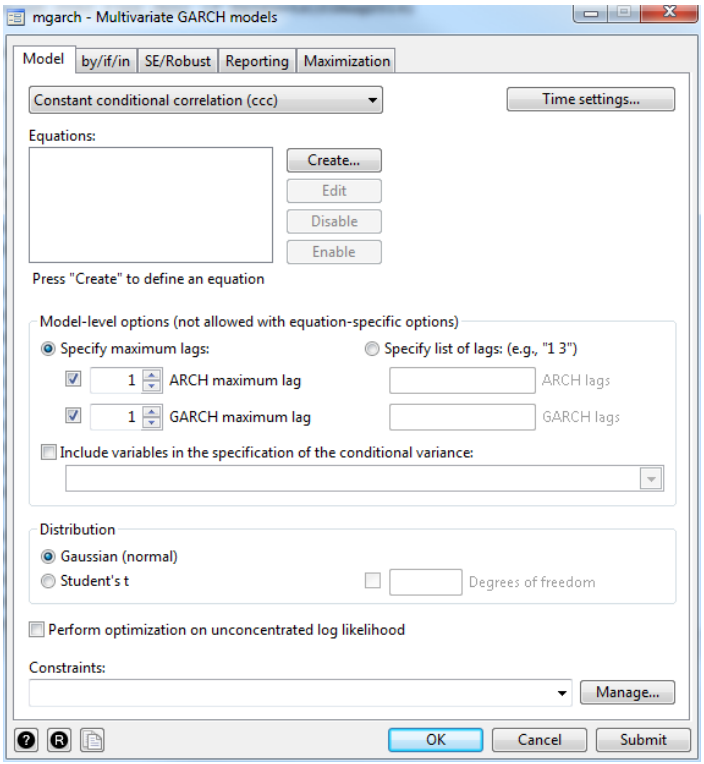


Figure 97: Specifying a Multivariate GARCH Model

Stata allows us to estimate four commonly used parametrisations: the diagonal vech model, the constant conditional correlation model, the dynamic conditional correlation model, and the time-varying

conditional correlation model. We select the **Constant conditional correlation (ccc)** model for now (Figure 97).[61]

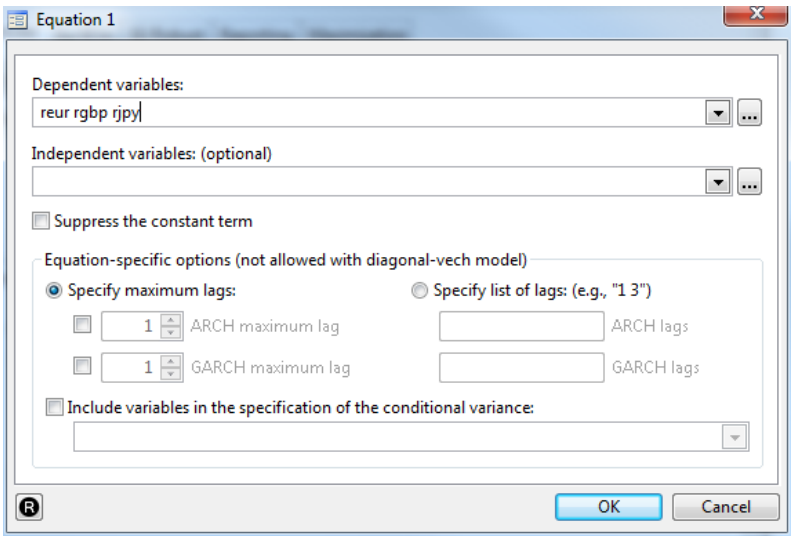


Figure 98: Specifying the Estimation Equation for a Multivariate GARCH Model

Next, we need to specify the variance equation by clicking on **Create...** next to the **Equations** dialogue box. A new window appears (Figure 98). We specify the three currency returns series in the **Dependent variables** box. Additional exogenous variables can be incorporated into the variance equation but for now we just leave the settings as they are and press **OK** to return to the main specification window. Next, we define the maximum lags of the ARCH and GARCH terms. We select **1 ARCH maximum lag** and **1 GARCH maximum lag**.

By default, Stata estimates the parameters of MGARCH models by maximum likelihood (ML) assuming that the errors come from a multivariate normal distribution. However, Stata also allows us to assume a multivariate Student's t distribution for the error terms. However, we will keep the Gaussian distribution for now. Alternatively, there are various other options to change the model specification, e.g., defining constraints on the parameters, adjusting the standard errors or the maximisation procedure. However, for now, we will keep the default settings. The complexity of this model means that it takes longer to estimate than any of the univariate GARCH or other models examined previously. Thus, it might make sense to **suppress the Iterations log** under the **Maximization** tab. In order to estimate the model, we press **OK**. The model output should resemble the table on the following page (note that the iteration log is not shown, command **nolog**).

The table is separated into different parts, organised by dependent variable. The header provides details on the estimation sample and reports a Wald test against the null hypothesis that all the coefficients on the independent variables in the mean equations are zero, which in our case is only the constant. The null hypothesis is not rejected even at the 10% level for any of the series. For each dependent variable, we first find the estimates for the conditional mean equation, followed by the conditional variance estimates in a separate panel. It is evident that the parameter estimates in the variance equations are all both plausible and statistically significant. In the final panel, Stata reports results for the conditional correlation parameters. For example, the conditional correlation between the standardized residuals for 'reur' and 'rgbp' is estimated to be 0.637.

[61] The Diagonal VECH model (DVECH) does not converge and, thus, does not produce any estimates given the data at hand and the specification that we want to estimate. Therefore, we use the Constant conditional correlation model in the following application. However, a corresponding Diagonal VECH model would theoretically be estimated in the same way and only the model type in the bottom left corner needs to be adjusted.

```
. mgarch ccc (reur rgbp rjpy =), arch(1/1) garch(1/1) nolog

Constant conditional correlation MGARCH model

Sample: 12/15/1998 - 7/3/2018                     Number of obs   =      7,141
Distribution: Gaussian                            Wald chi2(.)    =          .
Log likelihood =  -9473.28                        Prob > chi2     =          .
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
reur						
_cons	-.0069745	.0045096	-1.55	0.122	-.0158131	.0018641
ARCH_reur						
arch						
L1.	.0217505	.0016961	12.82	0.000	.0184261	.0250748
garch						
L1.	.976285	.0017656	552.96	0.000	.9728245	.9797454
_cons	.0004248	.0000911	4.66	0.000	.0002463	.0006033
rgbp						
_cons	-.004326	.0041468	-1.04	0.297	-.0124536	.0038017
ARCH_rgbp						
arch						
L1.	.0338588	.0033014	10.26	0.000	.027388	.0403295
garch						
L1.	.9610441	.0039965	240.47	0.000	.9532111	.9688772
_cons	.0009483	.000212	4.47	0.000	.0005328	.0013639
rjpy						
_cons	.0048798	.0048581	1.00	0.315	-.004642	.0144015
ARCH_rjpy						
arch						
L1.	.0420524	.0041536	10.12	0.000	.0339115	.0501932
garch						
L1.	.9504692	.004899	194.01	0.000	.9408673	.9600711
_cons	.0018945	.0003579	5.29	0.000	.0011931	.0025958
corr(reur,rgbp)	.6373394	.0070216	90.77	0.000	.6235773	.6511015
corr(reur,rjpy)	.312805	.0106788	29.29	0.000	.291875	.3337351
corr(rgbp,rjpy)	.2116868	.0113205	18.70	0.000	.1894991	.2338746

```
.
```

20 Modelling Seasonality in Financial Data

20.1 Dummy Variables for Seasonality

Reading: Brooks (2019, Section 10.3)

In this subsection, we will test for the existence of a January effect in the stock returns of Microsoft using the '**macro.dta**' workfile. In order to examine whether there is indeed a January effect in a monthly time-series regression, a dummy variable is created that takes the value one only in the months of January. To create the dummy **JANDUM**, it is easiest to first create a new variable that extracts the month from the **Date** series. To do so, we type the following command into the *Command* window and press **Enter**:

gen Month=month(dofm(Date))

where **month()** tells Stata to extract the month component from the 'Date' series and the 'dofm()' term is needed as the 'month()' command can only be performed on date series that are coded as daily data. If you inspect the new series in the Data Editor you will notice that the series Month contains a '1' if the month is January, a '2' if the month is February, a '3' if the month is March, etc. Now it is very simple to create the 'JANDUM' dummy. We type in the command window the following expression:

gen JANDUM = (Month==1)

and we press Enter. The new variable 'JANDUM' contains the binary value of the logical expression 'Month==1', which is false (0) for all months except January for which it is true and hence 1.

We can now run the APT-style regression first used in Section 7 of this guide but this time including the new 'JANDUM' dummy variable. The command for this regressions is as follows:

regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm APR00DUM DEC00DUM JANDUM

The results of this regression are presented below.

```
. regress ermsoft ersandp dprod dcredit dinflation dmoney dspread rterm APR00DUM DEC00DUM JANDUM

      Source |       SS           df       MS      Number of obs   =       383
-------------+----------------------------------   F(10, 372)      =     26.04
       Model |  14511.7654        10  1451.17654   Prob > F        =    0.0000
    Residual |  20733.0291       372  55.7339491   R-squared       =    0.4117
-------------+----------------------------------   Adj R-squared   =    0.3959
       Total |  35244.7945       382  92.26386     Root MSE        =    7.4655

------------------------------------------------------------------------------
     ermsoft |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     ersandp |   1.246083   .0900023    13.85   0.000     1.069106    1.42306
       dprod |  -.2849477   .7027587    -0.41   0.685    -1.666825    1.09693
     dcredit |  -.0103142   .0261098    -0.40   0.693    -.0616556    .0410271
  dinflation |   .2300732   1.368592     0.17   0.867    -2.461073    2.92122
      dmoney |   .0029512   .0155012     0.19   0.849    -.0275298    .0334323
     dspread |   1.188034   3.957871     0.30   0.764    -6.594571    8.970638
       rterm |   4.209344   1.635079     2.57   0.010      .994188    7.424501
   APR00DUM  |  -37.79762   7.560603    -5.00   0.000    -52.6645   -22.93074
   DEC00DUM  |  -28.86234   7.520532    -3.84   0.000    -43.65043  -14.07426
      JANDUM |   3.139099   1.654212     1.90   0.059    -.1136799    6.391877
       _cons |   1.044217   .4941695     2.11   0.035     .0725009    2.015933
------------------------------------------------------------------------------

.
```

As can be seen, the coefficient on the January dummy is statistically significant at the 10% level, and it has the expected positive sign. The coefficient value of 3.139, suggests that on average and holding everything else equal, Microsoft stock returns are around 3.1% higher in January than the average for other months of the year.

20.2 Estimating Markov Switching Models

Reading: Brooks (2019, Sections 10.5 – 10.7)

In this subsection, we will be estimating a Markov switching model in Stata. The example that we will consider in this subsection relates to the changes in house prices series used previously. So we **re-open ukhp.dta**. Stata enables us to fit two types of Markov switching models: Markov switching dynamic regression (MSDR) models, which allow a quick adjustment after the process changes state and Markov switching autoregression (MSAR) models that allow a more gradual adjustment. In this example, we will focus on the former case.

To open the specification window for Markov switching regressions, we select **Statistics/Time series/Markov-switching model**. In the specification window we first select the **Model**. As we want to test a **Dynamic regression** we just keep the default option. Next we are asked to select the **Dependent variable** and we select **dhp**. We want to estimate a simple switching model with just a varying intercept in each state. As Stata automatically includes the (state-dependent) intercept we do not need to specify any further variables in the boxes for the 'Nonswitch variables' or the variables with 'Switching coefficients'. However, if we wanted to include further variables that allow for either changing or non-changing coefficient parameters across states we could do this using the respective dialogue boxes. Let us move on to specifying the **Number of states**. The default is **2** states and, for now, we stick with the default option. Finally, we also want the variance parameters to vary across states, so we check the box **Specify state-dependent variance parameters**. Once all of these specifications are made, the window should resemble Figure 99.

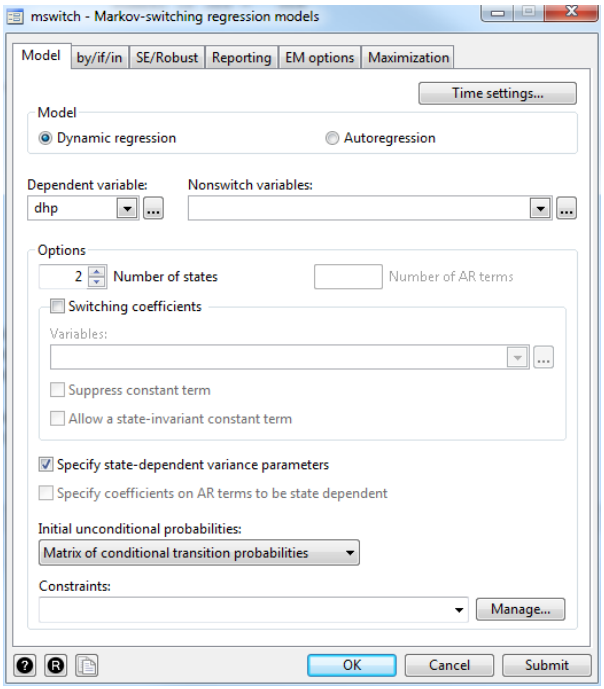


Figure 99: Specifying a Markov Switching Model

We click **OK** and the results should appear as in the following table. Examining the results, it is clear

that the model has successfully captured the features of the data. Two distinct regimes have been identified: regime 1 with a negative mean return (corresponding to a price fall of 0.24% per month) and a relatively high volatility, whereas regime 2 has a high average price increase of 0.78% per month and a much lower standard deviation.

```
. mswitch dr dhp, varswitch nolog

Performing EM optimization:

Performing gradient-based optimization:


Markov-switching dynamic regression

Sample:  1991m2 -  2018m3                       No. of obs      =        326
Number of states =     2                        AIC             =     2.9423
Unconditional probabilities: transition         HQIC            =     2.9701
                                                SBIC            =     3.0120
Log likelihood = -473.5924
```

dhp	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
State1						
_cons	-.2388203	.1373496	-1.74	0.082	-.5080205	.0303799
State2						
_cons	.7809635	.0730538	10.69	0.000	.6377807	.9241464
sigma1	1.190223	.0924647			1.022118	1.385975
sigma2	.8961665	.0445806			.8129146	.9879444
p11	.9693812	.0211635			.8867072	.992252
p21	.0168668	.0111869			.0045519	.0604745

```
.
```

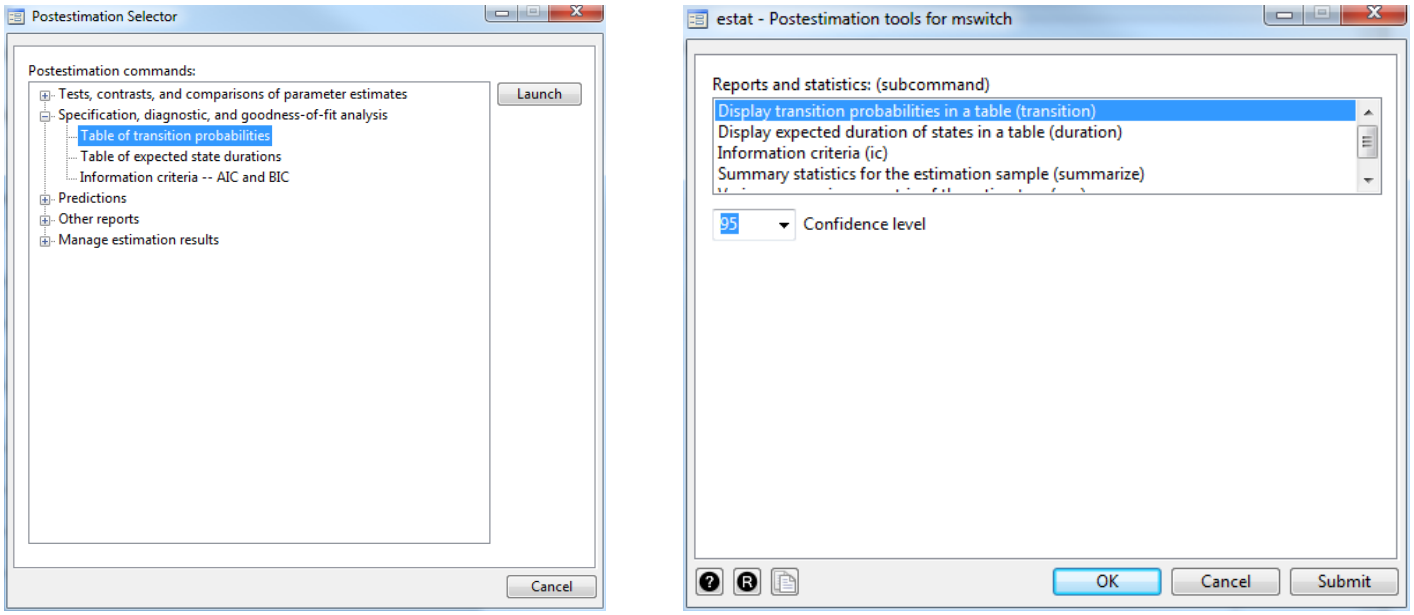


Figure 100: Generating a Table of Transition Probabilities

To see the transition probabilities matrix, we open the 'Postestimation Selector' (**Statistics/Postestimation**) and then select **Specification, diagnostic, and goodness-of-fit analysis/Table of transition probabilities** (Figure 100, left panel).

Clicking on **Launch**, the specification window as shown in Figure 100, right panel, appears and we simply click **OK** to generate the following transition matrix. Looking at the results, it appears that the regimes are fairly stable, with probabilities of around 97% of remaining in a given regime next period.

```
. estat transition

Number of obs = 326
```

Transition Probabilities	Estimate	Std. Err.	[95% Conf.	Interval]
p11	.9693812	.0211635	.8867072	.992252
p12	.0306188	.0211635	.007748	.1132928
p21	.0168668	.0111869	.0045519	.0604745
p22	.9831332	.0111869	.9395255	.9954481

```
.
```

We can also estimate the duration of staying in each regime. To do so, we simply select the second option in the 'Postestimation Selector' called **Table of expected state durations**. After launching this test and clicking **OK** in the new specification window, we should find the following output displayed in the *Output* window.

```
. estat duration

Number of obs = 326
```

Expected Duration	Estimate	Std. Err.	[95% Conf.	Interval]
State1	32.65966	22.57415	8.826685	129.0663
State2	59.28801	39.32262	16.5359	219.6866

```
.
```

We find that the average duration of staying in regime 1 is 33 months and of staying in regime 2 is 59 months. Finally, we would like to predict the probabilities of being in each one of the regimes at any given point in time. We have only two regimes, and thus the probability of being in regime 1 tells us the probability of being in regime 2 at a given point in time, since the two probabilities must sum to one. To generate the state probabilities, we select **Linear predictions, state probabilities, residuals, etc.** in the 'Postestimation Selector' (Figure 101, left panel).

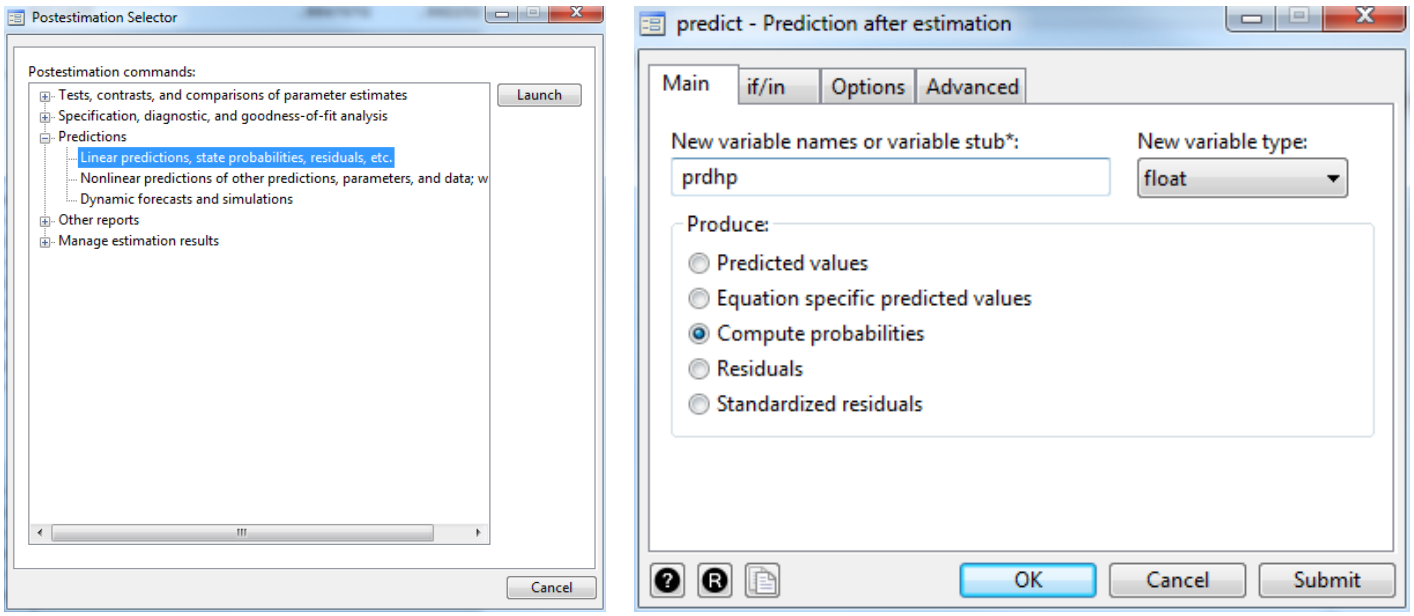


Figure 101: Generating Smoothed State Probabilities

In the 'predict' specification window we first name the new variable that should contain the probabilities of being in state 1 (Figure 101, right panel). We choose **New variable names or variable stub: prdhp**. Next we specify that Stata should **Compute probabilities**. Once all of this is specified, we click **OK** and should find the new variable in the *Variables* window. To visually inspect the probabilities, we can make a graph of them, i.e., we graph variable 'prdhp'. To do so we click **Graphics/Time-series graphs/Line plots**. We click on **Create...** and choose **Y variable: prdhp**. Then we click **Accept** and **OK** and the graph as shown in Figure 102 should appear.

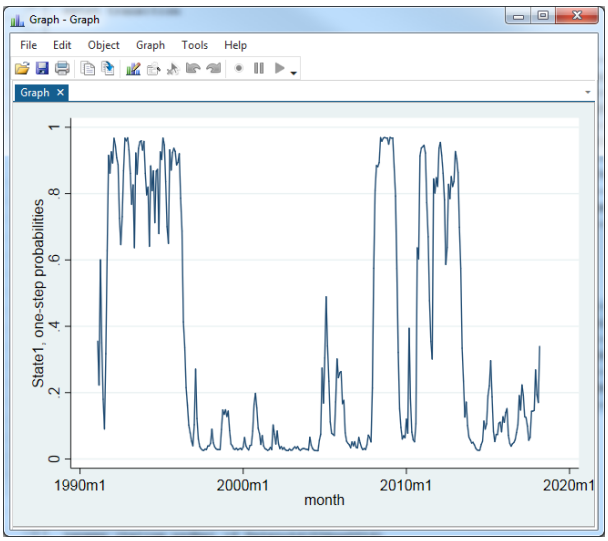


Figure 102: State Probabilities Graph

Examining how the graph moves over time, the probability of being in regime 1 was close to one until the mid-1990s, corresponding to a period of low or negative house price growth. The behaviour then changed and the probability of being in the low and negative growth state (regime 1) fell to zero and the housing market enjoyed a period of good performance until around 2005 when the regimes became less stable but tending increasingly towards regime 1. Since early 2013 it is again much more likely to be in regime 2.

21 Panel Data Models

Reading: Brooks (2019, Chapter 11)

The estimation of panel models, with either fixed and random effects, is very easy with Stata; the harder part is organising the data so that the software can recognise that you have a panel of data and can apply the techniques accordingly. While there are several ways to construct a panel workfile in Stata, the simplest way, which will be adopted in this example, is to use the following three stages:

1. Set up your data in an Excel sheet so that it fits a panel setting, i.e., construct a variable that identifies the cross-sectional component (e.g., a company's CUSIP as identifier for different companies, a country code to distinguish between different countries, etc.), and a time variable and stack the data for each company above each other. This is called the 'long' format.[62]

2. Import the data into Stata using the regular **Import** option.

3. Declare the dataset to be panel data using the **xtset** command.

The application to be considered here is that of a variant on an early test of the capital asset pricing model due to Fama and MacBeth (1973). Their test involves a 2-step estimation procedure: first, the betas are estimated in separate time-series regressions for each firm and, second, for each separate point in time, a cross-sectional regression of the excess returns on the betas is conducted

$$R_{it} - R_{ft} = \lambda_0 + \lambda_m \beta_{Pi} + u_i \tag{12}$$

where the dependent variable, $R_{it} - R_{ft}$, is the excess return of the stock i at time t and the independent variable is the estimated beta for the portfolio (P) that the stock has been allocated to. The betas of the firms themselves are not used on the RHS, but rather, the betas of portfolios formed on the basis of firm size. If the CAPM holds, then λ_0 should not be significantly different from zero and λ_m should approximate the (time average) equity market risk premium, $R_m - R_f$. Fama and MacBeth (1973) proposed estimating this second stage (cross-sectional) regression separately for each time period, and then taking the average of the parameter estimates to conduct hypothesis tests. However, one could also achieve a similar objective using a panel approach. We will use an example in the spirit of Fama–MacBeth comprising the annual returns and 'second pass betas' for 11 years on 2,500 UK firms.[63]

To test this model, we will use the '**panelx.xls**' workfile. Let us first have a look at the data in Excel. We see that missing values for the 'beta' and 'return' series are indicated by a '**NA**'. The Stata symbol for missing data is '**.**' (a dot) so that Stata will not recognise the 'NA' as indicating missing data. Thus we will need to clean the dataset first in order for Stata to correctly process the data.

We start by importing the excel file into Stata. Remember to tick the **Import first row as variable names** box. It is now helpful to use the **codebook** command to get a first idea of the data characteristics of the variables we have imported. We can either type **codebook** directly into the command window or we use the Menu by clicking **Data/Describe data/Describe data contents (codebook)**. We just click **OK** to generate the statistics for all variables in memory. The return and beta series have been imported as strings instead of numeric values due to the 'NA' terms for missing values that Stata does not recognise.

[62] You can also change your dataset into a long format using the Stata command **reshape**. Refer to the corresponding entry in the Stata manual for further details.

[63] Source: computation by Keith Anderson and the author. There would be some significant limitations of this analysis if it purported to be a piece of original research, but the range of freely available panel datasets is severely limited and so hopefully it will suffice as an example of how to estimate panel models with Stata. No doubt readers, with access to a wider range of data, will be able to think of much better applications. There are also several illustrative examples of applications in panel settings in the Stata manual.

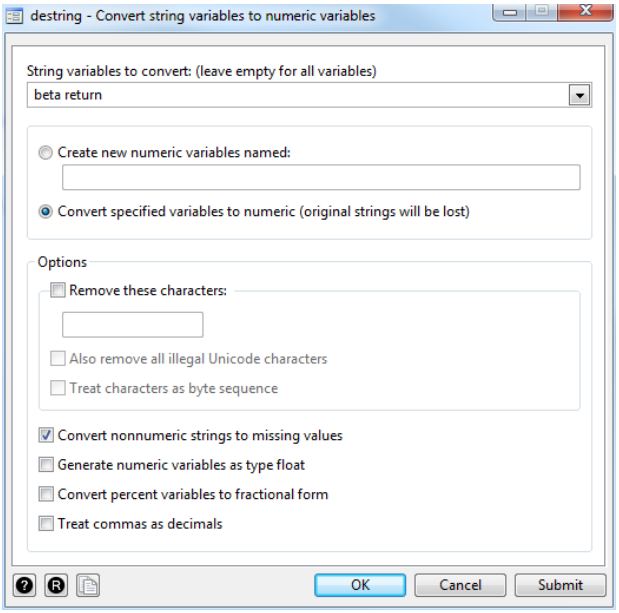


Figure 103: Transforming String Variables into Numeric Variables

So first we need to transform the string variables into numeric values. We click on **Data/Create or change data/Other variable-transformation commands** and choose the option **Convert variables from string to numeric**. In the specification window that appears we first select the two variable that we want to destring, i.e., 'beta' and 'return' (Figure 103). We are now given the option to either create a new variable which contains the destringed variables by selecting the first option and specifying a new variable name or we can replace the string variables with the newly created numeric series by clicking on **Convert specified variables to numeric (original strings will be lost)**. We choose the latter. Finally, we want Stata to replace all 'NA' values with the Stata symbol for missing values. This can be achieved by checking the box **Convert nonnumeric strings to missing values**. We click **OK**. When re-running the **codebook** command we should find that the series 'beta' and 'return' are now numeric values and that all missing values are indicated by a dot.

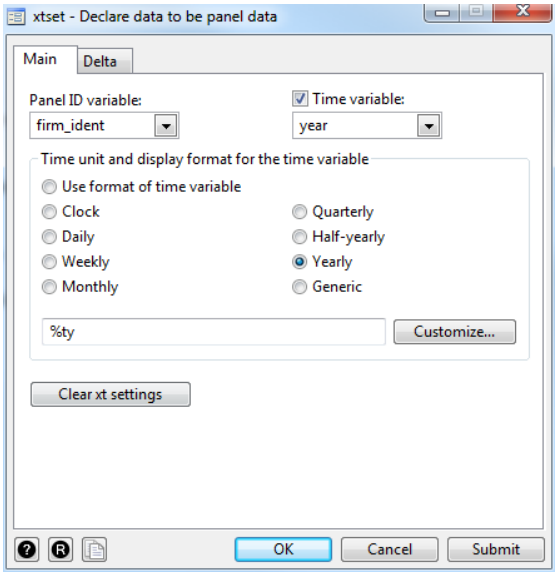


Figure 104: Declaring a Dataset to be Panel Data

The next step is to declare the dataset to be panel data. This includes defining the time component and

the cross-sectional component of our data. This step is important for commands that we will be using later in the analysis. We click on **Statistics/Longitudinal/panel data/Setup and utilities** and select **Declare dataset to be panel data**. In the specification window we define **Panel ID variable: firm_ident** and check the box for **Time variable** which we define to be **year** (Figure 104). We can now provide Stata with further information regarding the time unit of the time variable. We select **Yearly** as our dataset comprises yearly data. Once this has been specified we click **OK**. We should then find the following output in the output window.

```
. xtset firm_ident year, yearly
       panel variable:  firm_ident (strongly balanced)
        time variable:  year, 1996 to 2006
                delta:  1 year

.
```

Now our dataset is ready to be used for panel data analysis. You will find that Stata has many tools specific for panel data if you click on **Statistics/Longitudinal/panel data**. For example, if you select **Setup and utilities** Stata provides you with information about the structure of your panel, e.g., the number of time periods and the number of panel entities. Additionally, selecting **Summarize xt data** is the panel version of the regular command to create summary statistics of the data. If we select this option and choose the variables 'beta' and 'return' for which the summary statistics should be generated the following output should appear.

```
. xtsum return beta

Variable              |      Mean   Std. Dev.       Min        Max    Observations
----------------------+---------------------------------------------------------
return    overall     | -.0015455   .0383278  -1.005126   .7063541     N =   24091
          between     |             .0370384  -1.004813   .1573664     n =    2257
          within      |             .0339615   -.891553   .6615286  T-bar = 10.6739
                      |
beta      overall     |  1.104948   .2035695   .6608706   1.611615     N =    9073
          between     |             .1742001   .6608706   1.611615     n =    1851
          within      |             .1302356   .4626548   1.677356  T-bar = 4.90167

.
```

We find that besides the 'overall' versions of the test statistics (which are the ones that are reported when using the standard 'summarize' command) two additional versions are reported, i.e., 'between' and 'within', which capture the cross-sectional and the time-series dimensions of the data, respectively. For example, if we look at the 'Std. Dev.' column we see how much the series vary 'overall', how much variation there is 'between' companies and how much variation there is for one company over time, i.e., 'within' one company. This command is very useful to get a better understanding of the data structure and the source of variation in the data.

However, our primary aim is to estimate the CAPM-style model in a panel setting. Let us first estimate a simple pooled regression with neither fixed nor random effects. Note that in this specification we are basically ignoring the panel structure of our data and assuming that there is no dependence across observations (which is very unlikely for a panel dataset). We can use the standard 'regress' command for simple OLS models. In particular we type the following regression command into the *Command* window: **regress return beta** and press **Enter** to generate the estimates presented below.

```
. regress return beta
```

Source	SS	df	MS	Number of obs	=	8,856
				F(1, 8854)	=	0.03
Model	.000075472	1	.000075472	Prob > F	=	0.8680
Residual	24.2044271	8,854	.002733728	R-squared	=	0.0000
				Adj R-squared	=	-0.0001
Total	24.2045026	8,855	.002733428	Root MSE	=	.05229

return	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
beta	.0004544	.0027347	0.17	0.868	-.0049063	.0058151
_cons	.0018425	.0030746	0.60	0.549	-.0041844	.0078695

.

We can see that neither the intercept nor the slope is statistically significant. The returns in this regression are in proportion terms rather than percentages, so the slope estimate of 0.000454 corresponds to a risk premium of 0.0454% per month, or around 0.5% per year, whereas the average excess return across all firms in the sample is around 2.2% per year.

But this pooled regression assumes that the intercepts are the same for each firm and for each year. This may be an inappropriate assumption. Thus, next we (separately) introduce fixed and random effects to the model. To do so we click on **Statistics/Longitudinal/panel data/Linear models** and select the option **Linear regression (FE, RE, PA, BE)**. The linear regression command **xtreg** is a very flexible instruction in Stata as it allows you to fit random-effects models using the between regression estimator, fixed-effects models (using the within regression estimator), random-effects models using the GLS estimator (producing a matrix-weighted average of the between and within results), random-effects models using the Maximum-Likelihood estimator, and population-averaged models. Let us start with a fixed effect model. The dependent and independent variables remain the same as in the simple pooled regression so that the specification window should look like Figure 105.

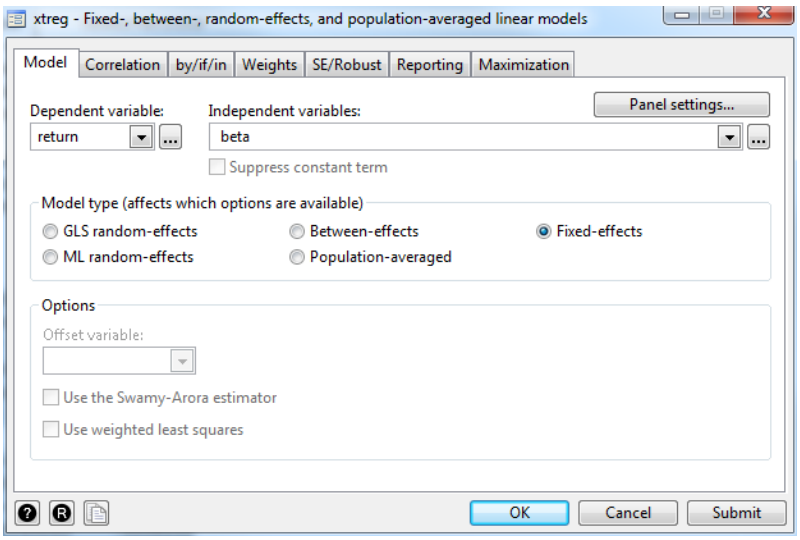


Figure 105: Specifying a Fixed Effects Model

Note that Stata offers many options to customise the model, including different standard error adjustments and weighting options. For now we will keep the default options. However, for future projects,

the correct adjustment of standard errors is often a major consideration at the model selection stage. We press **OK** and the following regression output should appear.

```
. xtreg return beta, fe

Fixed-effects (within) regression               Number of obs      =      8,856
Group variable: firm_ident                      Number of groups   =      1,734

R-sq:                                           Obs per group:
     within  = 0.0012                                         min =          1
     between = 0.0001                                         avg =        5.1
     overall = 0.0000                                         max =         11

                                                F(1,7121)          =       8.36
corr(u_i, Xb)  = -0.0971                        Prob > F           =     0.0039

------------------------------------------------------------------------------
      return |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        beta |  -.0118931   .0041139    -2.89   0.004    -.0199577   -.0038286
       _cons |   .0154962    .004581     3.38   0.001     .0065161    .0244763
-------------+----------------------------------------------------------------
     sigma_u |   .04139291
     sigma_e |    .0507625
         rho |   .39936854   (fraction of variance due to u_i)
------------------------------------------------------------------------------
F test that all u_i=0: F(1733, 7121) = 1.31                     Prob > F = 0.0000

.
```

We can see that the estimate on the beta parameter is negative and statistically significant here, while the intercept is positive and statistically significant. We now estimate a random effects model. For this, we simply select the option **GLS random-effects** in the **xtreg** specification window. We leave all other specifications unchanged and press **OK** to generate the regression output.

```
. xtreg return beta, re

Random-effects GLS regression                   Number of obs      =      8,856
Group variable: firm_ident                      Number of groups   =      1,734

R-sq:                                           Obs per group:
     within  = 0.0012                                         min =          1
     between = 0.0001                                         avg =        5.1
     overall = 0.0000                                         max =         11

                                                Wald chi2(1)       =       2.80
corr(u_i, X)   = 0 (assumed)                    Prob > chi2        =     0.0941

------------------------------------------------------------------------------
      return |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        beta |  -.0053994    .003225    -1.67   0.094    -.0117203    .0009216
       _cons |   .0063423   .0036856     1.72   0.085    -.0008814     .013566
-------------+----------------------------------------------------------------
     sigma_u |   .02845372
     sigma_e |    .0507625
         rho |   .23907489   (fraction of variance due to u_i)
------------------------------------------------------------------------------

.
```

The slope estimate is again of a different order of magnitude compared to both the pooled and the fixed effects regressions. As the results for the fixed effects and random effects models are quite different,

it is of interest to determine which model is more suitable for our setting. To check this, we use the Hausman test. The null hypothesis of the Hausman test is that the random effects (RE) estimator is indeed an efficient (and consistent) estimator of the true parameters. If this is the case, there should be no systematic difference between the RE and FE estimators and the RE estimator would be preferred as the more efficient estimator. In contrast, if the null is rejected, the fixed effect estimator needs to be applied.

To run the Hausman test we need to create two new variables containing the coefficient estimates of the fixed effects and the random effects model, respectively.[64] So let us first re-run the fixed effect model using the command **xtreg return beta, fe**. Once this model has been fitted, we click on **Statistics/Postestimation/Manage estimation results/Store current estimates in memory**. In the specification window we are asked to name the estimates that we would like to store (Figure 106). In this case, we name them **fixed** to later recognise them as belonging to the fixed effect estimator. We repeat this procedure for the random effects model by first re-running the model (using the command **xtreg return beta, re**) and storing the estimates under the name **random**.

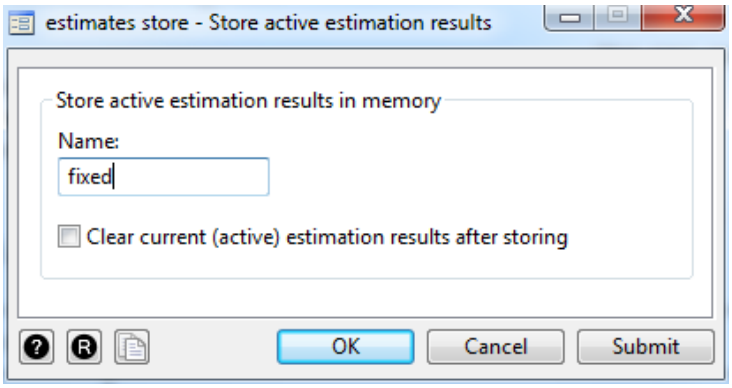


Figure 106: Storing Estimates from the Fixed Effects Model

Now we can specify the Hausman test. We click on **Statistics/Postestimation/Specification, diagnostic, and goodness-of-fit analysis** and select **Hausman specification test**.

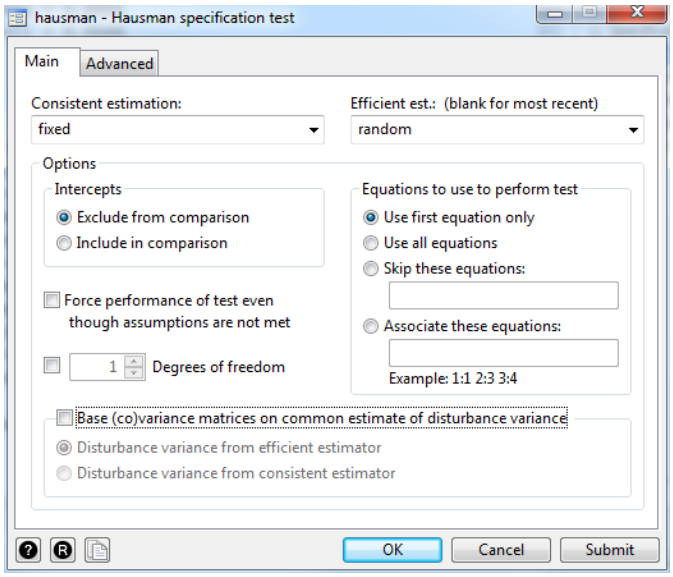


Figure 107: Specifying the Hausman test

[64] We base this Hausman test on the procedure described in the Stata manual under the the entry '[R] hausman'.

In the specification window we are asked to specify the consistent estimation and the efficient estimation. In our case the consistent estimates relate to the fixed effects model, i.e., **Consistent estimation: fixed**, and the efficient estimates relate to the random effects estimator, i.e., **Efficient est.: random** (Figure 107). We keep all other default settings and press **OK**. The output with the Hausman test results should appear as below.

```
. hausman fixed random

                 ---- Coefficients ----
             |      (b)          (B)            (b-B)     sqrt(diag(V_b-V_B))
             |    fixed       random       Difference          S.E.
-------------+------------------------------------------------------------
        beta |  -.0118931    -.0053994       -.0064938        .0025541
------------------------------------------------------------------------------
                           b = consistent under Ho and Ha; obtained from xtreg
            B = inconsistent under Ha, efficient under Ho; obtained from xtreg

    Test:  Ho:  difference in coefficients not systematic

                  chi2(1) = (b-B)'[(V_b-V_B)^(-1)](b-B)
                          =         6.46
                Prob>chi2 =       0.0110

.
```

The χ^2 value for the Hausman test is 6.46 with a corresponding p-value of 0.011. Thus, the null hypothesis that the difference in the coefficients is not systematic is rejected at the 5% level, implying that the random effects model is not appropriate and that the fixed effects specification is to be preferred.

21.1 Testing for Unit Roots and Cointegration in Panels

Reading: Brooks (2019, Section 11.8)

Stata provides a range of tests for unit roots within a panel structure. You can see the different options by selecting **Statistics/Longitudinal/panel data/Unit-root tests** and clicking on the drop-down menu for Tests in the specification window.[65] For each of the unit roots tests we can find the null and alternative hypotheses stated at the top of the test output. The Levin-Lin-Chu, Harris-Tzavalis, Breitung, Im-Pesaran-Shin, and Fisher-type tests have as the null hypothesis that all the panels contain a unit root. In comparison, the null hypothesis for the Hadri Lagrange multiplier (LM) test is that all the panels are (trend) stationary. Options allow you to include panel-specific means (fixed effects) and time trends in the model of the data-generating process.

For the panel unit root test we will use the six Treasury bill/bond yields from the '**fred.dta**' workfile. Before running any panel unit root or cointegration tests, it is useful to start by examining the results of individual unit root tests on each series, so we run the Dickey–Fuller GLS unit root tests (**dfgls**) on the levels of each yield series.[66]

You should find that for all series the test statistics are well below – 2.5 (using one lag) and thus the unit root hypothesis can be rejected at the 10% level.

As we know from the discussion above, unit root tests have low power in the presence of small samples, and so the panel unit root tests may provide different results. However, before performing a panel unit root test we have to transform the dataset into a panel format. Therefore, we need to stack

[65] Further details are provided in the **root** entry in the Stata manual.

[66] A description of how to run Dickey–Fuller GLS unit root tests is explained in section 16 of this guide.

all series below one another in one single data series. We first need to rename the series by putting a 'rate' in front of the series name:

rename GS3M rateGS3M

rename GS6M rateGS6M

rename GS1 rateGS1

rename GS3 rateGS3

rename GS5 rateGS5

rename GS10 rateGS10

Next, we reshape the data from a long to a wide format. Instead of copying and pasting the individual datasets we can use the Stata command '**reshape**' to do this job for us. We click on **Data/Create or change data/Other variable-transformation commands/Convert data between wide and long**. In the window that appears (Figure 108), we first select the type of transformation, i.e., **Long format from wide**.

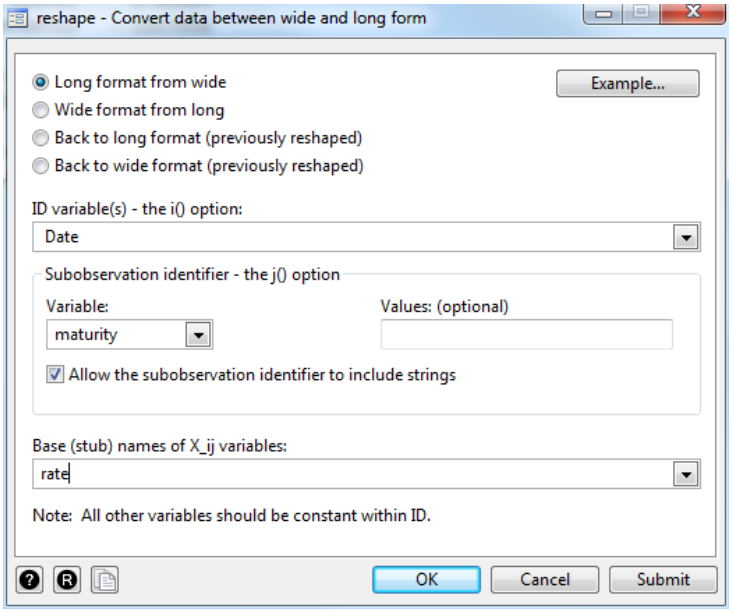


Figure 108: Reshaping the Dataset into a Panel Format

We specify the **ID variable(s): Date**. Then, we define the **Subobservation identifier** as the new **Variable: maturity** (which will become the panel id) containing the different maturities of the Treasury bill series. We also check the box **Allow the sub-observation identifier to include strings**. Finally, we specify the **Base (stub) names of X_ij variables:** which in our case is the **rate** in front of the renamed Treasury series of different maturities.[67] Now the window should resemble Figure 108 and we press **OK**.

We will find that the six individual treasury yield series have disappeared and there are now only three series in the dataset: 'Date', 'maturity' and 'rate'. We can have a look at the data using the *Data Editor*. We will find that our dataset now resembles a typical panel dataset with the series of yields for different maturities stacked below each other in the 'rate' variable, and 'maturity' serving as

[67]For more details and illustrative examples of how to use the **reshape** command, refer to the corresponding entry in the Stata manual.

the panel id. As Stata only allows numeric variables to be a panel id we need to transform the string variable 'maturity' into a numeric version. For this, we can use the Stata command '**encode**'. We can access this command by clicking on **Data/Create or change data/Other variable-transformation commands/Encode value labels from string variable**. In the specification window (Figure 109), we first need to tell Stata the **Source-string variable** which in our case is **maturity**. We want to create a numeric version of this variable named **maturity_num**. By clicking **OK**, the new variable is created. When checking the data type of the variable, e.g., by using the 'codebook' command, we should find that 'maturity_num' is a numeric variable whereas 'maturity' is (still) a string variable.

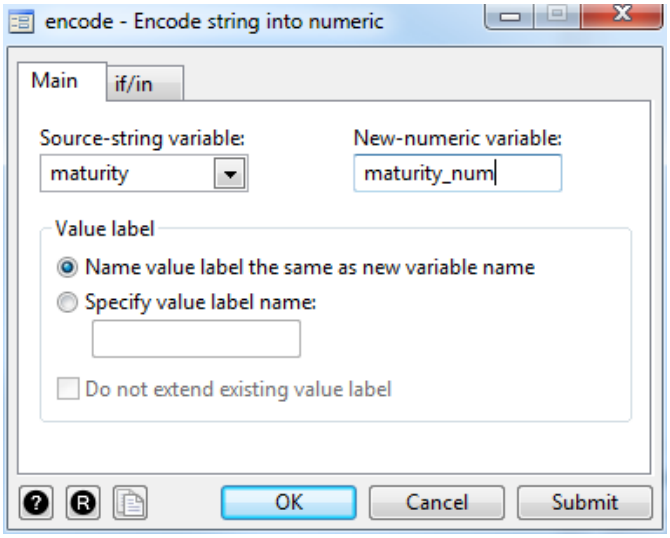


Figure 109: Encoding the Panel Variable

Now that all the data preparation has been done, we can finally perform the panel unit root test on the 'rate' series. We open the 'xtunitroot' specification window by selecting **Statistics/Longitudinal/-panel data/Unit-root tests**. As mentioned above, there is a variety of unit root tests that we can choose from (by clicking on the drop-down menu below **Test**) (Figure 110, left panel). For now, we keep the default test **Levin-Lin-Chu**. However, please feel free to test the sensitivities of our results to alternative test specifications. Next, we specify the **Variable: rate** as the variable on which we want to perform the test.

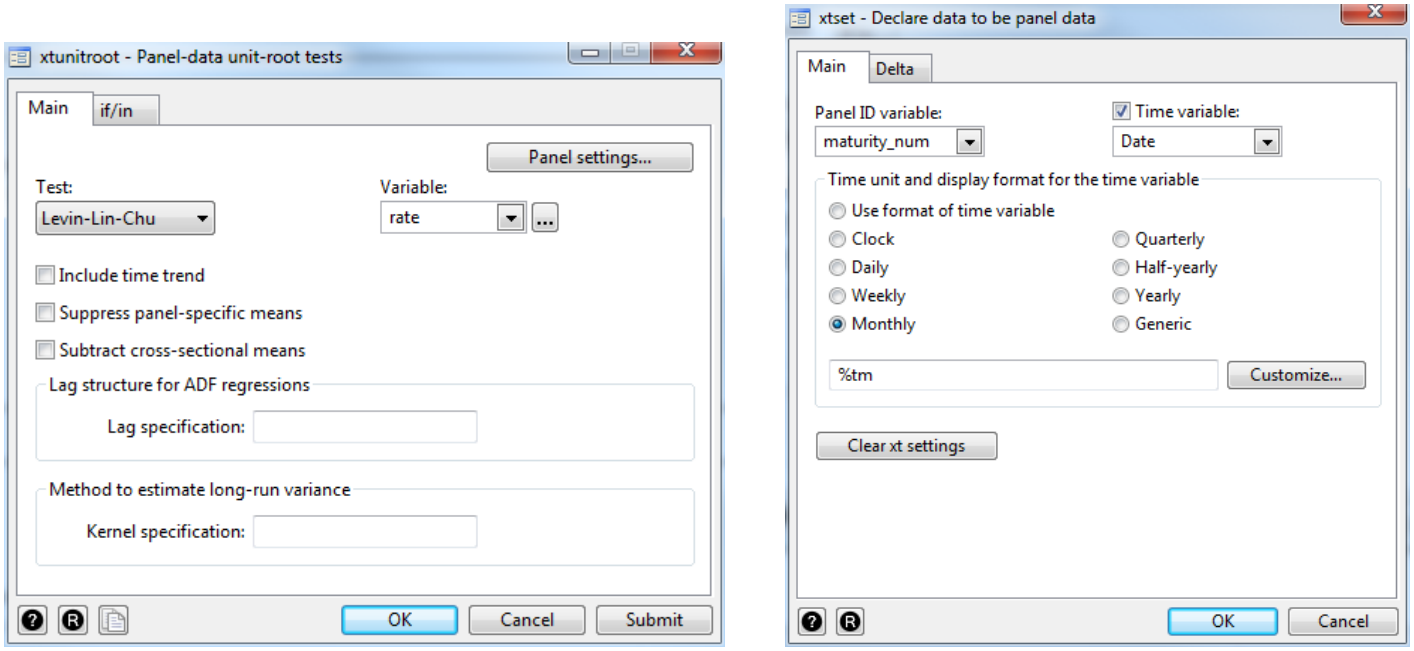


Figure 110: Specifying a Panel Unit Root Test

As a final step, we click on **Panel settings** to specify the panel id and time id in our dataset. In the window that appears (Figure 110, right panel), we define the **Panel ID variable: maturity_num** and the **Time variable: Date** and also tell Stata that our time variable is of **Monthly** frequency by selecting the respective option. We click **OK** to return to the main specification window and press **OK** again to perform the test. The test output of the Levin–Lin–Chu panel unit root test is presented below.

```
. xtunitroot llc rate

Levin-Lin-Chu unit-root test for rate
------------------------------------

Ho: Panels contain unit roots               Number of panels  =      6
Ha: Panels are stationary                   Number of periods =    437

AR parameter: Common                        Asymptotics: N/T -> 0
Panel means:  Included
Time trend:   Not included

ADF regressions: 1 lag
LR variance:     Bartlett kernel, 24.00 lags average (chosen by LLC)
------------------------------------------------------------------------------
                     Statistic      p-value
------------------------------------------------------------------------------
 Unadjusted t         -7.8581
 Adjusted t*          -6.5647        0.0000
------------------------------------------------------------------------------

.
```

As described above, the Levin–Lin–Chu test is based on the null hypothesis that the panels contain unit roots. It assumes a common p for the different panels. Looking at the test results, we find a test statistic of –6.56 with a corresponding p-value of 0.0000. Thus the unit root null is strongly rejected.

We can now re-run the panel unit root test using another test specification. What do you find? Do all tests arrive at the same solution regarding the stationarity or otherwise, of the rate series? In all cases, the test statistics are well below the critical values, indicating that the series contain unit roots. Thus the conclusion from the panel unit root tests are in line with the ones derived from the test

of the individual series. Some series contain a unit root and thus the panel unit root null hypothesis cannot be rejected. Note, however, that the additional benefits from using a panel in our case might be modest since the number of different panels ($N = 6$) is quite small, while the total number of time-series observations ($T = 437$) is relatively large.

It is also possible to perform panel cointegration tests in Stata. However, these cointegration tests do not come as a built-in Stata function but are in a user-written command, a so-called ado-file. One command for performing panel cointegration tests has been written by Persyn and Westerlund (2008) and is based on the four panel cointegration tests developed by Westerlund (2007). It is called **xtwest**. Another panel cointegration test is available via the ado-file **xtfisher** and is based on the test developed by Maddala and Wu (1999). Finally, you can perform the panel cointegration test **xtpedroni** developed in Pedroni (1999) and Pedroni (2001). Independent of which command you intend to use, you can install the ado-files by typing in **findit** followed by the name of the command, e.g., **findit xtwest**. You then need to follow the link that corresponds to the chosen command and select **click here to install**. We leave the implementation of the panel cointegration test for further studies.

22 Limited Dependent Variable Models

Reading: Brooks (2019, Chapter 12)

Estimating limited dependent variable models in Stata is very simple. The example that will be considered here concerns whether it is possible to determine the factors that affect the likelihood that a student will fail his/her MSc. The data comprise a sample from the actual records of failure rates for five years of MSc students at the ICMA Centre, University of Reading, contained in the spreadsheet 'MSc_fail.xls'. While the values in the spreadsheet are all genuine, only a sample of 100 students is included for each of the five years who completed (or not as the case may be!) their degrees in the years 2003 to 2007 inclusive. Therefore, the data should not be used to infer actual failure rates on these programmes. The idea for this is taken from a study by Heslop and Varotto (2007) which seeks to propose an approach to preventing systematic biases in admissions decisions.[68]

The objective here is to analyse the factors that affect the probability of failure of the MSc. The dependent variable ('fail') is binary and takes the value 1 if that particular candidate failed at first attempt in terms of his/her overall grade and 0 elsewhere. Therefore, a model that is suitable for limited dependent variables is required, such as a logit or probit.

The other information in the spreadsheet that will be used includes the age of the student, a dummy variable taking the value 1 if the student is female, a dummy variable taking the value 1 if the student has work experience, a dummy variable taking the value 1 if the student's first language is English, a country code variable that takes values from 1 to 10, a dummy that takes the value 1 if the student already has a postgraduate degree, a dummy variable that takes the value 1 if the student achieved an A-grade at the undergraduate level (i.e., a first-class honours degree or equivalent), and a dummy variable that takes the value 1 if the undergraduate grade was less than a B-grade (i.e., the student received the equivalent of a lower second-class degree).[69] The B-grade (or upper second-class degree) is the omitted dummy variable and this will then become the reference point against which the other grades are compared. The reason why these variables ought to be useful predictors of the probability of failure should be fairly obvious and is therefore not discussed. To allow for differences in examination rules and in average student quality across the five-year period, year dummies for 2004, 2005, 2006 and 2007 are created and thus the year 2003 dummy will be omitted from the regression model.

First, we import the dataset into Stata. To check that all series are correctly imported we can use the *Data Editor* to visually examine the imported data and the **codebook** command to get information on the characteristics of the dataset. All variables should be in the numeric format and overall there should be 500 observations in the dataset for each series with no missing observations. Also make sure to **save the workfile** in the '.dta'-format.

To begin with, suppose that we estimate a linear probability model of Fail on a constant, Age, English, Female, Work experience, A-Grade, Below-B-Grade, PG-Grade and the year dummies. This would be achieved simply by running a linear regression, using the command:

regress Fail Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004 Year2005 Year2006 Year2007

The results would appear as below.

[68] Note that since this example only uses a subset of their sample and variables in the analysis, the results presented below may differ from theirs. Since the number of fails is relatively small, I deliberately retained as many fail observations in the sample as possible, which will bias the estimated failure rate upwards relative to the true rate.

[69] The exact identities of the countries involved are not revealed in order to avoid any embarrassment for students from countries with high relative failure rates, except that Country 8 is the UK!

```
. regress Fail Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004 Year2005 Year2006 Year2007

      Source |       SS           df       MS      Number of obs   =       500
-------------+----------------------------------   F(11, 488)      =      3.15
       Model |  3.84405618        11  .349459653   Prob > F        =    0.0004
    Residual |  54.1779438       488  .111020377   R-squared       =    0.0663
-------------+----------------------------------   Adj R-squared   =    0.0452
       Total |    58.022         499  .116276553   Root MSE        =     .3332

------------------------------------------------------------------------------
        Fail |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         Age |   .0013219    .004336     0.30   0.761    -.0071976    .0098414
     English |  -.0200731   .0315276    -0.64   0.525    -.0820197    .0418735
      Female |  -.0293804   .0350533    -0.84   0.402    -.0982545    .0394937
WorkExperience|  -.0620281   .0314361    -1.97   0.049    -.1237948   -.0002613
     Agrade |  -.0807004   .0377201    -2.14   0.033    -.1548142   -.0065866
BelowBGrade |   .0926163   .0502264     1.84   0.066    -.0060703    .1913029
    PGDegree |   .0286615   .0474101     0.60   0.546    -.0644918    .1218147
    Year2004 |   .0569098   .0477514     1.19   0.234    -.0369139    .1507335
    Year2005 |  -.0111013   .0483674    -0.23   0.819    -.1061354    .0839329
    Year2006 |   .1415806   .0480335     2.95   0.003     .0472025    .2359587
    Year2007 |   .0851503   .0497275     1.71   0.087     -.012556    .1828567
       _cons |   .1038805   .1205279     0.86   0.389    -.1329372    .3406983
------------------------------------------------------------------------------

.
```

While this model has a number of very undesirable features as discussed in Brooks (2019), it would nonetheless provide a useful benchmark with which to compare the more appropriate models estimated below.

Next, we estimate a probit model and a logit model using the same dependent and independent variables as above. We begin with the logit model by clicking on **Statistics/Binary outcomes** and choosing **Logistic regression**. First, we need to specify the dependent variable (**Fail**) and independent variables (**Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004 Year2005 Year2006 Year2007**) as shown in Figure 111.

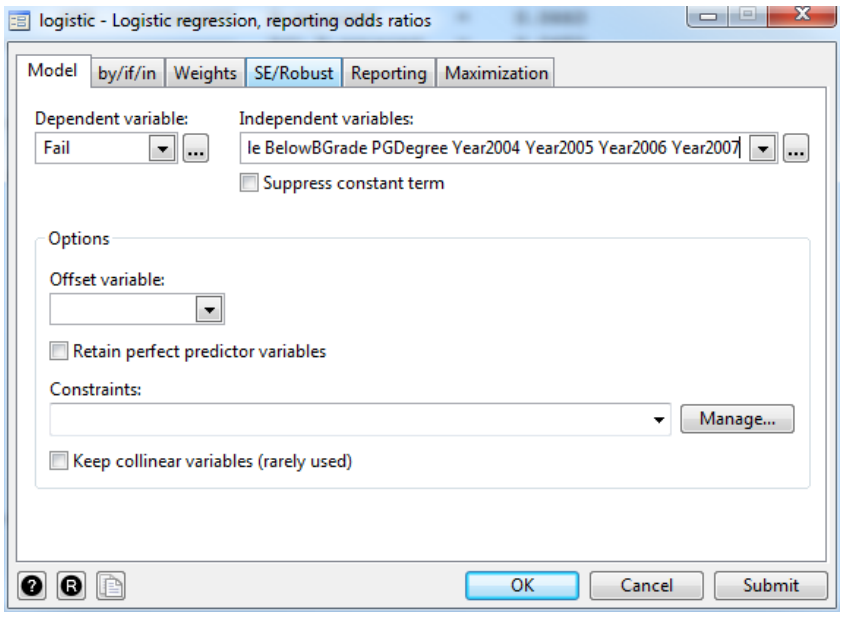


Figure 111: Specifying a Logit Model

Next, we want to specify the standard error correction. To do so, we click on the **SE/Robust** tab and select the **Standard error type: Robust** from the drop-down menu (Figure 112). This option will ensure that the standard error estimates are robust to heteroscedasticity.

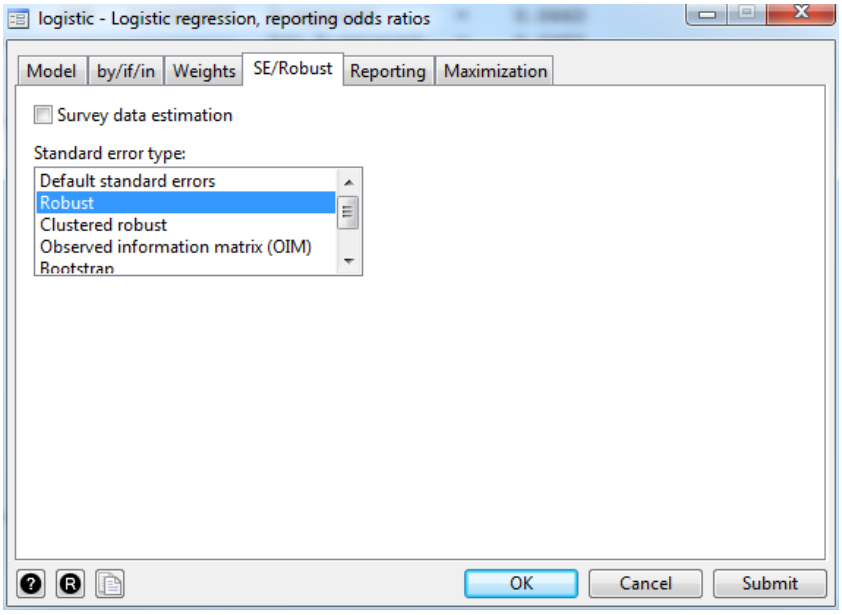


Figure 112: Specifying Standard Errors for a Logit Model

Using the other tabs you can also change the optimisation method and convergence criterion. However, we do not need to make any changes from the default, but simply click **OK**. The output for the logit regression should appear as below.

```
. logistic Fail Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004
>  Year2005 Year2006 Year2007, vce(robust)

Logistic regression                             Number of obs     =        500
                                                Wald chi2(11)     =      31.91
                                                Prob > chi2       =     0.0008
Log pseudolikelihood = -179.71667               Pseudo R2         =     0.0875

------------------------------------------------------------------------------
             |               Robust
        Fail | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         Age |   1.011072   .0464445     0.24   0.811     .9240204    1.106325
     English |   .8477939   .2503922    -0.56   0.576     .4752167    1.512477
      Female |   .7161297   .2578792    -0.93   0.354      .353568    1.450476
WorkExperience|   .5662222   .1638188    -1.97   0.049     .3211576    .9982875
     Agrade |   .3378916   .1665927    -2.20   0.028     .1285596    .8880762
 BelowBGrade |   1.754793   .6788647     1.45   0.146     .8221053    3.745626
    PGDegree |   1.236252   .5262321     0.50   0.618     .5367577    2.847316
    Year2004 |   1.921693    .930147     1.35   0.177     .7441857    4.962341
    Year2005 |   .8320819   .4656394    -0.33   0.743     .2778608    2.491753
    Year2006 |   3.478413   1.633688     2.65   0.008     1.385485    8.732943
    Year2007 |   2.340634   1.128883     1.76   0.078     .9094959    6.023742
       _cons |   .1047301   .1279304    -1.85   0.065     .0095567    1.147717
------------------------------------------------------------------------------
Note: _cons estimates baseline odds.

.
```

Next, we estimate the above model as a probit model. We click on **Statistics/Binary outcomes** but

now select the option **Probit regression**. We input the same model specifications as in the logit case and again select robust standard errors. The output of the probit model is presented below.

```
. probit Fail Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004
  Year2005 Year2006 Year2007, vce(robust)
Iteration 0:   log pseudolikelihood = -196.96021
Iteration 1:   log pseudolikelihood = -180.03898
Iteration 2:   log pseudolikelihood = -179.45746
Iteration 3:   log pseudolikelihood = -179.45634
Iteration 4:   log pseudolikelihood = -179.45634

Probit regression                               Number of obs     =        500
                                                Wald chi2(11)     =      33.51
                                                Prob > chi2       =     0.0004
Log pseudolikelihood = -179.45634               Pseudo R2         =     0.0889
```

Fail	Coef.	Robust Std. Err.	z	P>|z|	[95% Conf.	Interval]
Age	.005677	.0225819	0.25	0.802	-.0385828	.0499368
English	-.0937923	.1563826	-0.60	0.549	-.4002965	.212712
Female	-.1941073	.1863877	-1.04	0.298	-.5594205	.171206
WorkExperience	-.3182465	.1514845	-2.10	0.036	-.6151507	-.0213424
Agrade	-.5388141	.2313792	-2.33	0.020	-.992309	-.0853191
BelowBGrade	.3418026	.2195206	1.56	0.119	-.0884498	.772055
PGDegree	.1329571	.2261508	0.59	0.557	-.3102903	.5762045
Year2004	.3496632	.2416917	1.45	0.148	-.1240439	.8233702
Year2005	-.1083299	.2687962	-0.40	0.687	-.6351607	.4185009
Year2006	.6736117	.2387747	2.82	0.005	.2056219	1.141602
Year2007	.4337853	.248178	1.75	0.080	-.0526348	.9202053
_cons	-1.28721	.6101132	-2.11	0.035	-2.483009	-.0914097

```
.
```

As can be seen, for both models the pseudo-R^2 values are quite small at just below 9%, although this is often the case for limited dependent variable models.

Turning to the parameter estimates on the explanatory variables, we find that only the work experience and A-grade variables and two of the year dummies have parameters that are statistically significant, and the Below B-grade dummy is almost significant at the 10% level in the probit specification (although less so in the logit model). However, the proportion of fails in this sample is quite small (13.4%),[70] which makes it harder to fit a good model than if the proportion of passes and fails had been more evenly balanced. Note that Stata offers a variety of goodness-of-fit and classification tests, such as the Hosmer–Lemeshow goodness-of-fit test. You can access these tests by selecting **Statistics/Binary outcomes/Postestimation** and then choosing the test that you would like to perform.

A further test on model adequacy is to produce a set of in-sample forecasts – in other words, to construct the fitted values. To do so, we open the 'Postestimation Selector' and click on **Predictions/Probabilities, linear predictions and their SEs, etc.**.

In the 'predict' specification window that appears we define the name of the new variable that contains the predicted values of 'Fail' as **New variable name: Failf** (Figure 113). It should contain the **Probability of a positive outcome**, i.e., the candidate fails (Fail = 1), which is the default so we do not need to make any changes. We click **OK** and the new series 'Failf' should appear in the *Variables* window.

[70] Note that you can retrieve the number of observations for which 'Fail' takes the value 1 by using the command **count if Fail==1**.

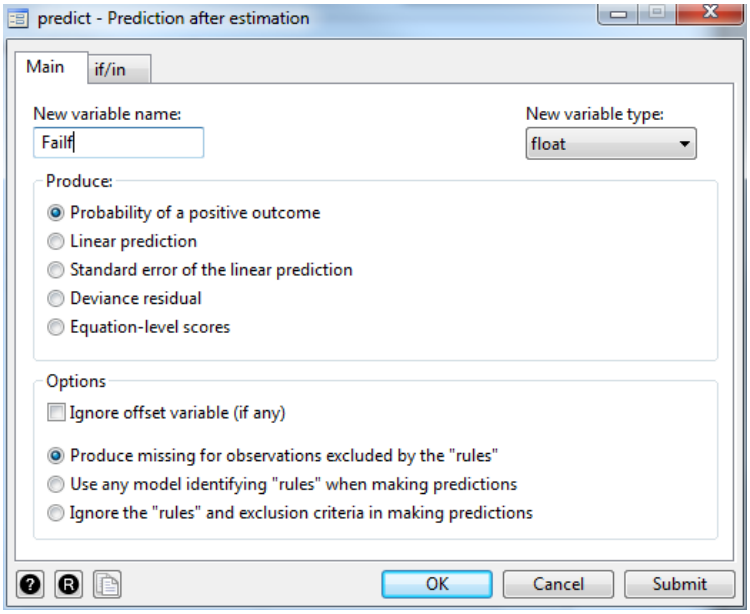


Figure 113: Creating Fitted Values from the Failure Probit Regression

To visually inspect the fitted values, we want to plot them as a graph. However, since our dataset does not contain a time variable that we can plot the 'Failf' series against, we create a new series that contains the row number of the respective observation. We can do so by using the command **generate seqnum=_n**, which specifies that the new variable 'seqnum' should contain the row number which is denoted by '_n'.

We can now create a plot of the fitted values by selecting **Graphics/Twoway graph (scatter, line, etc.)**. In the line plot specification window, we click on **Create...** and then select **Basic plots: Line** as well as **Y variable: Failf** and **X variable: seqnum**. The resulting plot should resemble that in Figure 114.

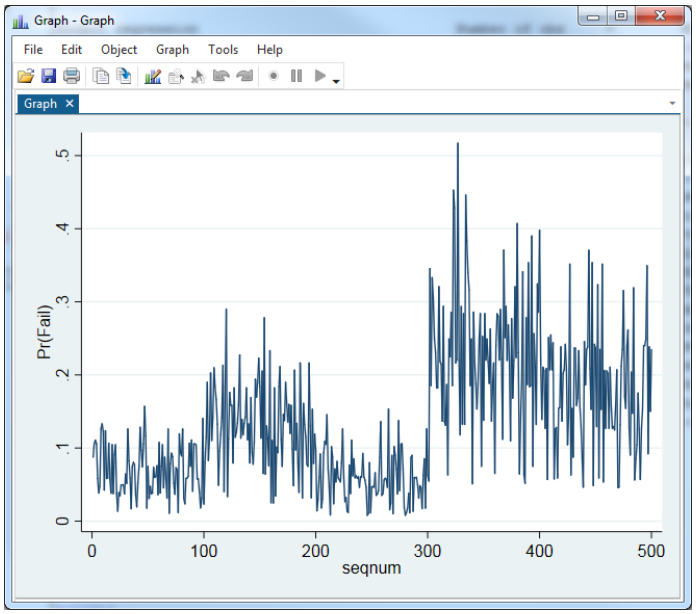


Figure 114: Graph of the Fitted Values from the Failure Probit Regression

The unconditional probability of failure for the sample of students we have is only 13.4% (i.e., only 67 out of 500 failed), so an observation should be classified as correctly fitted if either $y_i = 1$ and $\hat{y}_i > 0.134$

or $y_i = 0$ and $\hat{y}_i < 0.134$.

The easiest way to evaluate the model in Stata is to click **Statistics/Binary outcomes/Postestimation/Classification statistics after logistic/logit/probit/ivprobit**. In the specification window that appears, we select the option **Report various summary stats. including the classification table (classification)** and keep the default **Use estimation sample**. We define the **Positive outcome threshold** to be **0.134** (Figure 115). Then we click **OK** and the table should appear as below.

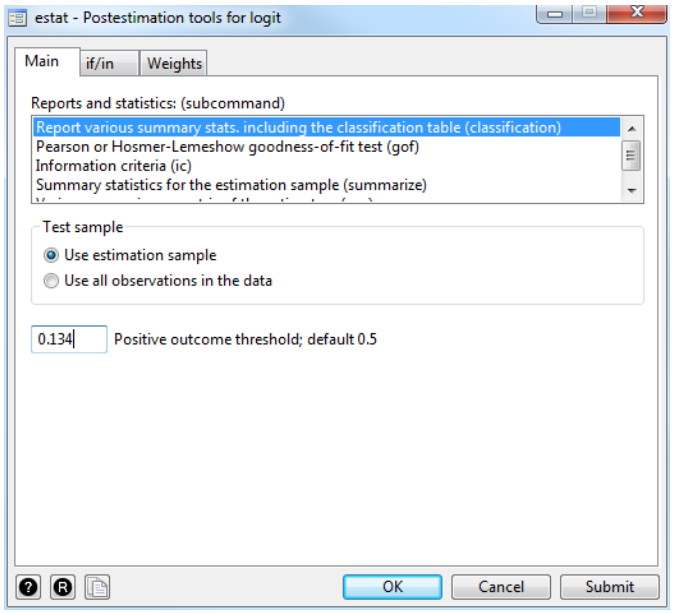


Figure 115: Generating a Classification Table for the Probit Model

```
. estat classification, cutoff(0.134)

Probit model for Fail

                 ———— True ————
Classified |         D            ~D |       Total
-----------+------------------------+------------
     +     |        46           155 |         201
     -     |        21           278 |         299
-----------+------------------------+------------
   Total   |        67           433 |         500

Classified + if predicted Pr(D) >= .134
True D defined as Fail != 0
-------------------------------------------------
Sensitivity                     Pr( +| D)   68.66%
Specificity                     Pr( -|~D)   64.20%
Positive predictive value       Pr( D| +)   22.89%
Negative predictive value       Pr(~D| -)   92.98%
-------------------------------------------------
False + rate for true ~D        Pr( +|~D)   35.80%
False - rate for true D         Pr( -| D)   31.34%
False + rate for classified +   Pr(~D| +)   77.11%
False - rate for classified -   Pr( D| -)    7.02%
-------------------------------------------------
Correctly classified                        64.80%
-------------------------------------------------

.
```

From the classification table we can identify that of the 67 students that failed, the model correctly predicted 46 of them to fail (and it also incorrectly predicted that 21 would pass). Of the 433 students who passed, the model incorrectly predicted 155 to fail and correctly predicted the remaining 278 to pass. Overall, we could consider this a reasonable set of (in sample) predictions with 64.8% of the total predictions correct, comprising 64.2% of the passes correctly predicted as passes and 68.66% of the fails correctly predicted as fails.

It is important to note that we cannot interpret the parameter estimates in the usual way (see the discussion in Brooks (2019)). In order to be able to do this, we need to calculate the marginal effects.

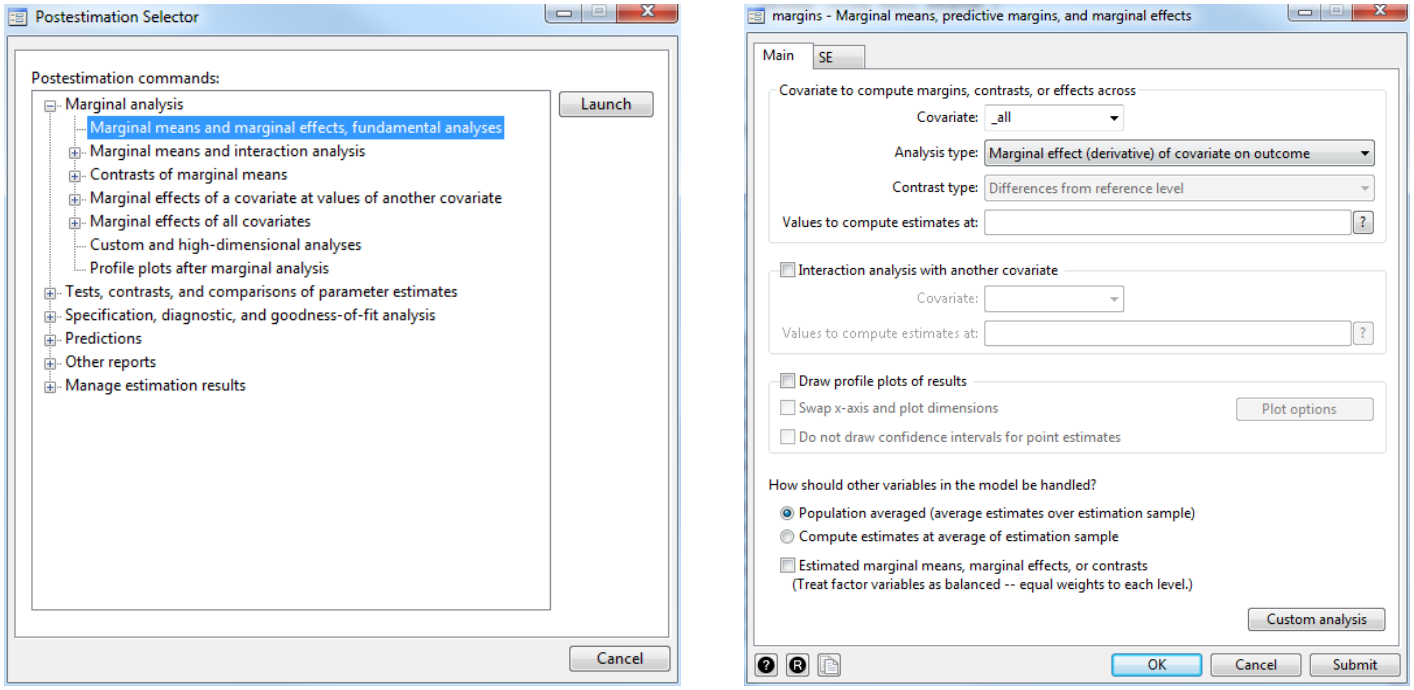


Figure 116: Generating Marginal Effects

Stata has an in-built command that allows to calculate marginal effects which can be accessed via **Statistics/Postestimation/Marginal analysis/Marginal means and marginal effects, fundamental analyses** (Figure 116, left panel). In the 'margins' specification window that appears (Figure 116, right panel) we first need to specify the **Covariate** for which we want to compute marginal effects. As we want to generate marginal effects for all of the explanatory variables we type **_all** in the dialogue box. Next we select the **Analysis type: Marginal effect (derivative) of covariate on outcome**. Leaving all other options to their default options and pressing **OK**, Stata should generate the table of marginal effects and corresponding statistics as shown on the following page.

```
. margins, dydx(_all)

Average marginal effects                        Number of obs     =        500
Model VCE    : Robust

Expression   : Pr(Fail), predict()
dy/dx w.r.t. : Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004 Year2005 Year2006 Year2007
```

	dy/dx	Delta-method Std. Err.	z	P>|z|	[95% Conf.	Interval]
Age	.0011179	.0044466	0.25	0.802	-.0075974	.0098331
English	-.0184688	.0307689	-0.60	0.548	-.0787748	.0418372
Female	-.038222	.036781	-1.04	0.299	-.1103115	.0338674
WorkExperience	-.0626665	.0298548	-2.10	0.036	-.1211809	-.0041521
Agrade	-.1060989	.0453505	-2.34	0.019	-.1949843	-.0172135
BelowBGrade	.067305	.0431667	1.56	0.119	-.0173001	.1519101
PGDegree	.0261808	.0445415	0.59	0.557	-.0611189	.1134806
Year2004	.0688528	.0477053	1.44	0.149	-.0246479	.1623536
Year2005	-.0213314	.0529625	-0.40	0.687	-.125136	.0824732
Year2006	.1326422	.0467502	2.84	0.005	.0410134	.2242709
Year2007	.0854175	.0486634	1.76	0.079	-.009961	.180796

.

We can **repeat this exercise for the logit model** using the same procedure as above. Note that we need to re-run the logit model first and then calculate marginal effects in the same way as described for the probit model. If done correctly, the table of marginal effects should resemble the following:

```
. margins, dydx(_all)

Average marginal effects                        Number of obs     =        500
Model VCE    : Robust

Expression   : Pr(Fail), predict()
dy/dx w.r.t. : Age English Female WorkExperience Agrade BelowBGrade PGDegree Year2004 Year2005 Year2006 Year2007
```

	dy/dx	Delta-method Std. Err.	z	P>|z|	[95% Conf.	Interval]
Age	.0011862	.0049468	0.24	0.810	-.0085095	.0108818
English	-.0177866	.0318111	-0.56	0.576	-.0801353	.044562
Female	-.0359674	.0389302	-0.92	0.356	-.1122691	.0403343
WorkExperience	-.0612683	.031165	-1.97	0.049	-.1223506	-.000186
Agrade	-.1168805	.0530145	-2.20	0.027	-.220787	-.012974
BelowBGrade	.060577	.0416694	1.45	0.146	-.0210936	.1422476
PGDegree	.0228459	.045919	0.50	0.619	-.0671536	.1128454
Year2004	.070364	.0523332	1.34	0.179	-.0322071	.1729352
Year2005	-.0198017	.0603289	-0.33	0.743	-.1380442	.0984407
Year2006	.1342824	.0505192	2.66	0.008	.0352666	.2332982
Year2007	.0916083	.0517308	1.77	0.077	-.0097822	.1929987

.

Looking at the results, we find that not only are the marginal effects for the probit and logit models quite similar in value, they also closely resemble the coefficient estimates obtained from the linear probability model estimated earlier in the section.

Now that we have calculated the marginal effects, these values can be intuitively interpreted in terms

of how the variables affect the probability of failure. For example, an age parameter value of around 0.0012 implies that an increase in the age of the student by one year would increase the probability of failure by 0.12%, holding everything else equal, while a female student is around 3.5% less likely than a male student with otherwise identical characteristics to fail. Having an A-grade (first class) in the bachelors' degree makes a candidate around 11% less likely to fail than an otherwise identical student with a B-grade (upper second-class degree). Finally since the year 2003 dummy has been omitted from the equations, this becomes the reference point. So students were more likely in 2004, 2006 and 2007, but less likely in 2005, to fail the MSc than in 2003.

23 Simulation Methods

23.1 Deriving Critical Values for a Dickey–Fuller Test Using Simulation

Reading: Brooks (2019, Sections 13.1 – 13.7)

In this and the following subsections we will use simulation techniques in order to model the behaviour of financial series. In this first example, our aim is to develop a set of critical values for Dickey–Fuller test regressions. Under the null hypothesis of a unit root, the test statistic does not follow a standard distribution, and therefore a simulation would be required to obtain the relevant critical values. Obviously, these critical values are well documented, but it is of interest to see how one could generate them. A very similar approach could then potentially be adopted for situations where there has been less research and where the results are relatively less well known.

The simulation would be conducted in the following four steps:

1. Construct the data generating process under the null hypothesis – that is, obtain a series for y that follows a unit root process. This would be done by:
   - Drawing a series of length T, the required number of observations, from a normal distribution. This will be the error series, so that $u_t \sim N(0, 1)$.
   - Assuming a first value for y, i.e., a value for y at time $t = 1$.
   - Constructing the series for y recursively, starting with y_2, y_3, and so on

$$\begin{aligned} y_2 &= y_1 + u_2 \\ y_3 &= y_2 + u_3 \\ &\vdots \\ y_T &= y_{T-1} + u_T \end{aligned}$$

2. Calculating the test statistic, τ.
3. Repeating steps 1 and 2 N times to obtain N replications of the experiment. A distribution of values for τ will be obtained across the replications.
4. Ordering the set of N values of τ from the lowest to the highest. The relevant 5% critical value will be the 5th percentile of this distribution.

Some Stata code for conducting such a simulation is given below. The simulation framework considers a sample of 1,000 observations and Dickey–Fuller regressions with no constant or trend, a constant but no trend, and a constant and a trend. 50,000 replications are used in each case, and the critical values for a one-sided test at the 1%, 5% and 10% levels are determined. The code can be found pre-written in a Stata do-file entitled **'dickey_fuller_simulation.do'**.

Stata programs are simply sets of instructions saved as plain text, so that they can be written from within Stata, or using a word processor or text editor. There are two types of Stata programs, do-files and ado-files. The latter are equivalent to user-written commands and once installed, can be used like any other Stata command such as **summarize** or **regress**. The former (do-files) need to be opened every time the user wants to run the set of commands and can be interactively adjusted. We will only deal with do-files and leave the issue of programming ado-files for more advanced Stata users. To run a do-file, we open the *Do-File Editor* using the respective symbol in the Stata menu. In the window that

appears we click on **File/Open...** and select the 'dickey_fuller_simulation.do'. We should now be able to see the set of Stata commands.

The different colours indicate different characteristics of the instructions. All expressions in blue represent Stata commands (such as 'summarize' or 'regress') and information on the individual commands can be obtained typing **help** followed by the respective command in the *Command* window. Expressions in red that are expressed in double quotes mark strings and might refer to file names or paths, value labels of variables or string values of variables. Variables in light blue are a form of auxiliary variable and usually their value is substituted for a predefined content.[71]

Finally, text expressed in green represents comments made by the Stata user. Comments are not part of the actual instructions but rather serve to explain and describe the Stata codes. There are different ways to create comments in Stata, either by beginning the comment with * or // or placing the comment between /* and */ delimiters.[72]

The following lines of code are taken from the do-file **'dickey_fuller_simulation.do'** which creates critical values for the Dickey–Fuller test. The discussion below explains the function of the command line.

```
1  /*  DERIVING CRITICAL VALUES FOR A DICKEY-FULLER TEST
2    USING MONTE CARLO SIMULATIONS */
3  clear
4  tempname tstats
5  set seed 12345
6  postfile `tstats' t_none t_const t_trend using "D:\Programming Guide\Stata Guide\
       results.dta", replace
7  local T=1200
8  local N=50000
9  quietly {
10     forvalues i=1/`N' {
11     drop _all
12     set obs `T'
13     generate y=0 in 1
14     replace y=y[_n-1]+rnormal() in 2/`T'
15     generate dy=y-y[_n-1] in 2/`T'
16     generate lagy=y[_n-1]
17     generate t=_n-200 in 201/`T'
18     // Regression with lag and no constant
19     regress dy lagy, noconstant
20     scalar t_none=_b[lagy]/_se[lagy]
21     // Regression with lag and constant
22     regress dy lagy
23     scalar t_const=_b[lagy]/_se[lagy]
24     // Regression with lag, trend and constant
25     regress dy lagy t
26     scalar t_trend=_b[lagy]/_se[lagy]
27     post `tstats' (t_none) (t_const) (t_trend)
28     }
29 }
30 postclose `tstats'
31 use "D:\Programming Guide\Stata Guide\results.dta", clear
32 describe
33 tabstat t_none t_const t_trend, statistics( p1 p5 p10 ) columns(statistics)
```

To run the program we can click on the two very right buttons in the symbol menu of the *Do-file Editor*. We have two options: (a) **Execute Selection quietly (run)** which will run the code but without

[71] Macros are commonly used in Stata programming and can be applied to a variety of contexts. For more information on the characteristics of Macros and their use in Stata, refer to the respective entry in the Stata manual.

[72] For more details on how to create comments in Stata, refer to the corresponding entry in the Stata manual.

showing the output in the Stata Output window, and (b) **Execute (do)** which will progressively report the output of the code. The latter is especially useful for debugging programs or running short programs, though it leads to a slower execution of the program than when running it in the quiet mode. We can also choose to run only parts of the instructions instead of the entire set of commands by highlighting the lines of commands that we would like to perform and then selecting either of the two execution options.

The first two lines are a simple comment that explains the purpose and contents of the do-file.[73] The lines that follow contain the actual commands that perform the manipulations of the data. The first couple of lines are mere preparation for the main simulation but are necessary to access the simulated critical values later on. Line 3 removes data and value labels from the memory to make sure there is no confusion with commands run before this program. Line 4 is an auxiliary command which tells Stata to create a temporary variable called **'tstats'** that will be used within the program but will be automatically deleted once the program is finished (both successfully and forcefully).

Line 5 **'set seed 12345'** sets the so-called random number seed. This is necessary to be able to replicate the exact t-values created with this program on any other computer and for any other try. While this explanation might not be very informative at this stage, the command serves to define the starting value for draws from a standard normal distribution which are necessary in later stages to create variables that follow a standard normal distribution.

In line 6, we tell Stata to create a file that contains the t-values that will be generated based on the different regressions. Specifically, we tell Stata that this file of results will contain three variables t_none t_const t_trend. **'t_none'**, **'t_const'** and **'t_trend'** will contain the t-values for three different regression models resembling the different unit root specifications: (a) without a constant or trend, (b) with a constant but no trend, and (c) with a constant and a trend, respectively. We also specify the location and the name of the file where the results should be stored, namely `"D:\Programming Guide\Stata Guide\results.dta"`. When running the program on your own computer you will have to adjust the location of the file according to your computer settings.

Lines 7 and 8 set up the conditions for the loop, i.e., the number of repetitions N that will be performed and the number of observations T. Loops are always indicated by braces; the set of commands over which the loop is performed is contained within the curly brackets (**{ }**). For example, in our command the loops end in lines 21 and 22.

Before turning to the specific conditions of the loop, let us have a look at the set of commands that we want to perform the loop over, i.e., the commands that generate the t-values for the Dickey–Fuller regressions. They are stated in lines 9 to 29. Line 11 **'drop _all'** tells Stata that it should drop all variables and data that it currently has in memory so that we start with a completely empty dataset. In the next line (**'set obs 'T"**) we specify that we will create a new dataset that contains T observations, which we predefined as 1200. Lines 13 to 17 are commands to generate the variables that will be used in the regressions.

Line 13 **'generate y=0 in 1'** creates a new variable **y** that takes the value 0 for the first observation and contains missing values for all remaining observations. Line 14 specifies the remaining values for **'y'** for observations 2 to 1,200, namely a random walk-series that follows a unit root process. Recall that a random walk process is defined as the past value of the variable plus a standard normal error term. It is very easy to construct such a series, the previous value of the 'y1' variable is referred to as **'y[_n-1]'** and the standard normal variate is added using the Stata function **'rnormal()'**. Note that **'_n'** is a so-called system variable that can be referred to in each Stata dataset. It indicates the number of the observation, e.g., **'_n[1]'** refers to the first observation, **'_n[2]'** to the second observation, etc. It can actively be referred to in Stata commands to indicate a certain observation as in our case.

[73] Adding comments to your do-files is a useful practice and proves to be particularly useful if you revisit analyses that you have first carried out some time ago as they help you to understand the logic of the commands and steps.

In lines 15 and 16 we generate first differences and lagged values of 'y1', respectively. Note that when generating random draws it sometimes takes a while before the constructed series have the properties that they should, and therefore, when generating the regressions, we exclude the first 200 observations, which is indicated by the term **'in 201/'T"** in line 17. As Stata does not have a built-in trend that can be added to its OLS regression command 'regress' we manually create a variable 't' that follows a linear trend. This is done in line 17. 't' takes the value 1 for observations 201 and increases by a value of 1 for all consecutive observations.

Lines 19–20, 22–23 and 25–26 contain the regression commands to generate the t-values. Line 19 contains the regression without constant (specified by the **', noconstant'** term) and trend. In particular, the line contains a regression of the first difference of **'y'** on the lagged value of 'y'. Line 22 refers to the DF regression with constant but without trend. And in line 25 the trend variable **'t'** is added so that the overall model contains both a constant and a linear trend as additional right-hand-side variables (besides **'lagy'**). Lines 20, 23 and 26 generate the t-values corresponding to the particular models. The command **scalar** indicates that a scalar value will be generated and the expression on the RHS of the equals sign defines the value of the scalar. It is the formula for computing the t-value, namely the coefficient estimate on **'lagy'** (i.e., **'_b[lagy]'**) divided by the standard error of the coefficient (i.e., **'_se[lagy]'**). Line 27 tells Stata that the t-values **'t_none', 't_const'** and **'t_trend'** for the three models should be posted to the **'results.dta'** file (which we have created in line 6).

If we were to execute this set of commands one time, we would generate one t-value for each of the models. However, our aim is to get a large number of t-statistics in order to have a distribution of values. Thus, we need to repeat the set of commands for the desired number of repetitions. This is done by the loop command **forvalues** in line 10. It states that the set of commands included in the braces will be executed 50,000 times (**'i=1/50000'**). Note that for each of these 50,000 repetitions a new set of t-values will be generated and added to the **'results.dta'** file so that the final version of the file will contain three variables ('t_none', 't_const', 't_trend') with 50,000 observations each. Finally, the **'quietly'** in line 9 tells Stata not to produce the output for the 50,000 repetitions of the DF regressions but execute the commands "quietly". **'postclose 'tstats"** in line 30 signals Stata that no further values will be added to the **'results.dta'** file.

In line 31, we tell Stata to open the file **'results.dta'** containing the 50,000 observations of t-values. The command **'describe'** gives us information on the characteristics of the dataset and serves merely as a check that the program has been implemented and executed successfully. Finally, line 33 provides us with the critical values for the three different models as it generates the 1st, 5th and 10th percentile for the three variables 't_none', 't_const', 't_trend'.

To run this program, we click on the button **'Execute (do)'** on the very right in the toolbar of the **Data Editor**. Note that due to the total number of 50,000 repetitions running this command will take some time.[74]

The critical values obtained by running the above program, which are virtually identical to those found in the statistical tables in Brooks (2019), are presented in Table 3 (to two decimal places). This is to be expected, for the use of 50,000 replications should ensure that an approximation to the asymptotic behaviour is obtained. For example, the 5% critical value for a test regression with no constant or trend and 500 observations is −1.93 in this simulation, and −1.95 in Fuller (1976).

[74]By changing the value for N in line 8, you can see how the number of iteration effects the results.

Table 3: Simulated Critical Values for a Dickey–Fuller Test

	1%	5%	10%
No constant, no trend (**t_none**)	−2.56	−1.93	−1.61
Constant, no trend (**t_const**)	−3.44	−2.87	−2.57
Constant and trend (**t_trend**)	−3.98	−3.42	−3.13

Although the Dickey–Fuller simulation was unnecessary in the sense that the critical values for the resulting test statistics are already well known and documented, a very similar procedure could be adopted for a variety of problems. For example, a similar approach could be used for constructing critical values or for evaluating the performance of statistical tests in various situations.

23.2 Pricing Asian Options

Reading: Brooks (2019, Section 13.8)

In this subsection, we will apply Monte Carlo simulations to price Asian options. The steps involved are:

1. *Specify a data generating process for the underlying asset.* A random walk with drift model is usually assumed. Also specify the assumed size of the drift component and the assumed size of the volatility parameter. Select a strike price K, and a time to maturity, T.

2. Draw a series of length T, the required number of observations for the life of the option, from a normal distribution. This will be the *error series*, so that $\epsilon_t \sim N(0, 1)$.

3. Form a series of observations of length T on the *underlying asset*.

4. *Observe the price of the underlying asset at maturity observation* T. For a call option, if the value of the underlying asset on maturity date $P_t \leq K$, the option expires worthless for this replication. If the value of the underlying asset on maturity date $P_t > K$, the option expires in the money, and has a value on that date equal to $P_T - K$, which should be discounted back to the present using the risk-free rate.

5. Repeat steps 1 to 4 a total of N times, and take the average value of the option over N replications. This average will be the *price of the option*.

A sample of Stata code for determining the value of an Asian option is given below. The example is in the context of an arithmetic Asian option on the FTSE 100, and two simulations will be undertaken with different strike prices (one that is out of the money forward and one that is in the money forward). In each case, the life of the option is six months, with daily averaging commencing immediately, and the option value is given for both calls and puts in terms of index options. The parameters are given as follows, with dividend yield and risk-free rates expressed as percentages:

Simulation 1: Strike = 6500
Spot FTSE = 6,289.70
Future FTSE = 6,405.35
risk-free rate = 6.24
Dividend yield = 2.42
Implied Volatility = 26.52

Simulation 2: Strike = 5500
Spot FTSE = 6,289.70
Future FTSE = 6,405.35
risk-free rate = 6.24
Dividend yield = 2.42
Implied Volatility = 34.33

All experiments are based on 25,000 replications and their antithetic variates (total: 50,000 sets of draws) to reduce Monte Carlo sampling error. Some sample code for pricing an Asian option for normally distributed errors using Stata is given as follows:

```
/*PRICING AN ASIAN OPTION USING MONTE CARLO SIMULATIONS*/
clear
tempname prices
set seed 123456
postfile `prices' put_val call_val using "D:\Programming Guide\Stata Guide\
    asianoption.dta", replace
local N=25000
quietly {
    forvalues i=1/`N' {
      #delimit ; //Initialise variables
      drop _all; set obs 125; local obs= 125;
      local K = 6500; local iv = 0.2652; // Simulation 1
      //local K = 5500; local iv = 0.3433; // Simulation 2
      local s0 = 6289.70; local fut = 6405.35;
      local rf = 0.0624; local dy = 0.0242;
      #delimit cr
      local ttm = 0.5
      local dt = `ttm'/`obs'
      local drift = (`rf'-`dy'-`iv'^2/2)*`dt'
      local vsqrdt = `iv'*`dt'^0.5
      // Generate a random path for the spot price
      generate rands = rnormal()
      generate spot = `s0'*exp(`drift'+`vsqrdt'*rands) in 1
      replace spot = spot[_n-1]*exp(`drift'+`vsqrdt'*rands) in 2/`obs'
      //Compute and save call and put price w.r.t. path
      summarize spot, meanonly
      scalar av = r(mean)
      scalar call_val = max(av-`K',0)*exp(-`rf'*`ttm')
      scalar put_val = max(`K'-av,0)*exp(-`rf'*`ttm')
      post `prices' (put_val) (call_val)
      scalar drop av call_val put_val
      // Reuse random number with opposite sign
      replace rands = -rands
      replace spot = `s0'*exp(`drift'+`vsqrdt'*rands) in 1
      replace spot = spot[_n-1]*exp(`drift'+`vsqrdt'*rands) in 2/`obs'
      //Compute and save call and put price w.r.t. path
      summarize spot, meanonly
      scalar av = r(mean)
      scalar call_val = max(av-`K',0)*exp(-`rf'*`ttm')
      scalar put_val = max(`K'-av,0)*exp(-`rf'*`ttm')
      post `prices' (put_val) (call_val)
    }
  }
postclose `prices'
use "D:\Programming Guide\Stata Guide\asianoption.dta", clear
describe
summarize call_val
summarize put_val
```

Many parts of the program above use identical instructions to those given for the DF critical value simulation, and so annotation will now focus on the construction of the program and on previously unseen commands. As with the do-file for the DF critical value simulation, you can open (and run) the program to price Asian options by opening the *Do-file Editor* in Stata and selecting the do-file **'option_pricing_simulation.do'**. You should then be able to inspect the set of commands and identify

commands, comments and other operational variables based on the colouring system described in the previous subsection.

In line 5 you can see that this program is going to create a new workfile **'asianoption.dta'** that will contain the 50,000 simulated values for the put options (**put_val**) and the call options (**call_val**). The following lines define how these values are generated.

Line 6 indicates that we will be performing **N=25,000** repetitions of the set of commands (i.e., simulations). However, we will still generate 50,000 values for the put and call options each by using a 'trick', i.e., using the antithetic variates. This will be explained in more detail once we get to this command. Overall, each simulation will be performed for a set of 125 observations (see line 10). The following lines specify the parameters for the simulation of the path of the underlying asset: the time to maturity (**ttm**), the implied volatility (**iv**), the risk-free rate (**rf**), and dividend yield (**dy**). We used the command **#delimit ;** to change the delimiter to a semicolon. This enables us to write several lines into one to save space. In line 13, we change the delimiter back to carriage return.

The command **'local'** in front of the terms tells Stata that it constructs these variables as sort of operational or auxiliary variables that represent the predefined content. '**local dt='ttm'/'obs'**' in line 17 splits the time to maturity (0.5 years) into 125 discrete time periods. Since daily averaging is required, it is easiest to set 'obs' to 125 (the approximate number of trading days in half a year), so that each time period '**dt**' represents one day. The model assumes under a risk-neutral measure that the underlying asset price follows a geometric Brownian motion, which is given by

$$dS = (\text{rf} - \text{dy})Sdt + \sigma dz, \tag{13}$$

where dz is the increment of a Brownian motion. The discrete time approximation to this for a time step of one can be written as

$$S_t = S_{t-1} \exp\left[\left(\text{rf} - \text{dy} - \frac{1}{2}\sigma^2\right)dt + \sigma\sqrt{dt}u_t\right] \tag{14}$$

where u_t is a white noise error process. The following lines define further characteristics of the options such as the strike price (**K=6500**) and the initial price of the underlying asset at $t = 0$ (**s0=6289.70**).[75]

Lines 21 to 23 generate the path of the underlying asset. First, random $\mathcal{N}(0, 1)$ draws are made (line 21), which are then constructed into a series of future prices of the underlying asset for the next 125 observations. Once all 125 observations have been created, we summarize the mean value of this series (i.e., the average price of the underlying over the lifetime of the option) which we capture in **'scalar av=r(mean)'** (lines 26).

Lines 27 and 28 construct the terminal payoffs for the call and the put options, respectively. For the call, **call_val** is set to the average underlying price less the strike price if the average is greater than the strike (i.e., if the option expires in the money), and zero otherwise. Vice versa for the put. The payoff at expiry is discounted back to the present based on the risk-free rate (using the expression **exp(-'rf'*'ttm')**). These two values, i.e., the present value of the call and put options, are then posted to the new workfile (line 29).

The process then repeats using the antithetic variates, constructed using **replace rands=-rands**. As can be seen, we simply replace the values in **rands** and update the spot price series based on the antithetic variates. Again, we calculate present values of the call and put options for these alternative paths of the underlying asset and send them to the new workfile. Note that this way we can double the 'simulated' values for the put and call options without having to draw further random variates.

This completes one cycle of the loop, which will then be repeated for further 24,999 times and overall create 50,000 values for the call and put options each. Once the loop has finished, we can open the new

[75] Note that the values given here correspond to Simulation 1. For the second simulation the values have to be altered accordingly.

workfile **'asianoption.dta'** which contains the 50,000 simulated values using the command in line 44. We can also look at the results using the *Data Editor*, and we will see that the dataset contains two arrays **'call_val'** and **'put_val'**, which comprise 50,000 rows of the present value of the call and put option for each simulated path. The last thing that we then need to do is calculate the option prices as the averages over the 50,000 replications. This can be done by reporting summary statistics in lines 46 and 47. The following output should appear in the *Output window*.

```
. summarize call_val

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
    call_val |     50,000    206.3041    378.6017         0   3337.203

. summarize put_val

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
     put_val |     50,000    350.6863    414.5153         0   2330.125

.
```

For the specifics stated above, the call price is approximately 206.30 and the put price lies around 350.69. Note that both call values and put values can be calculated easily from a given simulation, since the most computationally expensive step is in deriving the path of simulated prices for the underlying asset. In the following table, we compare the simulated call and put prices for different implied volatilities and strike prices along with the values derived from an analytical approximation to the option price, derived by Levy, and estimated using VBA code in Haug (1998), pp. 97–100.

Table 4: Simulated Asian Option Prices

Simulation 1: Strike = 6500, IV = 26.52%		**Simulation 2:** Strike = 5500, IV = 34.33%	
Call	Price	**Call**	Price
Analytical Approximation	203.45	Analytical Approximation	888.55
Monte Carlo Normal	206.30	Monte Carlo Normal	887.95
Put	Price	**Put**	Price
Analytical Approximation	348.70	Analytical Approximation	64.52
Monte Carlo Normal	350.69	Monte Carlo Normal	62.78

The main difference between the way that the simulation is conducted here and the method used for Stata simulation of the Dickey–Fuller critical values is that here, the random numbers are generated by opening a new series called **'rands'** and filling it with the random number draws. The reason that this must be done is so that the negatives of the elements of rands can later be taken to form the antithetic variates. Then the call and put prices for each replication are discounted back to the present using the risk-free rate, and outside of the replications loop, the options prices are the averages of these discounted prices across the 50,000 replications.

In both cases, the simulated option prices are quite close to the analytical approximations, although the Monte Carlo seems to overvalue the out-of-the-money call and to undervalue the out-of-the-money put. Some of the errors in the simulated prices relative to the analytical approximation may result from the use of a discrete-time averaging process using only 125 data points.

24 Value at Risk

24.1 Extreme Value Theory

Reading: Brooks (2019, Section 14.3)

In this section, we are interested in extreme and rare events such as stock price crashes. Therefore, we will look into the left tail of the distribution of returns. We will use the dataset provided in **'sp500.xlsx'** which contains daily returns on the S&P500 index from January 1950 until July 2018. From the price series we generate log returns by typing

generate ret = log(sp500) – log(sp500[_n-1])

This can be done without setting a time variable via `tsset` because '[_n-1] refers to the previous observation. To have a first glance at the data, we type:

summarize ret, detail

to obtain a more detailed summary statistics such as that below.

```
. summarize ret, detail

                            ret
-------------------------------------------------------------
      Percentiles      Smallest
 1%    -.0259661      -.2289973
 5%    -.0144386      -.0946951
10%     -.009887      -.0935365       Obs              17,246
25%    -.0040379      -.0921896       Sum of Wgt.      17,246

50%     .0004708                      Mean           .0002975
                       Largest       Std. Dev.      .0096484
75%     .0049492       .0683664
90%     .0101327       .0870888       Variance       .0000931
95%     .0143454       .1024574       Skewness      -1.017048
99%     .0251162        .109572       Kurtosis       30.17506

.
```

As we are interested in the lower tail, the first three numbers of the first column, the 1%, 5% and 10% percentiles, are of interest. They tell us, assuming the historical distribution of returns, what is the maximum loss to expect, if we exclude the most extreme 1,5 or 10% of the distribution. In finance, we also call them Value at Risk (VaR), e.g., a 1%-VaR of –0.026 as in our case means that the maximum loss for a day that we can expect equals 2.6%, based on the 99% largest returns. Since this estimate is derived directly from the historical data, we will also refer to it as the historical VaR.

If we were to assume that the returns follow a normal distribution, we can compute the 1%–VaR directly from the mean (μ) and standard deviation (σ). The α-percentile of a normal distribution is then given as

$$VaR^{\alpha}_{\text{normal}} = \mu - \Phi^{-1}(1-\alpha)\sigma \tag{15}$$

Since the last command we ran was `summarize`, Stata stores the values and we can access them using the function `r()`. Therefore we can compute the $VaR^{\alpha}_{\text{normal}}$ directly after the summary statistics by typing

display r(mean) – invnormal(1-0.01)*r(sd),

which will display the 1%-VaR as below.

```
. display r(mean) - invnormal(1-0.01)*r(sd)
-.02214817

.
```

We directly see that the value $VaR^{1\%}_{\text{normal}} = -0.022$ is smaller in absolute value than the one obtained from the historical distribution. Hence, assuming a normal distribution would lead to the assumption that the maximal loss to expect is only 2.2%.

24.2 The Hill Estimator for Extreme Value Distributions

In the next step, we want to look into the tail of the distribution to estimate the VaR and we will implement the Hill estimator of the shape parameter ξ of an extreme value distribution. The necessary code can be found in the file **'EVT.do'** as presented below.

```
1  /* Hill estimator for the shape parameter of an extreme value distribution */
2  clear
3  import excel "D:\Programming Guide\data\sp500.xlsx", sheet("sp500") firstrow
4  generate ret = log(sp500[_n]) - log(sp500[_n-1])
5  summarize ret
6  scalar alpha = 0.01 //percentile
7  scalar N = r(N)
8  scalar U = -0.025 //threshold
9  generate y = -ret if (ret<U)
10 summarize y
11 local N_U = r(N)
12 sort y
13 tempname hill
14 postfile `hill' xi var  using "D:\Programming Guide\Stata Guide\hill.dta", replace
15 forvalues k=1/`N_U' {
16   replace s = log(y) - log(y[`N_U'-`k'+1])
17   summarize s if _n >`N_U'-`k'
18   scalar xi = 1/(`k'-1)*r(sum) //Hill estimator
19   scalar var = -y[`k']*(N*alpha/`N_U')^(-xi) //VaR estimation
20   post `hill' (xi) (var)
21 }
22 postclose `hill'
```

In the first lines, we import the dataset, compute the returns and define the confidence level **alpha** and the threshold **U**. In line 9, we generate the data series $\tilde{y}$, which comprises the absolute value of all returns below the threshold U. We also save the length of this dataset in the variable `N_U` (line 11). For the Hill estimation we need to sort $\tilde{y}$, which we achieve in line 12.

As the Hill estimator is dependent on how many of the observations below the threshold are used (k), we will create a list and estimate ξ_k for all possible values of k. The code for this procedure spans from line 13 to 22, while it is specifically line 18 where we compute the Hill estimator as in Brooks (2019):

$$\hat{\xi}_k = \frac{1}{k-1}\sum_{i=1}^{k-1} \log(\tilde{y}_{(i)}) - \log(\tilde{y}_{(k)}) \tag{16}$$

We compute Equation (16) in three steps. First, we calculate the log differences of $\tilde{y}$ to the k^{th} biggest value $\tilde{y}_{(k)}$ and save it into the variable **s** (line 16). Second, we summarize this variable over those indices

that exceed $N_U - k$ (line 17). Finally, we compute the VaR following Brooks (2019):

$$\text{VaR}_{hill,k} = \tilde{y}_{(k)} \left[\frac{N\alpha}{N_U} \right]^{-\hat{\xi}_k}, \tag{17}$$

where we replace $\tilde{y}_{(k)}$ with its negative value to switch the sign back (line 19).

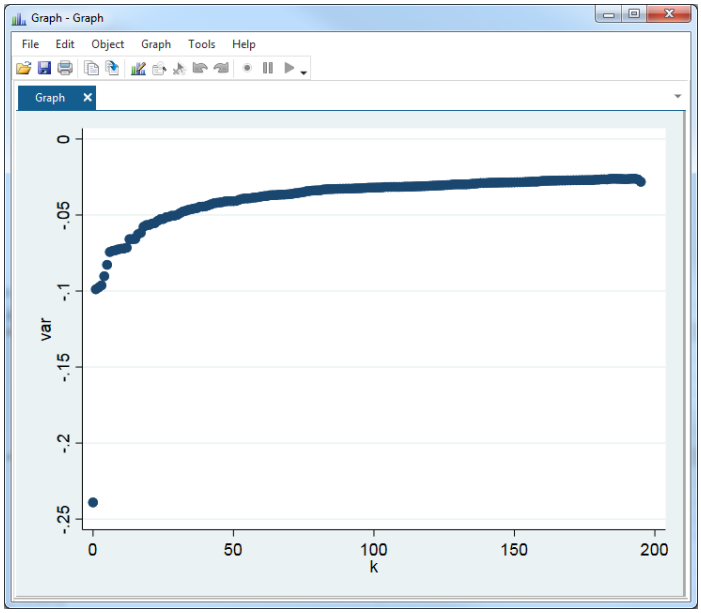


Figure 117: Hill Plot for Value at Risk

If we open the new file **'hill.dta'**, we can plot the series of VaR estimates against the number of extreme values k included. Looking at the Hill plot in Figure 117, we can see that the estimated VaR is much higher in absolute value. Using all observations we obtain $VaR_{hill} = -0.032$, which means that the expected maximum is 3.2%.

24.3 VaR Estimation Using Bootstrapping

Reading: Brooks (2019, Section 13.9)

The following Stata code can be used to calculate the minimum capital risk requirement (MCRR) for a ten-day holding period (the length that regulators require banks to employ) using daily S&P500 data, which is found in the file **'sp500.dta'**. The code is presented on the following page followed by comments on some of the key lines.

Again, annotation of the Stata code will concentrate on commands that have not been discussed previously. The first lines of command, set up the dataset for the further analyses. As we will be changing the dataset throughout the program, we type the command 'preserve' in line 13 which saves the dataset in its current stage and guarantees that it will be restored to this stage once the program has ended. Note that when starting your program with the command 'preserve' it even restores the dataset to its initial state when the program is not executed entirely, e.g., due to a bug in the program or due to pressing **'break'**.

```
1 /*Calculation of MCRR for  a 10-day horizon using bootstrapping*/
2 clear
3 import excel "D:\Programming Guide\data\sp500.xlsx", sheet("sp500") firstrow
```

```
generate n=_n
tsset n
local T = _N //17247
generate rt=log(sp500/L.sp500)
arch rt in 2/`T', arch(1/1) garch(1/1) nolog
quietly {
predict h in 2/`T', variance
generate resid = rt-_b[_cons]
generate sres=resid/h^0.5
preserve
set seed 12345
tempfile sp500_using bootstrapped
tempname mcrr
postfile `mcrr' min max using "D:\Programming Guide\Stata Guide\VaR.dta", replace
  forvalues i=1/100 {
    save "`sp500_using'", replace
    keep sres
    rename sres sres_b
    bsample in 2/`T'
    save "`bootstrapped'", replace
    use "`sp500_using'", clear
    merge 1:1 _n using "`bootstrapped'"
    set obs 17257

    scalar a0 = [ARCH]_b[_cons]
    scalar a1 = [ARCH]_b[L.arch]
    scalar b1 = [ARCH]_b[L.garch]

    replace h=a0+a1*resid[`T']^2+b1*h[`T'] in 17248
    replace rt=_b[_cons]+sqrt(h)*sres_b[1] in 17248
    replace sp500 = sp500[`T']*exp(rt) in 17248

    forvalues i=17249/17257 {
      replace h = a0+(a1+b1)*h[`i'-1] in `i'
      replace rt=_b[_cons]+sqrt(h)*sres_b[`i'-`T'] in `i'
      replace sp500 = sp500[`i'-1]*exp(rt) in `i'
    }

    summarize sp500 if _n>`T'
    post `mcrr' (r(min)) (r(max))
    drop in 17248/17257
    drop  sres_b _merge
  }
  postclose `mcrr'
  use "D:\Programming Guide\Stata Guide\VaR.dta", clear
  //Long and Short positions
  generate l1=log(min/2815.6201)
  summarize l1
  scalar mcrrl=1-exp((-1.645*r(sd))+r(mean))
  generate s1=log(max/2815.6201)
  summarize s1
  scalar mcrrs=exp((1.645*r(sd))+r(mean))-1
}
display mcrrl
display mcrrs
```

We also create a new variable **`_n'** as a substitute for the missing time variable in the dataset and **`tsset'** the data to this variable. This is necessary as some of the commands that we will use later only work

when 'tsset' is performed.

The command **generate rt=log(sp500/L.sp500)** in line 7 creates continuously compounded returns which are the dependent variable in the GARCH(1,1) model, i.e., one ARCH and one GARCH term, that is estimated in the following line. As we lose one data point when calculating returns we only perform the estimation for observations 2 to T.

The mean equation of the model contains a constant only, which is automatically included by Stata, so we only need to specify the dependent variable **rt** in the regression command. The following line generates a series of the fitted conditional variances for the model which are stored in the series **h**. The two lines **generate resid=rt-_b[_cons]** and **generate sres=resid/h^0.5** will construct a set of standardised residuals.

Next, we set up a set of (temporary) files that we will use within the bootstrap loop to temporarily store subsets of the data as well as to store our final estimates from the replication. In particular, the command **tempfile sp500_using bootstrapped** in line 15 creates two auxiliary files that will be automatically deleted after the program has ended and serve to temporarily store a subset of the data. The **postfile** command should be well known from the previous simulation examples and sets up the file **'VaR.dta'** to store the results of the bootstrap loop.

Next follows the core of the program, which involves the bootstrap loop. The number of replications has been defined as 10,000. What we want to achieve is to re-sample the series of standardised residuals (**'sres'**) by drawing randomly with replacement from the existing dataset. However, as only the **'sres'** series should be re-sampled while all the other data series should remain in their current order, we need to transfer the 'sres' data series to the temporary file 'bootstrapped', perform the re-sampling and re-import the re-sampled series into the full dataset. This is achieved by the set of commands in lines 19 to 25. Note that the command **bsample** is the command performing the actual bootstrapping while the other results are merely auxiliary commands to store and merge the bootstrapped dataset. In order to distinguish the bootstrapped standardised residuals from the original series, we rename the series to **'sres_b'** before merging it with the full dataset.

The next block of commands constructs the future path of the S&P500 return and price series as well as the conditional variance over the ten-day holding period. The first step to achieve this is to extend the sample period to include 10 further observations, i.e., observations 17248 to 17257. Unfortunately, Stata does not allow us to simply extend some of the series to these new observations so we need to manually tell Stata which values to input in the new cells. As these series are all recursively constructed we start with the one-step ahead forecasts (observation 17248) in line 32 to 34 and then assign the values for the other observations in a loop.

The one-step-ahead forecast of the conditional variance (See Brooks (2019), Chapter 9) is

$$h^f_{1,T} = \alpha_0 + \alpha_1 u^2_T + \beta h_T,$$

where h_T is the conditional variance at time T, i.e., the end of the in-sample period, and u_T is the squared disturbance term at time T, and α_0, α_1 and β are the coefficient estimates obtained from the GARCH(1,1) model estimated over the observations 2 to T and the s-step-ahead forecast can be produced by iterating

$$h^f_{s,T} = \alpha_0 + (\alpha_1 + \beta) h^f_{s-1,T} \qquad s \geq 2.$$

First we create a set of scalars that contain the estimated coefficients (**a0, a1** and **b1**). Then, we generate the one-step-ahead forecast of h according to the formula above and place the value in observation 17248 of the series 'h'. Using this we can write the one-step ahead forecast of the return as **_b[_cons] + sqrt(h)*sres_b[1]**. With the forecasted return we can also determine the new price as the previous price compounded with the forecasted return in line 34.

Once we have determined the one-step ahead forecasts, the next steps are obtained recursively. Therefore we use the loop in line 36 to 40 and consecutively update the variance, return and price series.[76]

Finally, we obtain the minimum and maximum values of the S&P500 series over the ten days and post them into the **'mcrr'** file.

This set of commands is then repeated 10,000 times so that after the final repetition there will be 10,000 minimum and maximum values for the S&P500 prices in **'VaR.dta'**.

The final block of commands generates the MCRR for a long and a short position in the S&P500. The first step is to construct the log returns for the maximum loss over the ten-day holding period, which for the long position is achieved by the command **generate l1=log(min/2815.6201)** and for the short position by **generate s1=log(max/2815.6201)**. We now want to find the 5th percentile of the empirical distribution of the maximum losses for the long and short positions. Under the assumption that the 'l1' and 's1' statistics are normally distributed across the replications, the MCRR can be calculated by the commands **scalar mcrrl=1-exp((-1.645*r(sd))+r(mean))** and **scalar mcrrs=exp((1.645*r(sd))+r(mean))-1**, respectively. The results generated by running the above program should be displayed in the Stata *Output* window and should approximate to:

`mcrrl` = 0.03071088 for the long position, and

`mcrrs` = 0.03638716 for the short position.

These figures represent the MCRR for a long and short position, respectively, as a percentage of the initial value of the position for 95% coverage over a 10-day horizon. This means that, for example, approximately 3.1% of the value of a long position held as liquid capital will be sufficient to cover losses on 95% of days if the position is held for 10 days. The required capital to cover 95% of losses over a 10-day holding period for a short position in the S&P500 index would be around 3.6%. This is as one would expect since the index had a positive drift over the sample period. Therefore, the index returns are not symmetric about zero as positive returns are slightly more likely than negative returns. Higher capital requirements are thus necessary for a short position since a loss is more likely than for a long position of the same magnitude.

[76] Note that the order of the commands is important as the new assigned conditional variance **h** in line 37 is used in line 38 to update the return series, which then is used to update the index level in line 39.

25 The Fama–MacBeth Procedure

Reading: Brooks (2019, Section 14.2)

In this section we will perform the two-stage procedure developed by Fama and MacBeth (1973). The Fama–MacBeth procedure as well as related asset pricing tests are described Chapter 14 of Brooks (2019). There is nothing particularly complex about the two-stage procedure – it only involves two sets of standard linear regressions. The hard part is really in collecting and organising the data. If we wished to do a more sophisticated study – for example, using a bootstrapping procedure or using the Shanken (1992) correction. This would require more analysis then is conducted in the illustration below. However, hopefully the Stata code and the explanations will be sufficient to demonstrate how to apply the procedures to any set of data.

The example employed here is taken from the study by Gregory et al. (2013) that examines the performance of several different variants of the Fama–French and Carhart models using the Fama–MacBeth methodology in the UK following several earlier studies showing that these approaches appear to work far less well for the UK than the US. The data required are provided by Gregory et al. (2013) on their web site.[77] Note that their data have been refined and further cleaned since their paper was written (i.e., the web site data are not identical to those used in the paper) and as a result the parameter estimates presented here deviate slightly from theirs. However, given that the motivation for this exercise is to demonstrate how the Fama–MacBeth approach can be used in Stata, this difference should not be consequential. The two data files used are **'monthlyfactors.dta'** and **'vw_sizebm_25groups.dta'**. The former file includes the time-series of returns on all of the factors (smb, hml, umd, rmrf, the return on the market portfolio (rm) and the return on the risk-free asset (rf)), while the latter includes the time-series of returns on 25 value-weighted portfolios formed from a large universe of stocks, two-way sorted according to their sizes and book-to-market ratios.

The first step in this analysis for conducting the Fama–French or Carhart procedures using the methodology developed by Fama and MacBeth (1973) is to create a new Stata workfile which we call **'ff_data.dta'** and import the two .dta files into it. The data in both cases run from October 1980 to December 2017, making a total of 447 data points. However, in order to obtain results as close as possible to those of the original paper, when running the regressions, the period is from October 1980 to December 2010 (363 data points). We then need to set up a do-file (**fama_french.do**) along the lines of those set up in the previous subsections to conduct the two-stage procedure. The code is provided on the next page.

Before starting with the actual two-stage Fama–MacBeth procedure we save a snapshot of the current state of the workfile (**'snapshot save'**, line 4). This is useful as we will be performing transformations of the data in the file that are irreversible (once performed). If a snapshot of the file is saved than we can always return to the file at the state at which the snapshot was saved, undoing all the data manipulations that were undertaken in the meantime.

We can think of the remainder of this program as comprising several sections. The first step is to transform all of the raw portfolio returns into excess returns which are required to compute the betas in the first stage of Fama–MacBeth (lines 5 to 7). This is fairly simple to do and we just write over the original series with their excess return counterparts. The command **'local obs=363'** in line 9 ensures that the same sample period as the paper by Gregory et al. (2013) is employed throughout.

Next, we run the first stage of the Fama–MacBeth procedure, i.e., we run a set of time-series regressions to estimate the betas. We want to run the Carhart (1997) 4-factor model separately for each of the twenty-five portfolios. A Carhart 4-factor model regresses a portfolio return on the excess market return (**'rmrf'**), the size factor (**'smb'**), the value factor (**'hml'**) and the momentum factor (**'umd'**).

[77] `http://business-school.exeter.ac.uk/research/centres/xfi/famafrench/files/`

```
/* The Fama-MacBeth two stage procedure */
clear
use "D:\Programming Guide\Stata Guide\stata_files\ff_data.dta"
snapshot save
foreach var of varlist S* M* B* {
  replace `var'=`var'-rf //Transform actual returns into excess returns
}
// FIRST STAGE TIME-SERIES REGRESSIONS ////////////////////////////////////////
local obs=363 // set to 447 for whole dataset
tempname betas
postfile `betas' beta_c beta_rmrf beta_umd beta_hml beta_smb using "D:\Programming
    Guide\Stata Guide\betas.dta", replace
foreach y of varlist S* M* B* {
  regress `y' rmrf umd hml smb if month>=tm(1980m10) & month<=tm(2010m12)
  scalar beta_c=_b[_cons]
  scalar beta_rmrf=_b[rmrf]
  scalar beta_umd=_b[umd]
  scalar beta_hml=_b[hml]
  scalar beta_smb=_b[smb]
  post `betas' (beta_c) (beta_rmrf) (beta_umd) (beta_hml) (beta_smb)
}
postclose `betas'
drop smb hml umd rf rm rmrf
xpose, clear varname // Transpose data to cross sectional
drop in 1
merge 1:1 _n using "D:\Programming Guide\Stata Guide\betas.dta"
// SECOND STAGE CROSS-SECTIONAL REGRESSIONS ///////////////////////////////////
local obs=363  // set to 447 for whole dataset
tempname lambdas
postfile `lambdas' lambda_c lambda_rmrf lambda_umd lambda_hml lambda_smb rsquared
    using "D:\Programming Guide\Stata Guide\lambdas.dta", replace
forvalues i=1(1)`obs' {
  regress v`i' beta_rmrf beta_umd beta_hml beta_smb
  scalar lambda_c=_b[_cons]
  scalar lambda_rmrf=_b[beta_rmrf]
  scalar lambda_umd=_b[beta_umd]
  scalar lambda_hml=_b[beta_hml]
  scalar lambda_smb=_b[beta_smb]
  scalar rsquared=e(r2)
  post `lambdas' (lambda_c) (lambda_rmrf) (lambda_umd) (lambda_hml) (lambda_smb) (
      rsquared)
}
postclose `lambdas'
snapshot restore 1
use "D:\Programming Guide\Stata Guide\lambdas.dta", clear
local obs=363
foreach var of varlist lambda* { // Calculate lambda estimates and t-ratios
  summarize `var'
  scalar `var'_mean=r(mean)
  scalar `var'_tratio=sqrt(`obs')*r(mean)/r(sd)
}
summarize rsquared
display lambda_c_mean lambda_c_tratio
display lambda_rmrf_mean lambda_rmrf_tratio
display lambda_smb_mean lambda_smb_tratio
display lambda_hml_mean lambda_hml_tratio
display lambda_umd_mean lambda_umd_tratio
```

Since the independent variables remain the same across the set of regressions and we only change the dependent variable, i.e., the excess return of one of the 25 portfolios, we can set this first stage up as a loop. Lines 12 to 20 specify this loop. In particular, the lines of command specify that for each of the variables listed behind the term **'varlist'**, Stata should perform an OLS regression of the portfolio's excess return (indicated by the **'y'** which is to be replaced by the particular portfolio return listed in the loop) on the four factors. Since the portfolio return variables all start with S, M or B, we can write **S* M* B*** to tell Stata to use all variables that start with these letters. Note that we restrict the sample for this model to the period October 1980 to December 2012 (**if month>=tm(1980m10) & month<=tm(2010m12)**) instead of using the whole sample.

We need to store the estimates from these regressions into separate series for each parameter. This is achieved by the commands in lines 10 and 11 as well as lines 19 and 21. In particular, the command **'postfile'** sets up a new file named 'betas.dta' in which the beta coefficients on the four factors plus the constant are saved. Thus, for each of the twenty-five portfolios we will generate four beta estimates that will then be posted to the file **'betas.dta'**. After the loop has ended, the file **'betas.dta'** will contain 25 observations for the 6 variables **'beta_c', 'beta_rmrf', 'beta_umd', 'beta_hml'** and **'beta_smb'**.

To illustrate this further, the loop starts off with the excess return on portfolio 'SL' (the first item that starts with S) and the coefficient estimates from the 4-factor model (**'beta_c' 'beta_rmrf' 'beta_umd' 'beta_hml' 'beta_smb'**) will be transferred to the **'betas.dta'** file as the first entry for each variable.

Then Stata moves on to the second variable in the **'varlist'**, i.e., the excess return on portfolio **'S2'**, and will perform the 4-factor model and save the coefficient estimates in **'betas.dta'**. They will constitute the second entry in the beta series. Stata will continue with this procedure until it has performed the set of commands for the last variable that start with B, i.e., **'BH'**. The beta coefficients for the regressions using **'BH'** as the dependent variable will be the 25th entry in the file **'betas.dta'**.

Having run the first step of the Fama–MacBeth methodology, we have estimated the betas, also known as the factor exposures. The slope parameter estimates for the regression of a given portfolio (i.e., **'beta_rmrf', 'beta_umd', 'beta_hml', 'beta_smb'**) will show how sensitive the returns on that portfolio are to the corresponding factors and the intercepts ('betas_c') will be the Jensen's alpha estimates.

These intercept estimates stored in **'beta_c'** should be comparable to those in the second panel of Table 6 in Gregory et al. (2013) – their column headed '*Simple 4F*'. Since the parameter estimates in all of their tables are expressed as percentages, we need to multiply all of the figures given from the Stata output by 100 for them to have the same scale.

If the 4-factor model is a good one, we should find that all of these alphas are statistically insignificant. We could test this individually, if we wished, by adding an additional line of code in the loop to save the t-ratio in the regressions.[78]. However, we avoid this here, since we will be working with the β estimates only.

The second stage of the Fama–MacBeth procedure is to run a separate cross-sectional regression for each point in time. An easy way to do this is to, effectively, rearrange or transpose the data so that each column represents one month and the rows are the excess returns of one of the 25 portfolios. In Stata this can be easily achieved by the command `xpose, clear` (line 23). '**xpose**' transposes the data, changing variables into observations and observations into variables. The supplement **'varname'** adds the variable **'_varname'** to the transposed data containing the original variable names. The command **'xpose'** deletes untransposed data so that after the execution the dataset cannot be returned to its previous stage. To prevent our original data from getting lost, we saved the state of the data prior to any transformations. For the second stage, we will not be using the time-series factors and variables

[78] An appropriate command would be **scalar beta_c_tratio=_b[_cons]/_se[_cons]**. Note that you would also need to create the variable as part of the file and store it afterwards.

'smb', 'hml', 'umd', 'rf', 'rm' and **'rmrf'** so we drop these from the dataset prior to transposing the data (line 22). Then we transpose the data in line 23. Note that Stata has allocated new variable names to the observations, which are constructed as **'v'** plus the position of the observation in the previous dataset. For example, all data points in the first row (of the previous dataset) corresponding to October 1980 are now stored in **'v1'**; all data points in the second row corresponding to November 1980 are now stored in **'v2'**; and so on. The first row of the transposed dataset contains the months (though in numeric coding). We drop this observation to make each row correspond to one of the 25 portfolios (line 24). Thus, the first column contains the excess returns of all 25 portfolios for October 1980 and the last column contains the excess returns of the 25 portfolios for December 2017.

Finally, we merge the transformed dataset with the betas estimated at the first stage of Fama–MacBeth (line 25), which basically involves adding these variables to the existing dataset. This is necessary as these variables will serve as the independent variables in the second stage of Fama–MacBeth. We are now in a position to run the second-stage cross-sectional regressions corresponding to the following equation.

$$\bar{R}_i = \alpha + \lambda_M \beta_{i,M} + \lambda_S \beta_{i,S} + \lambda_V \beta_{i,V} + \lambda_U \beta_{i,U} + e_i \tag{18}$$

This is performed in lines 30 to 39. Again, it is more efficient to run this set of regressions in a loop. In particular, we regress each of the variables (i.e., **'v1', 'v2', 'v3',. . . ,'v363'**) representing the excess returns of the 25 portfolios for a particular month on the beta estimates from the first stage (i.e., **'beta_rmrf', 'beta_umd', 'beta_hml'** and **'beta_smb'**). Then we store the coefficients on these factor exposures in the scalars **'lambda_c', 'lambda_rmrf', 'lambda_umd', 'lambda_hml'** and **'lambda_smb'**.

'lambda_c' will contain all intercepts from the second stage regressions, **'lambda_rmrf'** will contain all parameter estimates on the market risk premium betas, and so on. We also collect the R^2 values for each regression in the scalar **'rsquared'** as it is of interest to examine the cross-sectional average. We store these lambdas in the file **'lambdas.dta'**.

Note that the command **'forvalues i=1(1)'obs"** in line 30 indicates that we restrict the sample to December 2010 to make it comparable to Gregory et al. (2013). Thus, the first regression will be of **'v1'** corresponding to October 1980 on a constant, **beta_rmrf, beta_umd, beta_hml** and **beta_smb** with the estimates being stored in the first row of the variables **'lambda_c', 'lambda_rmrf', 'lambda_umd', 'lambda_hml', 'lambda_smb'** and **'rsquared'** in the file **'lambdas.dta'**. The second regression will be of **'v2'** corresponding to November 1980 on the set of betas and the estimates will be stored in the second row in **'lambdas.dta'**. This will go on until **'v363'** corresponding to December 2010 and the lambda estimates for this regression will be reported in row 363 of **'lambdas.dta'**. Finally, we restore the dataset to its initial state with the command 'snapshot restore 1' (line 43). The estimates saved in the files **'betas.dta'** and **'lambdas.dta'** will not become lost when executing this command.

The final stage of the Fama–MacBeth procedure is to compute the averages, standard deviations and t-ratios from the series of estimates from the second stage. For every factor j we compute the time-series averages and standard deviations of the estimates $\hat{\lambda}_{j,t}$ as follows:

$$\hat{\lambda}_j = \frac{1}{T}\sum_{t=1}^{T} \hat{\lambda}_{j,t}, \quad \hat{\sigma}_j = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{\lambda}_{j,t} - \hat{\lambda}_j)^2} \quad \text{and} \quad t_{\lambda_j} = \sqrt{T}\,\frac{\hat{\lambda}_j}{\hat{\sigma}_j} \tag{19}$$

To do this we first **'summarize'** each of the variables and then create a scalar containing the mean of the variable ('**scalar 'var'_mean=r(mean)**') and a scalar containing the t-ratio of the estimate ('**scalar 'var'_tratio=sqrt('obs')*r(mean)/r(sd)**'). Thus **'lambda_c_mean'** will contain the mean of the

cross-sectional intercept estimates, and the corresponding t-ratio will be stored in **'lambda_c_tratio'** and so on. Note that we loop this step in order to preserve space.

Finally, we tell Stata to display these values (lines 50 to 54) in order to inspect the results of the Fama–MacBeth procedure. The lambda parameter estimates should be comparable with the results in the columns headed '*Simple 4F Single*' from Panel A of Table 9 in Gregory et al. (2013). Note that they use γ to denote the parameters which have been called λ in this text. The parameter estimates obtained from this simulation and their corresponding t-ratio are given in the table below. Note that the latter do not use the Shanken (1992) correction as Gregory et al. (2013) do. These parameter estimates are the prices of risk for each of the factors, and interestingly only the price of risk for value is significantly different from zero.

Table 5: Fama–MacBeth Market Prices of Risk

Parameter	Estimate	t-ratio
λ_C	0.50	1.33
λ_{RMRF}	0.07	0.15
λ_{SMB}	0.09	0.51
λ_{HML}	0.39	2.08
λ_{UMD}	0.41	0.78
Mean R^2		

References

Brooks, C. (2019). *Introductory Econometrics for Finance*. Cambridge University Press, Cambridge, UK, 4th edition.

Carhart, M. M. (1997). On persistence in mutual fund performance. *Journal of Finance*, 52(1):57–82.

Durbin, J. and Watson, G. S. (1951). Testing for serial correlation in least squares regression. ii. *Biometrika*, 38(1/2):159–177.

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007.

Fama, E. F. and MacBeth, J. D. (1973). Risk, return, and equilibrium: Empirical tests. *Journal of Political Economy*, 81(3):607–636.

Fuller, W. A. (1976). *Introduction to Statistical Time Series*, volume 428. John Wiley and Sons.

Gregory, A., Tharyan, R., and Christidis, A. (2013). Constructing and testing alternative versions of the fama–french and carhart models in the uk. *Journal of Business Finance and Accounting*, 40(1-2):172–214.

Haug, E. G. (1998). *The Complete Guide to Option Pricing Formulas*. McGraw-Hill New York.

Heslop, S. and Varotto, S. (2007). Admissions of international graduate students: Art or science? a business school experience. *ICMA Centre Discussion Papers in Finance*, 8.

Maddala, G. S. and Wu, S. (1999). A comparative study of unit root tests with panel data and a new simple test. *Oxford Bulletin of Economics and statistics*, 61(S1):631–652.

Nelson, D. B. (1991). Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, 59(2):347–370.

Pedroni, P. (1999). Critical values for cointegration tests in heterogeneous panels with multiple regressors. *Oxford Bulletin of Economics and Statistics*, 61:653–670.

Pedroni, P. (2001). Purchasing power parity tests in cointegrated panels. *Review of Economics and Statistics*, 83(4):727–731.

Persyn, D. and Westerlund, J. (2008). Error-correction-based cointegration tests for panel data. *Stata Journal*, 8(2):232–241.

Shanken, J. (1992). On the estimation of beta-pricing models. *Review of Financial Studies*, 5(1):1–33.

Westerlund, J. (2007). Testing for error correction in panel data. *Oxford Bulletin of Economics and Statistics*, 69(6):709–748.