A (very) short introduction to Gretl using scripts

Gaëlle Le Fol

August 22, 2022

1 Introduction

Gretl is a package for econometric analysis, written in the C programming language. It is a free, open-source software. It has an easy intuitive interface (with rolling menus). A wide variety of econometrics estimators are available such as least squares, maximum likelihood, GMM, single-equation and system methods) or time series methods like ARMA, GARCH, VARs and VECMs, unit-root and cointegration tests, etc. This software can be downloaded at `http://gretl.sourceforge.net/` and run under Windows, Mac/OS or Linux.
Gretl can also use scripts and this (short) introduction aims at giving you the first necessary help to work with scripts.

2 Getting started

2.1 Installing Gretl

Gretl can be download on any computer for free. Check for system requirements. Gretl for

- Windows can be downloaded from `http://gretl.sourceforge.net/win32/`
- macOS can be download from `http://gretl.sourceforge.net/osx.html`

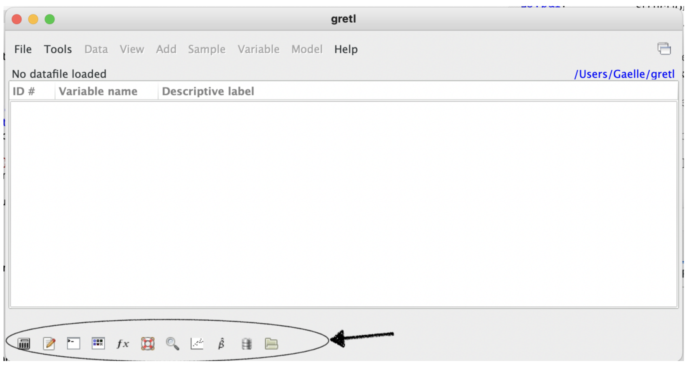
and choose Version 2022a.

2.2 Choosing preferences

Go into the **Tools** Menu and pick **Preferences** and then **General** to choose, the Main Gretl' directory (where most of Gretl's files are - like help files for example), the appearence (Theme preference) and the language (Language preference). Note that, any change will only take effect after you restart Gretl.

2.3 Gretl windows

The data window, called **gretl,** is the window where you can access the data and any other window proposed by Gretl from the Menus or from the bottom of the window.


gretl
File Tools Data View Add Sample Variable Model Help
No datafile loaded
/Users/Gaelle/gretl
ID # Variable name Descriptive label

The "**gretl: script editor**", is the window that allows you to create new scripts or gives you access to existing scripts. The provided editor is syntax highliting (see below).
The "**gretl console**" window is where you can execute directly some instructions.
The "**gretl: icon view**" is giving you access to all objects created since the beginning of the session (scalars, models results, data sets, matrices ...).
The "**gretl: script output**" is the execution journal window where you will also find the results of the executed set of instructions.

2.4 Working with scripts

To create a new script go intro the **File** Menu and choose **Script Files** > **New script** > **Gretl script**. Alternatively, you can also click on the "New script" button of the bottom of the Gretl main window. A new window pops up called **gretl: script editor**.

This editor is a syntax highlighting editor with the following color indications:

- **Commands**: commands are instructions, eventually with options, that return a set of values and characters. Ex.: **genr**, **labels**, **open**, **print**, **set**, **summary**.
- `Options`: options inside commands are usually preceded by a double minus sign like for example `--by`, `--delete`,`--output`, `--quiet`.
- `Comments`: comments start with a hashtag sign (#) and finish when ending the line.
- `Functions`, and Implicit functions: functions are instructions that return a value. Ex.: `abs`, `mean`,

obs, ones, sum. Implicit functions are also instructions that return a value but they can only be used as the extraction of a command result and as such, they refer to the last command that has been executed. They are always preceded by a $ sign. Ex.: $coeff, $ess, $uhat, $yhat.

- Text: Text is a string character to be printed in a text file or to save some results out of a command. It has to be delimited by double quote signs.
- Types: Types of objects. It can be a matrix, a series, a scalar, or a list.

2.5 Getting help

In order to access the list of the commands or functions and implicit functions (and the details on them), go into the **Help** menu of the Gretl main window and choose **Command references** or **Functions references**. You can aslo directly type **help** in the Gretl console window :

```
> help # List of available commands
> help functions # List of available fcts
> help smpl # Details of the command smpl
```

2.6 Running a script & visualizing objects

To run a script, you should either "Ctrl + R" (Windows) or "Command + Control + R" (Mac), or clic on the run button . If you want to run only part of the script, you just have to select the part you want to run before running it.

2.7 Working directory

Your working directory is the folder on your computer in which you are currently working. This working directory is the one where Gretl will look for data sets to load by default or where it will write data sets, graphs or text files.

```
set workdir "/Users/Gaelle/.../myworkingdir"
```

3 Data structure

3.1 Scalars, vectors and matrices

Vector are one-dimensional arrays and matrices are one- or two-dimensional arrays of double-precision floating-point numbers. They both fall into the matrix type. Vectors and matrices creating examples are:

```
matrix X = zeros(2,3) # 2-lines and 3-colums matrix of 0
matrix Z = ones(2,1) # 2-lines vector of 1
scalar a = 2*3 # scalar = 6
matrix A = { 1, 2, 3 ; 4, 5, a } # 2-lines and 3 colums matrix with special values
```

Operations on matrices

```
matrix TA = t(A) # matrix transposition, same as A'
matrix MP = A * B # matrix product
matrix INVC = inv(C) # inverse matrix
matrix KP = A ** B # Kronecker product
matrix ACUBE = A^3 # matrix power
matrix HC = A ~ B # horizontal "concatenation" operators
matrix VC = A | B # vertical "concatenation" operators
matrix VC2 = { X, Y } # vertical "concatenation of series"
```

3.2 Series

Series are variables and the element on which Gretl can work directly. Converting a matrix into a series is sometimes very interesting. For example, if you can plot matrices (using the accurate option in the **gnuplot** command), plotting series are a lot easier. To convert, vectors or matrices to series, you must be sure that they have the same number of observations as the other series in the dataset. Otherwise, you will get an error.

```
series Aseries=A[1;2:T+1] # Convert first column of matrix A, from line 2 to T+1 into a serie
```

3.3 Lists

Another basic structure is the list. - an array containing integers or characters. The main advantage of lists is that the "columns" do not have to be of the same length, unlike matrices and series.

```
list myvariables = varname1 varname2
```

4 Graphs

4.1 Scatter plots

A scatter plot is a type of plot using Cartesian coordinates to display values for two variables of a set of data.

```
gnuplot A B --output=AB.pdf # saved in a pdf file
gnuplot A B --output=display # display on the screen
gnuplot A B --fit=linear --output=display # with a regression line
```

4.2 Time series plots

A time series plot is a type of plot displaying the evolution of one variable against time. Note that, you cannot do a time series plot if you did not declare your data as being a time series (see section 6, hereafter).

```
gnuplot A --time-series --with-lines
--output=display
gnuplot A B --time-series --with-lines
--output=ABTS.pdf {set title 'Evolution of
A and B'; set xlabel "Dates"; set grid}# Time
evolution of A and B, with a title, a label
on the X-axis, and a grid
```

4.3 Histogram

A histogram is a graph used to represent the frequency distribution of one variable. Histograms often classify data into various "bins" or "range groups" (on the $x-$axis) and count how many data points belong to each of those bins (on the $y-$axis).

```
freq A --plot=display # Analyse the frequency
distribution of A and draw a histogram
```

5 Reading, merging and writing data files

5.1 Loading data

You can load many different data sets, - ASCII files (".txt" or ".cvs"), Excel files (".xls" or ".xlsx"), ..., and also Gretl' dataset (".gdt"), but any call with no extension means gdt. By default, Gretl considers that the data is in the working directory.

```
open data1 # Open the Gretl dataset named
data1
open "/Users/Gaelle/.../mydata/data1" #
Loading data1.gdt giving the path to it
open "Data1.xlsx" --rowoffset=1
--sheet="Sheet1" # Open the spreadsheet
"Sheet1" from the Excel dataset named Data1,
starting line 1
open "Data1.txt" # Open the ASCII file named
Data1.txt
```

5.2 Merging data

You can merge two datasets by adding new variables or new observations. Note that the first dataset is giving the final size of the merger.

```
open "Data1.gdt" # Open the Gretl dataset
named Data1
append "Data2.gdt" # Append the contens of
Data2 to the current dataset
```

Note that, if the two datasets you are merging have the same variables names, Gretl will pile them up as you would do when adding observations. On the contrary, if the variables names are not the same, Gretl will merge the two datasets horizontally as you would do when adding variables.

5.3 Writing data

You can save data to a file. By default, all current series are saved but you can also list the variables to be saved. By default, Gretl will save "gdt" files but you can also choose other formats.

```
store "Data1_2" # Store the current dataset
in a Gretl database called "Data1_2.gdt"
store "Data1_2" A B # Store only the two
series A and B in the Gretl database called
"Data1_2.gdt"
store "Data1_2.csv" # Store the current
dataset in a comma separated value format
database called"Data1_2.csv"
```

6 Interesting other commands

6.1 Data management

The **setobs** command allows to declare the type of data you are working with as well as the periodicity of your data.

```
setobs 1 1 --cross-section # Cross section
data from 1 with a step of 1
setobs 1916 1950 --time-series # Yearly data
from 1916 to 1950
setobs 12 1:1 --time-series # Monthly data
from 1:1 ...  1:12, 2:1 ...
```

The **smpl** command helps restricting the sample. You can delete the missing observations.

```
list X = x1 x2 x3 # X is a list of 3 names
x1 x2 and x3
smpl --no-missing X # Getting rid of missing
obs.  in X
smpl 2012:02 2022:08 # Restricting the
sample to Feb.  2012 to Aug.  2022
smpl 1 100 # Restricting the sample to the
first 100 observations
smpl --full # Coming back to the entire
sample
```

The **dataset** command performs various operations on the data set as a whole, depending on the given keyword, which must be `addobs`, `insobs`, `clear`, `compact`, `expand`, `transpose`, `sortby`, `dsortby`, `resample`, `renumber` or `pad-daily`. Note: except for clear, these actions are not available when the dataset is currently subsampled by selection of cases on some Boolean criterion.

```
dataset   clear # Clears out the current data, returning gretl to its initial \empty" state
open "Data1.gdt" # Open the daily dataset Data1
dataset compact 12 last # Compact to Montly data, keeping the last daily obs.
```

The genr command allows to generate different variables, matrices or scalars. It can also creates time trends, indices or dummy variables:

```
genr b=1.5 # b is a scalar
genr b={1.5} # b is a matrix 1 x 1
genr time # time is a time trend var.
genr index # index is an index var.
genr timedum # dt1, dt2 are dummy time var.
```

To rename or delete a variable, you can use the following commands:

```
rename X XNEW # X becomes XNEW
delete X # X is deleted
```

The summary command computes summary statistics and can also provide statistics for sub-samples.

```
summary X # Statistics for X
summary X --simple # basic statistics only
summary X --by=year # descriptive statistics by year.  Note that year has to be an existing serie
```

The freq command with no options given, displays the frequency distribution for the series var (given by name or number), with the number of bins and their size chosen automatically.

```
freq x --normal # Compute the frequency dist. and test for normality
freq x --plot="Fig1.pdf" # Compute the frequency dist.  and save the histogram in a file called Fig1.pdf
```

The ols command is performing an Ordinary Least Square estimation and provides usual regression statistics.

```
ols y const x1 x2 # Regression of y on a constant (const) and 2 explanatory var.  x1 and x2
ols 4 0 5 # Regression of var nb 4 (of the current dataset) on the explanatory var nb 5 with no constant
```

6.2 Printing & writing

The print command is a basic printing command which prints out any type of object (scalar, matrix, series ...), separated by a space, in the output window. Note that, series cannot be mixt with anyother type of object.

```
print y x1 x2 # Print the series y x1 x2
print A a # Print the matrix A and the scalar a
```

The printf command prints scalar values, series, matrices, or strings under the control of a format string.

```
ols 1 0 2 3
scalar b = $coeff[2] # b takes the estimated OLS coef.  associated with var2
scalar se_b = $stderr[2] # se_b takes the estimated OLS standard dev.  of the coef. associated with var2
printf "b = %.8g, std % .8g, t = %.4f \n ",b, se_b, b / se _b # Print the sentence with the 2 estimated parameters and their ratio
```

The possible available formats for the printf command are %s for strings, %d for integer, %x1.x2f for floating-point value of x1 digits in total with x2 decimals (%10.6f : 10 digits in total with 3 digits in the decimal part, %.5f : five decimals), %e for scientific notation, and %g prints the number in the shortest representation among the two.

The outfile command starts a block in which any printed output is diverted to a file or buffer. Such a block is terminated by the command end outfile, after which output reverts to the default stream.

```
outfile Results.txt
      ols 1 0 2 3
end outfile # Print the submitted command and its results in the file text Results.txt
```

```
outfile Results.txt --quiet # To only print the results of the osl regression
      ols 1 0 2 3
end outfile
```

7 Programming tools

7.1 If-statement

If-statments should be used, if a set of instructions has to be executed only under some conditions. The instructions can be of the following simple form:

```
if condition
      instructions
endif
```

where the condition must be a Boolean expression.

Example :

```
if k>=150
      dum[k,1]=1
endif
```

If you have two (or more) sets of instructions, we have:

```
if condition
        instructions1
else
        instructions2
endif
```

Example :

```
if name == "Linear relation"
outfile "file1.txt" --quiet
else
outfile "file2.txt" --append --quiet
endif
```

Finally, if you have 2 (or more) conditions, we get:

```
if condition1
        instructions1
elif condition2
        instructions2
else
        instructions3
endif
```

7.2 Loops

The loop command opens a special mode in which the program accepts commands to be executed repeatedly. You exit the mode of entering loop commands with endloop.

```
loop 1000 # Loop 1000 times
loop i=1991..2000 # Loop for i from 1991 to
2000, step = 1
loop 1000 --progressive # For Monte Carlo
simulations.  Will print summary statistics
at the end of the loop.
loop  while b > .00001 # The loop will conti-
nue as long as b is greater than 0.00001
loop    for (r=-.99; r<=.991; r+=.001) # Loop
for values of r from -0.99 to 0.99 by steps
of 0.01
loop foreach i xlist # Loop for each value of
i in a list called xlist
loop foreach i x1 x2 x3 # Loop for each value
of i in x1 x2 and x3
```

Each of these loop commands must be ended by an endloop command.

7.3 Building your own function or command

Gretl allows users to define their own functions or commands. The difference betwwen the two being what thei-y return (if anything). A command do not return anything while a function does. They both may be called via the command line. The syntax for defining a function or a command looks like this:

```
function type fctname (parameters)
          instructions
          return returned object
end function
```

The `type`, which states the type of value returned by the function, if any. This must be one of `void` (if the function does not return anything), `scalar`, `series`, `matrix`, `list`, `string`, `bundle`, `matrices`, `bundles` or `strings`.

The `parameters`, to be listed between brakets, include not only the name of the parameters but it should be preceded by the type of the parameter, and that is again one of `scalar`, `series`, `matrix`, `list` or `string`.

Example of a command:

```
function void ols_ess1 (series y, list xvars)
# The command is called ols_ess it takes y
and the list of explanatory variables as
parameters and returns nothing)
    ols y const xvars --quiet # We run the
    # regression and print nothing out
    printf "ESS = %g\n",$ess # We print the
    # ESS with a format
end function
```

Same type of example building a function:

```
function scalar ols_ess2 (series y, list
xvars) # The function is called ols_ess
it takes y and the list of explanatory
variables as parameters and returns the
error sum of squares (ess)
    ols y const xvars --quiet # We run the
    # regression and print nothing out
    printf "ESS = %g\n",$ess # We print the
    # ESS with a format
    return $ess # The function returns the
    # ESS
end function
```

Calling your function: A user function is called by typing its name followed by zero or more arguments enclosed in parentheses. If there are two or more arguments they must be separated by commas.

We built a function called `ols_ess` with two arguments, an endogenous variable (series) and a list of exogenous variables (list). It runs the ols regression and return the error sum of squares. We will first open the data set that contains the variables. Built the list of exogenous variables names, and call our command or function:

```
open data1 # The data are in data1 - a gretl
data base
list xlist = 2 3 4 # We built a list made of
the values 2 3 and 4
```

Calling my command `ols_ess1`:

```
ols_ess1(price, xlist)# We pass the prices
series for y and xlist for xvars.  We run
the ols regression and print the ESS, the
error sum of squares of the regression of
prices on a constant and variables numbers 2,
3 and 4
```

and same for my function `ols_ess2`:

```
scalar ESS = ols_ess2(price, xlist) # Same as
the command, but we return the value os the
computed ESS.
```

8 References

For more details and/or more commands and functions, please refer to the Cottrell & Lucchetti (2022) Gretl's Users Guide: http://gretl.sourceforge.net/gretl-help/gretl-guide.pdf
Another interesting reference, is the Adkins' guide (2018) "Using gretl for Principles of Econometrics" that you can find at http://www.learneconometrics.com/gretl/poe5/using_gretl_for_POE5.pdf.