Practical Empirical Research Using Gretl and Hansl

Artur Tarassow*
Department of Socioeconomics, University of Hamburg

August 29, 2018

Abstract

This article provides an introduction to the free open-source statistics and econometrics software named Gretl for cross-sectional, time-series as well as panel data. Gretl is an attempt to bridge the different demands for teaching at different levels but also by providing functionalities required for professional work. We provide basic information on Gretl and its powerful and intuitive scripting language "Hansl". We introduce Gretl and Hansl by going step-by-step from the scratch through a real-life data analysis example. Finally, we estimate a vector autoregressive dynamic regression model using Gretl for conducting impulse-response analysis as well as for forecasting purposes.

*The author is grateful to XXX for helpful comments. Any remaining errors or omissions are strictly our own.

1 Introduction

Data analysis is a vital part of the economic training and of an professional economist's daily life. Economists are trained to be sophisticated users of data, and econometrics — the interplay of economic theory, statistics and mathematics — is a subject which usually economics students have to attend. Undergraduates studying economics are typically exposed courses in statistics and econometrics, covering the practice and interpretation of least-squares regression, data management and data visualisation. Graduate students learn the advanced treatments of the subject, likely involving Maximum Likelihood estimation, simulation and so on. Professional economists — whether in public or private institutions — are likely to find themselves using, or possibly developing, econometric methods.

Gretl is an open-source and freely available statistics and econometrics software attempting to bridge the different demands for teaching at different levels as well as professional work. It comprises a full-featured graphical interface but also a powerful scripting language called hansl. Gretl's scripting language can be seen as a domain-specific language for statistics and econometrics. Gretl handles datasets — a matrix but with a richer structure including some meta-information — similar to Eviews but can also deal with pure matrices and arrays such as Matlab, Gauss and Julia for advanced programming purposes.

In the following, we will introduce Gretl — and to some extend hansl — alongside a practical empirical example using time-series data. Nevertheless, it should be mentioned that Gretl can also deal with methods and estimators for cross-sectional as well as panel data structures.

The paper is structured as follows. The next section discusses the market of econometrics software and introduces Gretl and its scripting language hansl. Section 3 introduces Gretl's graphical user interface and explains how to conduct data basic management and analyses steps accompanied by an overview about essential commands. In Section 4 a vector autoregressive time-series model will be estimated and it will be shown how the model's dynamics can be analysed for structural or forecasting purposes. Section 5 introduces the work with matrices before Section 6 summarises.

2 Econometric Software and Gretl

2.1 The 'Market' of econometrics software

The demands placed on an econometric software are a delicate issue. For teaching purpose it makes sense to have undergraduate students work with reasonably user-friendly software — in the easiest case controlling the software by a graphical user interface through point-and-click using the mouse. However, it makes little sense to introduce a software to undergraduates which does not support more advanced methods and does not offer a fully-fledged scripting and programming environment — aspects which are required for an advanced treatment in graduate classes and for professionals. And of course there's a premium on teaching "marketable skills" rather than dead-end expertise.

Gretl is a program which attempts to bridge these different demands. It comprises a full-featured graphical interface (the GUI): its underlying functionality can be driven either Gretl's scripting language or by the apparatus of menus, dialog boxes and so on. The developers try to ensure that almost everything that can be done via scripting can also be done via the GUI, and vice versa, with only few exceptions (Cottrell, 2017).

Gretl's main competitors are the major proprietary econometrics packages, Stata and Eviews, and also the major open-source statistical software project, R. As in Gretl, datasets and series are also basic in Stata and Eviews, and econometric functionality is supported by a wide range of built-in commands. Even though both Stata and Eviews support scripting, their respective languages are quite odd from the point of view of a programmer used to general-purpose scripting languages or Matlab-like interfaces for matrix manipulation. Gretl offers the common apparatus of fully-fledged programming languages (function-calling, function-definition, declaration of and assignment to named variables of various types) similar of what is known from Matlab, R and Python.

2.2 Gretl and Hansl

Gretl is acronym for **G**nu **R**egression, **E**conometrics and **T**ime-series **L**ibrary. The software is available for Windows, Mac OS X as well as Linux through the official Gretl homepage:

`http://gretl.sourceforge.net/`. It is free, open-source software which may be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation.

Gretl comprises a common command line (CLI) program and a GUI client, and makes use of reliable free and open-source software packages in the 'background' for computations, e.g. (multi-threaded) LAPACK/BLAS, fftw, GTK, gnuplot, etc. The first Gretl version was released in January 2000 and has been under active development since then. Its user interface is available in 17 languages thanks to an active community. Gretl is shipped with a User Guide of already 380+ pages, a Command Reference of 210+ pages, and a "A Hansl Primer" tutorial to Gretl's scripting language.

As the unique selling point, Gretl offers a high-level matrix oriented language with a syntax similar to Matlab or Gauss, and an intuitive high-level scripting language called *Hansl* that is attuned to econometrics. This makes Gretl especially interesting for lecturing statistics and econometrics but also for serious research.

Hansl — "**H**ansl's **A** **N**eat **S**cripting **L**anguage" — was developed over time and has become a very advanced high-level scripting language with more than 140 commands specifically tailored to the domain of econometrics to date. The lead developers, Allin Cottrell and Riccardo "Jack" Lucchetti, have pushed the progress of Gretl mainly through Hansl in recent times. Users and developers are asked to program and publish user-contributed function packages by means of Hansl rather than the low-level language C. A great plus of Hansl-based function packages is that it comes with a handy optional GUI integration making it possible for developers to add access to specific functions through the GUI rather than solely through functions. For more technical details on Hansl see Cottrell (2017).

To date Gretl *natively* implements a variety of models, methods and functions among others such as:

1. Time series data: ARIMA, GARCH-type, (S)VARs and (S)VECMs, unit root and cointegration tests, State Space modelling, Kalman filter, Structural Time Series Models, mixed time-series frequencies (MIDAS), etc.

2. Limited dependent variables: Logit, Probit, Tobit, interval regression, models for count and duration data, etc.

3. Panel-data: Instrumental variables, Probit, GMM-based dynamic panel models, etc.

4. Output models as RTF or LaTeX files, in tabular or equation format.

5. Using gnuplot for compiling high-quality graphs.

6. Estimation of models that are not natively present in Gretl neither handled by function packages, via the `mle` or `gmm` command block.

7. Support for machine learning via the LIBSVM library.

Besides Gretl's core functionality, several addons (`http://gretl.sourceforge.net/addons-data/addons.xml`) and numerous contributed function packages (`http://ricardo.ecn.wfu.edu/gretl/cgi-bin/gretldata.cgi?opt=SHOW_FUNCS`) are hosted online and are easily accessible through Gretl.

Additionally, and this may make Gretl very attractive for advanced applications, is the fact that that using Hansl one can estimate models that are not natively present in libgretl neither handled by function packages, via the `mle` command for Maximum likelihood estimation, the `gmm` command for General Methods of Moments estimation and other similar ones that require non-linear optimization. The Gretl team has implemented very efficient optimisation techniques much on the same level as what you get in Python, R or Matlab. For details and examples, we refer to Ch. 23 and 24 in the Gretl manual.

Another feature is that Gretl supports the capability to communicate with other software packages such as Stata, Python, R, Julia, Ox and Octave. See Section A.5 in the Appendix for details.

3 Getting Started in Gretl

3.1 Gretl's Graphical User Interface

For starters, the most intuitive way is to use the program through its built-in graphical user interface (GUI). The GUI is shown in Figure 1 and consists in total of four separate windows. At the top of the main window (see Figure 1(a)) you find the menu bar comprising commands to import and manipulate data, analyse data, and manage output. Below the menu bar, the user finds information on the loaded dataset and the working directory. Most space is given to the variables list where each variable is associated with a unique ID number, the variable's name and an optional descriptive label. Below the variables list, information on the loaded dataset is given such as the total number of observations or the number (and eventual time-span) of the currently selected sub-sample, respectively. The toolbar, containing a number of useful utilities, can be found at the bottom of the window.

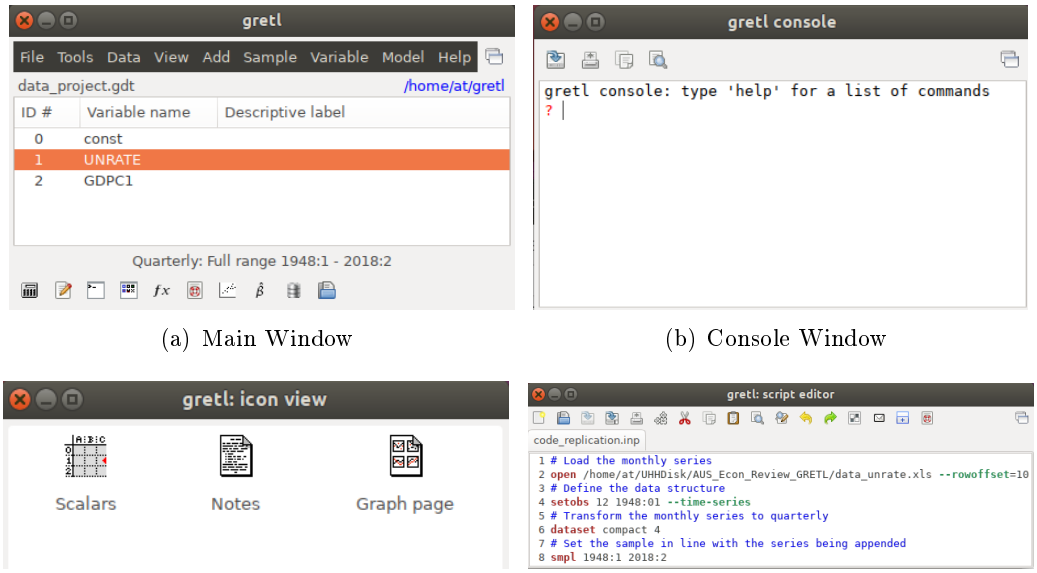
The console window, as depicted in Figure 1(b), accepts Gretl commands which can be directly executed pressing "Enter". The output will immediately appear in the console. From the command prompt, '?', you can type in commands from the Gretl language or calculate something. Try for instance,

```
? eval sqrt(25)/5
```

where the `eval` command evaluates an expression and prints its value.

The icon view window (see Figure 1(c)) comprises the access to different objects such as scalar values, matrices and model objects among others — more on this below. Lastly, the script editor, as depicted in Figure 1(d), collects several lines of programming code into a file which can be executed all at once in a script. Serious and reproducible research relies on such scripts for recording each step of the data analysis including comments on the code.

Lee Adkin's (2014) terrific and freely available econometrics Ebook using Gretl for the applications provides a much deeper introduction, and is highly recommended for learning and mastering Gretl (4th edition downloadable here:
`www.learneconometrics.com/gretl/using_gretl_for_POE4.pdf`; the 5th edition will be



(a) Main Window

(b) Console Window

(c) Icon View Window

(d) Script Editor Window

Figure 1: The Gretl Interface

published online very soon!). Also the manual, which can be opened from the "Help, User's guide" menu, is the first source you should check in case of questions or problems.

3.2 Loading and Appending Data

Practical empirical research starts with basic data management. Gretl can import data from a variety of formats from the local hard-disk and from the internet. In the GUI program this can be done via the "File, Open Data, User file" menu. In script mode, simply use the `open` command. The supported import formats include (i) plain text files (typically comma-separated or "CSV"), (ii) spreadsheet formats as MS Excel, Gnumeric and Open Document, (iii) Stata data files (.dta), (iv) SPSS data files (.sav) and (v) Eviews workfiles (.wf1) among some others.

The following conditions must be satisfied to make sure Gretl reads data successfully:

1. The first row must contain valid variable names, otherwise the program will automatically add names, v1, v2 and so on.

2. Data values should constitute a rectangular block, with one variable per column (and one observation per row).

3. Numeric data are expected, but in the case of importing from plain text, the program offers handling of character (string) data.

4. Optionally, the first column may contain strings such as dates, or labels for cross-sectional observations

One can append additional data either by the `append` command or via the "File, Append data" menu: Gretl will check the new data for conformability with the existing dataset and, if everything seems OK, will merge the data. For more complex tasks, there exists the powerful `join` command (execute `help join` in the console for details).

Once data is read-in, it may be necessary to supply some information on the type of the dataset. The three types that can be distinguished are cross-section, time-series and panel data. The primary tool for doing this is the "Data, Dataset structure" menu entry in the GUI, or the `setobs` command when using scripts or the console.

In our real-world example, we deal with two macroeconomic time-series downloaded from the St, Louis Federal Reserve Bank database called FRED. The first series is real GDP of the U.S. economy for which data values are collected at a quarterly frequency since 1948 (`https://fred.stlouisfed.org/series/GDPC`). The second series refers to the civilian unemployment rate observed at monthly frequency since 1948 (`https://fred.stlouisfed.org/series/UNRATE`).

The two separate spreadsheets look as depicted in Figure 2 with basic information on data characteristics reported in Table 1. The two files are stored online to allow the replication of the following example.

	A	B
1	FRED Graph Observations	
2	Federal Reserve Economic Data	
3	Link: https://fred.stlouisfed.org	
4	Help: https://fred.stlouisfed.org/help-faq	
5	Economic Research Division	
6	Federal Reserve Bank of St. Louis	
7		
8	GDPC1	Real Gross Domestic Product,
9		
10	Frequency: Quarterly	
11	observation_date	GDPC1
12	1948-01-01	2086.017
13	1948-04-01	2120.450
14	1948-07-01	2132.598
15	1948-10-01	2134.981
16	1949-01-01	2105.562
17	1949-04-01	2098.380
18	1949-07-01	2120.044

	A	B
1	FRED Graph Observations	
2	Federal Reserve Economic Data	
3	Link: https://fred.stlouisfed.org	
4	Help: https://fred.stlouisfed.org/help-faq	
5	Economic Research Division	
6	Federal Reserve Bank of St. Louis	
7		
8	UNRATE	Civilian Unemployment Rate,
9		
10	Frequency: Monthly	
11	observation_date	UNRATE
12	1948-01-01	3.4
13	1948-02-01	3.8
14	1948-03-01	4.0
15	1948-04-01	3.9
16	1948-05-01	3.5
17	1948-06-01	3.6
18	1948-07-01	3.6

Figure 2: Raw Data of Real GDP and Unemployment Rate for the U.S., Spreadsheet

Series	Frequency	Starting Date	Ending Date
GDPC1	quarterly	1948Q1	2018Q2
UNRATE	monthly	1948M1	2018M7

Table 1: Data Information of Raw Data

Working with these two series involves at least two problems. First, the frequency among the variables differs which requires some method to equalize it. Second, both series are stored in separate Excel files and need being joined. Both tasks can easily be solved.

Before proceeding, you should download both Excel-files and save them in a folder. The two xls-files can be downloaded from:

- data_unrate.xls: `https://scm.darnold.org/atecon/gretl_aer/raw/master/data_unrate.xls`
- data_gdp.xls: `https://scm.darnold.org/atecon/gretl_aer/raw/master/data_gdp.xls`

First, we load the monthly UNRATE series stored in the "data_unrate.xls" file via the "File, Open data, User file" menu which opens the "spreadsheed import" window (see Figure 3). As the name of the first variable ('observation_date') has its entry in the spreadsheet in column A and row 11, we start the data import of the rectangular block at this position as depicted in Figure 3.

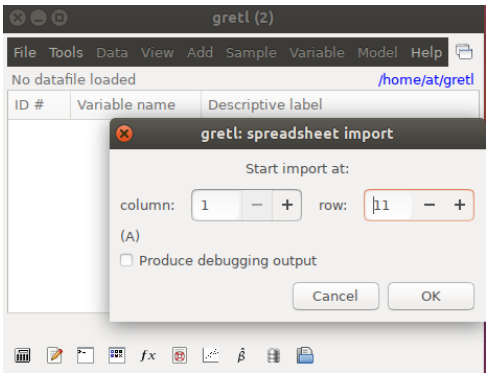


Figure 3: Spreadsheet Import Window

In case Gretl has not successfully recognized the time-series structure given some observation column (here 'observation_date'), the user can manually determine the frequency

and starting date by the menus automatically showing up in the process. Alternatively, we can set the data structure by the following command through the console

```
setobs 12 1948:1 --time-series
```

where '12' refers to the number of observations per year ('12' for monthly data) followed by the initial observation date and an option saying that the data has a time series structure. The variables which appear in the main window are of the so called 'series' type.

As real GDP is only quarterly observed, we need to transform the monthly UNRATE series to quarterly observations. A standard way to do this is to compute the average of three consecutive months of a specific quarter. This is easily done via the "Data, Compact data" menu or the command

```
dataset compact 4
```

which reduces the dataset from monthly to quarterly frequency. The resulting quarterly time-series will ranges 1948Q1 to 2018Q2 as indicated at the bottom of the main window.

The next step involves appending the quarterly real GDP series from another source to the existing dataset. However, when using the `append` command, we need to make sure that the currently active dataset covers the same time span as the one wished to append (the `join` command is much more flexible with regard to this). Thus, to avoid an error, we restrict the currently selected sample for the period between 1948Q1 to 2018Q2 by the command

```
smpl 1948:1 2018:2
```

or through "Sample, Set range" menu using the GUI.

Appending a file can be done by using the "File, Append data" menu followed by the same request of the 'spreadsheet import' window as already shown in Figure 3. Gretl automatically recognizes that the appended dataset has the same length as the active one and will add the real GDP series to the dataset. This will finally result in Figure 4.

Instead of using the GUI, all previous steps from (down-)loading the data, frequency reduction and appending can be done using a collection of Gretl commands as shown in the Appendix A.1.

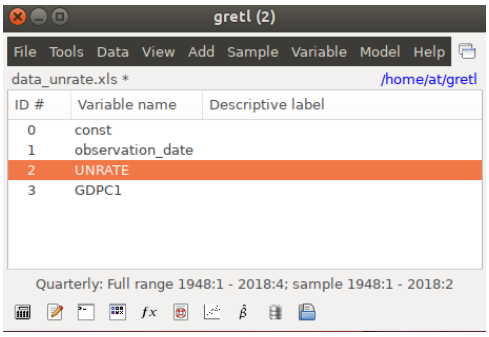


Figure 4: Joint Time-Series Dataset

It should be mentioned that Gretl includes many example datasets, some of them taken from popular econometrics textbooks, which is especially useful for teaching purposes. These can be accessed through the "File, Open data, Sample file" menus. Additionally, users can easily download data from some large databases such as DB.NOMICS, quandl.com and yahoo.com:

- DB.NOMICS: `https://next.nomics.world/`
  - Access through the "File, Databases, DB.NOMICS" menu
  - See video tutorial on youtube: `https://youtu.be/B2KmX9A6ICg`
- Quandl: `quandl.com`
  - Access through the `getQuandl` package: `http://ricardo.ecn.wfu.edu/gretl/cgi-bin/current_fnfiles/getQuandl.gfn`
- Yahoo: `yahoo.com`
  - Access through the `yahoo_get` package: `http://ricardo.ecn.wfu.edu/gretl/cgi-bin/current_fnfiles/yahoo_get.gfn`

3.3 Basic Commands

The purpose of this section is to introduce some basic commands which are frequently executed when doing data analysis. In the following, we proceed with the dataset compiled in the previous step. Once the data is loaded, one should check its validity. Initially, we

will generate a so called list which can (and should!) be used to make command scripts less verbose and repetitious, and more easily modifiable. In the following, a list named `L` comprises both variables of interest namely `UNRATE` and `GDPC1`. The list is set up by the `list` command:

```
list L = UNRATE GDPC1
```

For printing the values for each variable in a list by their respective observations, we use the `print` command with the `--byobs` option:

```
print L --byobs
```

The corresponding output for the first five observations looks like:

```
             UNRATE        GDPC1

1948:1       3.73333       2086.02
1948:2       3.66667       2120.45
1948:3       3.76667       2132.60
1948:4       3.83333       2134.98
1949:1       4.66667       2105.56
```

One can easily compute basic descriptive statistics for elements in a list by means of the `summary` (execute `help summary` in the console for obtaining additional options) command:

```
summary L
```

which returns various statistics as well as information on eventual missing values:

```
                Mean           Median         Minimum         Maximum
UNRATE          5.7771         5.5667          2.5667          10.667
GDPC1           8557.0         7013.6          2086.0           18507

                Std. Dev.      C.V.           Skewness   Ex. kurtosis
UNRATE          1.6318         0.28245         0.62860       0.096705
GDPC1           4956.1         0.57919         0.44045        -1.1495

                5% perc.       95% perc.       IQ range   Missing obs.
UNRATE          3.5050          9.0617         2.2083              0
GDPC1           2465.1           17234         9046.0              0
```

In case one wants to compute these descriptive statistics using only observations for a specific sub-set, say between 1980Q1 and 2007Q4, one simply needs to restrict the sample accordingly before calling the summary command:

```
smpl 1980:1 2007:4
summary L
```

One can also restrict the underlying sample in a more complicated way. For instance, running the command

```
smpl UNRATE>4 && UNRATE <= 9.6 --restrict --replace
```

would restrict the selected dataset to observations which fulfil the criteria that `UNRATE` is greater 4 but equal or less than 9.6, and replace any eventual previous restrictions imposed by the `--replace` option. The user will be informed that the currently selected sample includes only 237 out of all 282 valid observations, as depicted in Figure 5. The information that the restricted sample is "undated" states that due to the restrictions imposed, the time-series structure of consecutive observations has been broken. The original full sample with a time-series structure can be restored by the

```
smpl full
```

command.

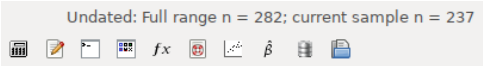


Figure 5: Sample Selection Information After Imposed Restriction

Sometimes one wants to manipulate data entries due to wrong or missing values recorded. Let us assume that variable `UNRATE` includes a missing value for observation 1981Q2 but you know that the U.S. unemployment rate has been 7.4 % for this observation. Replacing a valid value or a missing value for a specific observation can be simply done by the command

```
UNRATE[1981:2] = 7.4
```

where `[1981:2]` refers to a specific index (row) identified by a date-string of variable `UNRATE`.

A basic statistics for a bivariate analysis is the correlation coefficient. The correlation coefficient can be easily computed for all pairs of variables in a list by means of the command

```
corr L
```

which computes Pearson's product-moment correlation. Other types of correlation are also available — just execute `help corr` in the console for details. The correlation between $UNRATE_t$ and $GDPC1_t$ — where subscript t refers to the t-th observation for $t = 1, \ldots, T$ — is close to zero and is not significantly different from zero in this example:

```
corr(UNRATE, GDPC1) = -0.01416004
Under the null hypothesis of no correlation:
t(280) = -0.236967, with two-tailed p-value 0.8129
```

An overview about some of the available descriptive statistics are listed in Table 2.

Command	Description
`min`	Prints the minimum of the series
`max`	Prints the maximum of the series
`mean`	Prints the mean of the series
`median(X)`	Prints the median of the series X
`summary`	Prints summary statistics of the series
`xtab`	Displays a contingency table or cross-tabulation for series
`freq`	Prints/ Plots the frequency distribution of the series
`corr`	Prints the pairwise correlation coefficients for series
`xcorrgm`	Prints and graphs the cross-correlogram between two series
`pergm`	Computes and displays the spectrum of the series

Table 2: Examples of Descriptive Statistics in Gretl

3.4 Plotting

Visualising data marks an important step in empirical work. Gretl generates graphs by means of the reliable open-source tool gnuplot which is available for many platforms. Gnuplot is a very full-featured graphing program with myriad options (for details see Ch. 6 in the gretl manual). Gretl gives you direct access, via a graphical interface, to a subset of gnuplot's options and it tries to choose sensible values for you; it also allows you to take complete control over graph details if you wish.

In a time-series context, an obvious informative graph is the time-series plot which depicts data over time. This is easily done in Gretl by (the backslash allows for line breaks)

```
gnuplot L --with-lines=GDPC1 --with-impulses=UNRATE \
--time-series --output=display
```

and the graph can be seen in Figure 6(a). We ask `gnuplot` to plot the variables in list `L` over time using the `--time-series` option. Furthermore, we want series GDPC1 being drawn as a line but UNRATE as impulses in this example. The option `--output=display` returns immediately a plot onto the screen.

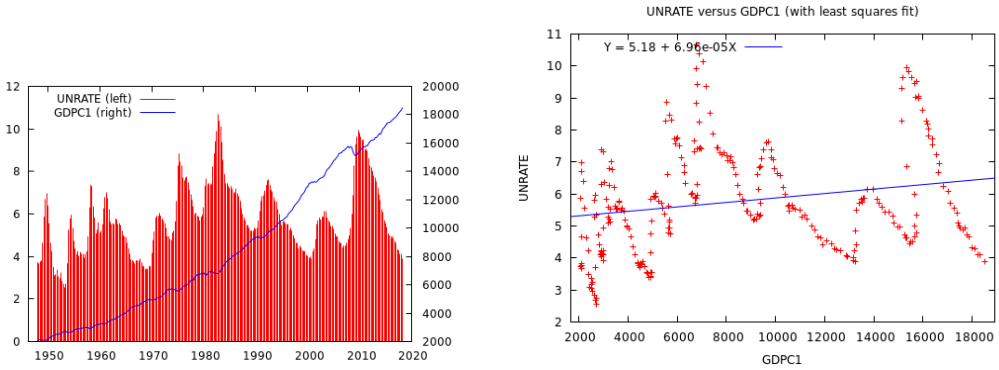
Also scatter-plots serve as standard tools for visualising relationships between variables. A simple scatter-plot between the two series included in list `L` — augmented by a linear OLS-based regression fit — is called by (see Figure 6(b)):

```
gnuplot L --fit=linear --output=display
```

The option `--output=filename` allows the user to specify a particular output format. Supported output formats are *.eps* (Encapsulated PostScript), *.pdf* (PDF) and *.png* (PNG) among others. For instance, for storing a figure in png-format in a specific directory, use

the expression:

```
--output="C:\Project\Figures\Fig1.png"
```



(a) Time-Series Plot

(b) Scatter-Plot

Figure 6: Time-Series and Scatter-Plot using Gretl

Another useful type of graph is the Boxplot which summarizes a number of descriptive statistics for the researcher. The `boxplot` command produces this type of graph. Before showing an example, however, let us generate a new series named `GreatModeration` which shall be a binary dummy variable taking the value of one for observations between 1985Q1 and 2007Q4, and otherwise zero:

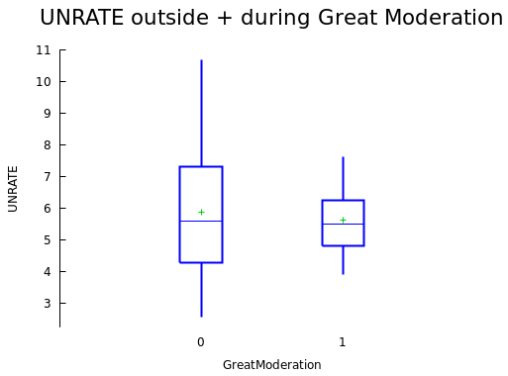
```
series GreatModeration = (obs>="1985:1" && obs<="2007:4")
print GreatModeration --byobs
```

The ones indicate the well-known Great Moderation period during which business cycle indicators such as GDP growth and price inflation rates followed a rather stable path associated with only small variance around some mean value.

Let us draw the boxplot for the `UNRATE` series and examine its distribution conditional on the value of the dummy variable `GreatModeration` by using the `--factorized` option:

```
boxplot UNRATE GreatModeration --factorized --output=display \
{ set title 'Distribution of UNRATE outside and during Great Moderation' ; }
```

Figure 7 shows that the variance of the unemployment rate has indeed been smaller during the Great Moderation phase in the U.S. compared to the remaining period; even though mean and median values have been of comparable magnitude. Note that we added a self-explanatory title to the graph.



(a) Boxplot

Figure 7: Factorized Boxplot using Gretl

4 Let's estimate a dynamic time-series model

In this section we want to illustrate how to estimate a dynamic time-series model using Gretl. We will to examine the dynamic relationship between the change in the unemployment rate and growth in aggregate output (as proxied by the Gross Domestic Product). The famous macroeconomist Arthur Melvin Okun formulated a negative relationship between these two variables which has become known as 'Okun's Law'.

Before estimating the regression model, we need to generate the two variables of interest. The following lines generate two new series objects where `dU` refers to the period-change in the unemployment rate, $dU_t = UNRATE_t - UNRATE_{t-1}$, and `gY` refers to the (exact) period-growth rate of real GDP (in percent), $gY_t = 100 \times \left(\frac{GDPC_t - GDPC_{t-1}}{GDPC_{t-1}}\right) = 100 \times \left(\frac{GDPC_t}{GDPC_{t-1}} - 1\right)$:

```
series dU = UNRATE - UNRATE(-1)
series gY = 100 * (GDPC1/GDPC1(-1) - 1)
```

As can be seen, in a time-series (but also panel) context, Gretl easily handles lags and leads by using an expression such as `x(p)` where p generates the p-th lag of series `x` if p takes a negative and the p-th lead if it takes a positive integer value.

4.1 The VAR model

The vector autoregressive model of order p (VAR(p)) is a commonly-used tool in macroeconomics. In its most basic incarnation it is written as

$$z_t = \mu + \sum_{i=1}^{p} \Phi_i z_{t-i} + \varepsilon_t \tag{1}$$

where z_t is an $k \times 1$-dimensional vector of observable quantities at time t, and μ is a $k \times 1$-dimensional vector of constants. The autoregressive coefficient matrices, Φ_i, are of size $k \times k$. In our case, we set $z_t = [dU_t, gY_t]'$. ε_t is an k-dimensional vector white noise process with time-invariant, positive definite covariance matrix $E(\varepsilon_t \varepsilon_t') = \Sigma_\varepsilon$. The econometrician's job is to estimate the unknown quantities μ, Φ_i and Σ_ε from the data so that the model can be used for policy analysis and forecasting.

For simplicity, we will base the estimation on a specific sub-sample ranging from 1985Q1 to 2007Q4 comprising in total $T = 92$ valid observations as this sample is associated with parameter stability. As a consequence of two oil crises in the mid and late 1970s as well as the recent Great Financial Crisis, Okun's Law has been associated with changed model coefficients and heteroskedasticity — so called structural breaks which violate some of the standard Gauss Markov assumptions required for an unbiased and efficient OLS estimator. The sample restriction is imposed by:

```
smpl 1985:1 2007:4
```

4.2 Determination of the optimal lag-order

The unknown optimal finite lag-order p is determined by means of information criteria (see every good introductory textbook on time-series analysis for details). We use Gretl's built-in `var` command with the option `--lagselect` to determine the optimal lag order by setting the maximum lag order to test twice the data frequency (=8 for quarterly data). Gretl will estimate model eq. (1) (the intercept is automatically added) for eight different lag lengths and compute three standard information criteria for each lag-order. Of course, the user could also use the GUI through the "Model, Time series, Multivariate, VAR lag selection"

menu for this.

Again we make use of lists, here named `ylist`, comprising both variables of interest. The commands and the corresponding output are:

```
list ylist = dU gY
var 8 ylist --lagselect
```

```
lags    loglik    p(LR)       AIC          BIC          HQC

1      -7.70739             0.297987     0.462451*    0.364366
2      -0.56199  0.00642    0.229608*    0.503716     0.340241*
3       1.35475  0.42901    0.274897     0.658647     0.429782
4       3.11277  0.47544    0.323635     0.817029     0.522773
5       4.19395  0.70592    0.387088     0.990124     0.630479
6       6.09821  0.43254    0.432648     1.145327     0.720291
7       8.02423  0.42640    0.477734     1.300056     0.809630
8      14.89473  0.00817    0.415332     1.347297     0.791481
```

While the Akaike (AIC) and Hannan–Quinn (HQC) information criteria select the optimal lag-order to be $p^* = 2$ — resulting in a VAR(2) model — the Schwarz (BIC) criteria selects a parsimonious VAR(1) model. We stick to the VAR(2) in the following example but the reader is invited to re-run the analysis for a VAR(1) model.

4.3 VAR estimation and specification testing

Estimating the VAR(2) model by means of OLS can be done by

```
Model <- var 2 ylist        # 'Model <-' constructs a model object
```

The `var` command is followed by the lag-order and the separate names of the endogenous variables or, as done here, a list of series. Further exogenous variables as well as deterministics like a linear trend can be added (execute `help var` in the console for details).

Note the optional expression "`Model <-`" preceding the actual `var` command. This expression generates a so called model object (here named "Model") which instantly appears in the icon view (remember Figure 1(c)). A double-click on the "Model"-icon opens the "model"-window (see Figure 8) where the user can proceed with the GUI for evaluating the estimation results, executing various tests, structural analyses, plot several estimation-related graphs such as residuals and fitted values and do forecasting. This underlines the nice implementation of graphical objects which is especially useful for beginners and teaching.

The well-formatted estimation output covers standard estimation information for each VAR equation.

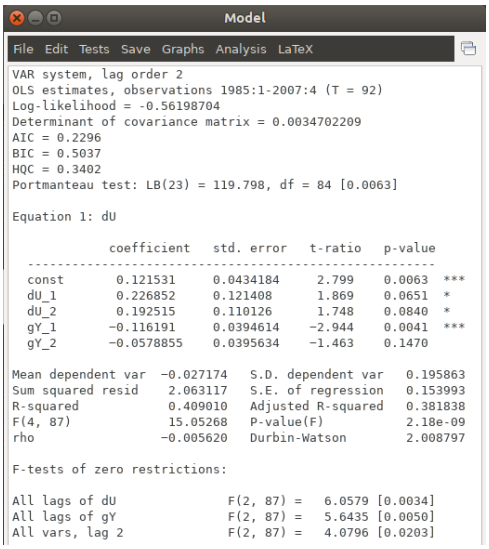


Figure 8: Model Window After VAR Model Estimation

Three diagnostic (multivariate) tests based on residuals are available after estimating a VAR model — for normality, auto-correlation and ARCH (Autoregressive Conditional Heteroskedasticity). These are implemented by the `modtest` command, using the options `--normality`, `--autocorr` and `--arch`, respectively. For details we refer to Ch. 29.4 in the Gretl manual. The following lines illustrate the use of these commands but the tests can also be executed under the "Tests" menu in the Model Window:

```
modtest 4 --autocorr  # test on 4th order serial correlation
modtest 4 --arch      # test for ARCH of order up to 4
modtest --normality
```

The output indicates that the model does not seem to suffer from any misspecification issues:

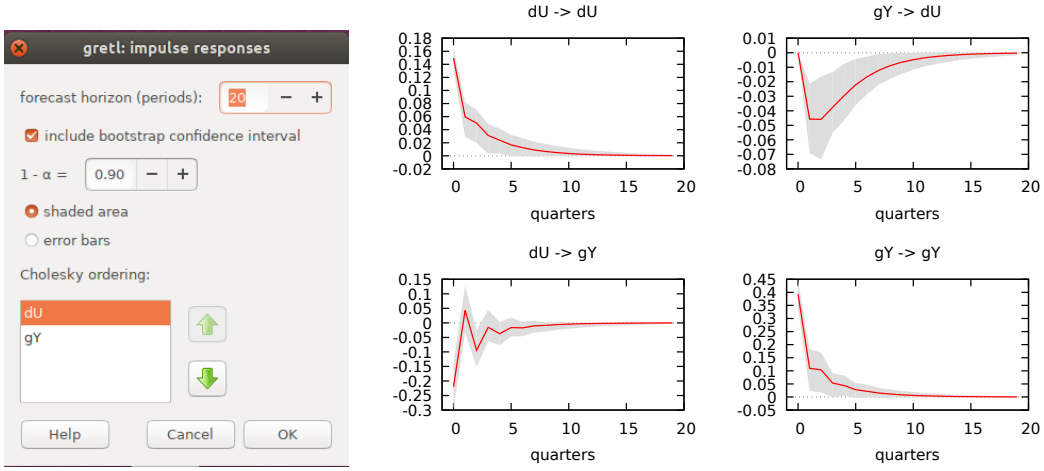
```
Test for autocorrelation of order up to 4
           Rao F   Approx dist.  p-value
lag 1     0.529    F(4, 168)     0.7146
lag 2     0.457    F(8, 164)     0.8849
lag 3     0.667    F(12, 160)    0.7808
lag 4     0.803    F(16, 156)    0.6806

Test for ARCH of order up to 4
             LM       df      p-value
lag 1    10.305      9       0.3264
lag 2    16.952     18       0.5264
lag 3    24.042     27       0.6280
lag 4    27.449     36       0.8463

Doornik-Hansen test
Chi-square(4) = 4.42308 [0.3518]
```

4.4 Structural analysis

Gretl's built-in `var` command supports the Cholesky decomposition-based approach, the classic and most popular "Structural VAR" (SVAR) variant. Impulse-response functions can be easily computed through the "Graphs, Impulse responses (combined)" menu in the Model Window. This opens a new window where the user can set some parameters such as the forecast horizon, the width of the confidence interval and the Cholesky ordering (see Figure 9(a). Figure 9(b) depicts the impulse-response results with bootstrap (per default 999 iterations) confidence intervals for each variable combination of the VAR(2) model. Computation is very fast as the underlying code is written in C. In case the user wants to use commands instead of the GUI, the steps can also be done via the command line interface by using the `--impulse-response` option when executing the `var` command.



(a) Settings for Impulse Response Analysis

(b) Results with 90% bootstrap confidence intervals

Figure 9: Impulse Response Functions with Bootstrap Confidence Intervals

Another tool, the so called Forecast Error Variance Decomposition, can be called through the Model Window through the "Graphs, Forecast variance decomposition" menu in a similar vein as done for impulse responses, or by the `--variance-decomp` option when executing the `var` command. We would also like to emphasize that the presented tools are also available for the "vector error-correction model" (VEC) model framework considering cointegration relationships between non-stationary series. Additionally, more advanced identification

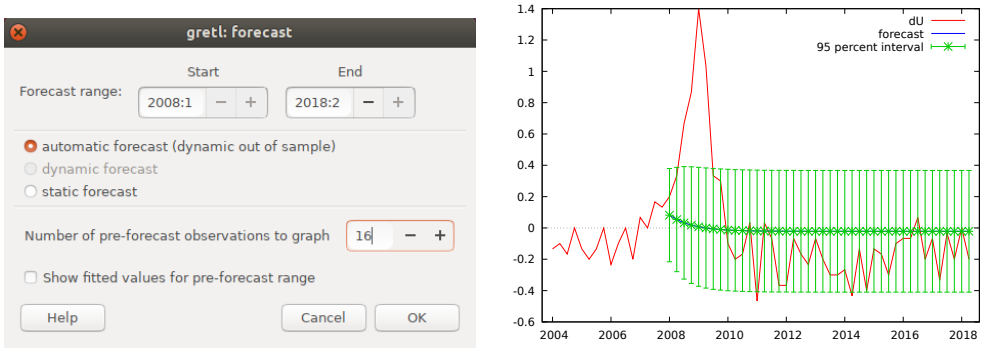
methods are included in the Gretl add-on `SVAR` which can be downloaded through the "Help, Check for addons" menu and which comes with a detailed manual accessible through the console by `help SVAR`. The SVAR functions can be called through the "Model, Time series, Multivariate, Structural VARs" menu.

4.5 Out-of-sample forecasting

Lastly, given our in-sample based estimates ending in 2007Q4, we want to forecast the dynamics of dU_t for the following quarters. This is called an out-of-sample forecasting exercise. Again, the user can use the `fcast` command for scripting or the Model Window GUI for doing this. When using the latter approach, forecast settings can be called through the "Analysis, Forecasts" menu in the Model Window. The settings for the forecast analysis are depicted in Figure 10(a), and the forecasting results with 95% intervals are shown in Figure 10(b). Obviously, our VAR(2) has failed to capture the rapid rise in the change in the unemployment rate as a result of the Great Financial Crisis in 2008/9. See Ch. 32.5 in the Gretl manual for details on how VAR-based forecasts are computed.

The `fcast` command also involves the option to compute basic forecast evaluation statistics, as can be seen when executing:

```
fcast 2008:1 2018:2 --dynamic
```



(a) Settings for Forecast Analysis

(b) Out-of-sample Forecast of dU_t

Figure 10: VAR Forecast Analysis

5 Working with matrices

Gretl's matrix language has similarity to the syntax of widespread commercial software packages such as MATLAB and GAUSS. The Gretl project allows an easy transfer of series objects to the matrix-world, and *vice versa*. For details on matrix manipulations, matrix algebra, statistics and transformations see Ch. 16 in the Gretl manual.

The following example shows how we could have estimated the VAR(2) model easily by means of standard linear algebra using the solution to the minimization problem underlying standard OLS. The solution is the famous equation $\hat{\beta} = (X'X)^{-1}X'y$ where y is a $T \times k$ vector of the endogenous, $[dU_t, gY_t]$, and X is the $T \times p*k$ matrix of k regressors and p lags (disregarding the intercept vector).

We declare the two matrix objects `y` and `X` by defining the respective matrices. The $\hat{\beta}$-matrix, `bhat`, is easily computed by using the `inv()` function for computing the inverse of a matrix, and `'` is a shortcut for the transpose of a matrix (equal to the `transp()` function). The commands for doing all necessary steps are:

```
matrix y = {dU}~{gY}   # takes eventually restricted sample into account
list xlist = const dU(-1 to -2) gY(-1 to -2)
matrix X = {xlist}
bhat = inv(X'X)*X'y
bhat = bhat'                # transpose k by 2 matrix
colnames(bhat, "const dU_1 dU_2 gY_1 gY_2")
rownames(bhat, "dU-eq: gY-eq:")
printf "%12.6g\n", bhat
```

The `printf` command prints the result

```
               const        dU_1        dU_2        gY_1        gY_2
dU-eq:      0.121531    0.226852    0.192515   -0.116191  -0.0578855
gY-eq:      0.352663     0.69436   -0.592979    0.278473     0.26639
```

which is exactly the coefficient matrix we have obtained before by the `var` command as shown in Figure 8.

6 Summary

Our intent has been to introduce the statistics and econometrics software Gretl and its scripting language Hansl. Gretl handles standard data types for doing data analysis. The language supports commands as well as a fully-fledged programming language. Commands

are accessible either via a graphical user interface or a command line. The software includes many functions tailored for statistics and econometrics purposes. Additionally, Gretl's community has programmed and published over 100 additional addons or packages continuously extending its functionalities. These features declare Gretl a serious competitor on the market for statistics and econometrics software which should be considered by students, teachers as well as professionals.

This paper has provided a step-by-step guide ranging from data loading, management and initial analysis over model specification and testing up to model analysis. We have estimated a dynamic vector autoregressive time-series model and shows how easily one can conduct a structural as well as forecasting analysis using Gretl.

We want to mention that Gretl is a continuously developing software. Interested users are asked to post Gretl-related questions, to request features and to report bugs on the Gretl mailings (`http://gretl.sourceforge.net/lists.html`). Furthermore, everybody is encouraged to develop user-contributed packages on which an open-source statistics and econometrics software heavily relies on.

References

Adkins, L. C. (2014). *Using gretl for Principles of Econometrics, 4th Edition*. Number 1412.

Cottrell, A. (2017). Hansl: A dsl for econometrics. In *Proceedings of the 2Nd International Workshop on Real World Domain Specific Languages*, RWDSL17, pages 1:1–1:10, New York, NY, USA. ACM.

A Appendix

A.1 Data Preparation and Cleaning Process

Below is a script for loading, defining, transforming, appending and storing the dataset used in Section 3.2. The script replicates all steps which were done using the GUI elements in Section 3.2 before. Copy and paste the following commands into a new script file in the editor window (see Figure 1(d)) and execute all commands by pressing "Ctrl+R".

```
clear                          # clear memory
set verbose off        # avoid printing commands

# Define a string variable for the URL where files are located
string url = "https://scm.darnold.org/atecon/gretl_aer/raw/master"
print url

# Load the monthly series from the server
open "@url/data_unrate.xls" --rowoffset=10 --preserve

# Define the data structure
setobs 12 1948:01 --time-series

# Transform the monthly series to quarterly
dataset compact 4

# Set the sample in line with the series being appended
smpl 1948:1 2018:2

# Append the real GDP series from the server
append "@url/data_gdp.xls" --rowoffset=10

# Drop the series 'observation_date'
delete observation_date

# Store the two time-series in native Gretl data format (*.gdt)
store "data_project.gdt"        # alternatively one can use "*.csv"
# Open the clean project file
open "data_project.gdt"
```

A.2 Add-ons and Contributed Packages

Both add-ons and (user-contributed) function packages add functionalities such as estimators, hypothesis tests or other analytical procedures to gretl's repertoire of built-in procedures. A list (including manuals) of available add-ons can be found here: `http://gretl.sourceforge.net/addons.html`. A list of contributed packages including a help test or manual and sample script can be accessed here: `http://ricardo.ecn.wfu.edu/gretl/cgi-bin/gretldata.cgi?opt=SHOW_FUNCS`

For details on function packages read the "Gretl Function Package Guide" which can be

downloaded by clicking the "Help, Function package guide" menu.

Function packages can either be acquired through the GUI or the command line. Using the command line, one needs to download (at least once to store it on the hard-disk) the package from the package server, and load it into memory at the beginning of each gretl session. A function package ships new functions which can be called. For details one has to read the help text or manual of a specific package. The following commands illustrate the steps:

```
install ADMBP                # download package from server
include ADMBP.gfn            # load into memory
help ADMBP                   # opens help text or pdf-manual (if available)
```

A.3 Loops

For details read Ch. 12 in the Gretl manual. The command `loop` opens a special mode in which gretl accepts a block of commands to be repeated zero or more times. The general form of a loop is:

```
loop control-expression [ --progressive | --verbose | --quiet ]
        loop body
endloop
```

Gretl supports various loop control variants such as "Count", "While", "Index", "Foreach" and "For" loops.

A.4 Functions

For details read Ch. 13 in the Gretl manual. Gretl offers a mechanism for defining functions, which may be called via the command line, in the context of a script, or (if packaged appropriately, see section 13.5) via the program's graphical interface. The syntax for defining a function looks like this:

```
function type funcname (parameters)
        function body
end function
```

Here is an example for the use of functions:

```
# function definition
function scalar ols_ess (series y, list xvars)
        ols y 0 xvars --quiet
        printf "ESS = %g\n", $ess
```

```
        return $ess
end function

# main script
open data4-1
list xlist = 2 3 4
# function call (the return value is ignored here)
ols_ess(price, xlist)
```

A.5 Communication with other software packages

Gretl support the communication with other popular software packages. Through the so called "foreign mechanism" the user can interpolate into a hansl script a set of statements to be executed by another program, with apparatus available to ferry data between the programs. This facility is supported for Octave, R, Python, Ox, Stata and Julia. This is especially useful to exploit functionality in the "foreign" program that is not currently available in Gretl. For details see ch. 39 to 44 in the Gretl manual.