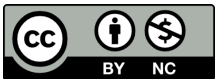
Hansl para principiantes

Ignacio Díaz-Emparanza

29 de septiembre de 2014

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea



El autor desea agradecer la ayuda financiera del Gobierno Vasco a través del Grupo consolidado: GIC12/14 y del Ministerio de Economía y Competitividad a través de los proyectos ECO2010-15332 y ECO2013-40935-P.

Índice

Preámbulo

El objetivo de esta guía es dar unas primeras indicaciones al usuario de *Gretl* que ya ha utilizado el programa mediante su interfaz gráfica (ventanas y menús) y desea aprender a programar nuevos estimadores, modelos y funciones que *Gretl* no tiene implementados y, por tanto, no están accesibles por medio del menú.

Esta guía se puede considerar que es un complemento de la guía *Introducción a Gretl 1.9.12 para Econometría* de Goitisolo y Moral (2013), que explica el uso de *Gretl* mediante su interfaz gráfica y, en parte, se ha basado en su metodología y ejemplos. Quiero agradecer aquí la colaboración de Beatriz Goitisolo y Paz Moral que han leído versiones provisionales de este documento y cuyos comentarios han ayudado a mejorarlo de manera muy significativa.

Nota sobre la terminología: A lo largo de esta guía se utiliza varias veces el anglicismo *comando* que viene de la palabra inglesa *command*. El diccionario de la Real Academia Española tiene una entrada para la palabra *comando* que interpreta como: *1) Mando militar; 2) Pequeño grupo de tropas de choque, destinado a hacer incursiones ofensivas en terreno enemigo 3) Grupo armado de terroristas*. Obviamente ninguna de estas acepciones es válida en nuestro contexto. Utilizo el término *comando*, ya que lo considero más preciso y específico que otras posibles traducciones de la palabra inglesa *command*, como *orden* o *instrucción*. Algún otro diccionario (por ejemplo WorReference.com) ya traduce este término como *instrucción informática* y es en este sentido como será aquí utilizado, es decir, para denotar una palabra o cadena de caracteres que sirve para pedir al ordenador que ejecute una determinada acción. Esa palabra o cadena a veces suele requerir también uno o varios parámetros que irán separados de ella mediante un espacio, especificados entre paréntesis o precedidos de dos guiones `--`.

Nota sobre copiar y pegar: En todos los guiones de este documento se utilizan fuentes monoespaciadas. Es posible seleccionar en su visor de PDF el texto de estos guiones y copiarlo en el editor de guiones de Gretl. Sin embargo, hemos comprobado que a veces esto añade espacios donde realmente no los hay. Por ejemplo, suelen añadirse a la izquierda y derecha del signo =. Por favor, tenga esto en cuenta si obtiene errores en la ejecución de alguno de estos guiones.

Ignacio Díaz-Emparanza

1. Introducción a Hansl: Usando Gretl desde la línea de comandos

Hansl (el nombre puede considerarse un acrónimo recursivo de "Hansl's a neat scripting language"[1]) es el lenguaje de programación de *Gretl*. No es nada nuevo y no es un complemento o algo así, es parte integral de *Gretl*.

El componente fundamental del *motor de cálculo* de *Gretl* es la biblioteca de funciones *libgretl* que está construida en el lenguaje de programación C. Esta biblioteca contiene la mayor parte de las funciones matemáticas disponibles en *Gretl* y distribuye el acceso a otras complementarias que contienen código específico para determinados fines. Dicha biblioteca se puede consultar mediante dos interfaces de usuario: la interfaz gráfica, formada por menús y ventanas (que en el manual en inglés de *Gretl* se conoce como GUI[2]) y la interfaz de texto, en la que las órdenes se dan mediante comandos (que en el manual se denomina CLI[3]). Los comandos de la interfaz de texto forman un lenguaje que es lo que denominamos *Hansl*.

Junto a la versión 1.9.90 de *Gretl* sus programadores, Allin Cottrell y Jack Lucchetti, han publicado el documento de presentación de *Hansl*, *A Hansl Primer*, que está accesible desde el menú Ayuda de la propia interfaz gráfica de *Gretl*. Recomendamos su lectura como segundo paso en el aprendizaje de *Hansl* después del presente documento. Este documento está también basado en la versión 1.9.90 de *Gretl*.

Hansl está formado por un conjunto de comandos que sirven para pedir al ordenador que realice diferentes acciones. A diferencia de otros lenguajes, los comandos de *Hansl* están especialmente diseñados para el cálculo econométrico. Los lenguajes de programación normalmente tienen muchos comandos y funciones y para utilizarlos suele ser necesario tener cerca los manuales. *Gretl* y *Hansl* tienen dos manuales (en inglés): la *Gretl User's Guide* (Guía del Usuario) y la *Gretl Command Reference* (Guía de Instrucciones). Estas guías están disponibles en formato `pdf` en el menú Ayuda de *Gretl*. En ese mismo menú, en sus dos primeros elementos, podemos también consultar la *Guía de Instrucciones* y la *Guía de Funciones* mediante una interfaz interactiva. Además, se pueden descargar las versiones actualizadas de los manuales desde http://sourceforge.net/projects/gretl/files/manual/

2. Estructura de un guion

Los programas (o guiones) en *Hansl* son ficheros de texto plano (ASCII), que por defecto llevan extensión .inp y contienen las instrucciones que queremos que el programa

[1] *Hansl es un lenguaje de programación ordenado.*
[2] Graphical User Interface
[3] Command line interface

ejecute de manera secuencial, línea por línea. Para reflejar en este documento el contenido de un archivo de texto se presentará la información en recuadros de fondo gris con fuente de texto monoespaciada, así:

Guion 1:

```
#Función de consumo
open data3-6.gdt
ols Ct const Yt
```

En este pequeño ejemplo podemos ver ya algunas cosas importantes:

- Una línea que comienza con el carácter `#` es un comentario. Cuando se encuentra este símbolo, todo lo que haya desde ese punto hasta el final de la línea se trata como un comentario y, por tanto, será ignorado por el intérprete de comandos de *Gretl*.
- La siguiente línea contiene un *comando* (`open`) que requiere un *argumento*, el nombre del archivo: `data3-6.gdt`. Esto es bastante típico en *Hansl*. La mayoría de las tareas se realizan invocando comandos. Una vez abierto un fichero, se pueden realizar cálculos u operaciones con las variables que contiene.

  El archivo *data3-6.gdt*, que es uno de los que vienen con la instalación de *Gretl*, contiene dos variables: `Ct`, consumo personal, e `Yt`, renta personal *per cápita* de EEUU en datos anuales desde 1959 hasta 1994, en dólares constantes de 1992.
- En la tercera línea de este ejemplo se calcula la regresión mínimocuadrática de `Ct` sobre una constante e `Yt`.

Algunos lenguajes de programación utilizan, por ejemplo, el punto y coma `;` para indicar el final de un comando. *Hansl* no tiene un símbolo explícito para terminar un comando. En lugar de esto, utiliza el carácter de *nueva línea*[4] y el carácter de *final de fichero*[5]. El carácter de *nueva línea* es un carácter oculto que solo se aprecia en la pantalla porque el texto a continuación aparece más abajo, en una nueva línea. Así que entonces, al finalizar un comando hay que insertar una nueva línea, excepto cuando el comando está en la última línea del fichero. Es decir, no se pueden poner dos comandos seguidos en la misma línea.

Además, no se puede insertar una nueva línea en mitad de un comando, o no directamente. Si necesitamos partir un comando en más de una línea, se puede usar para ello la barra invertida "\", que hace que *Gretl* ignore el siguiente carácter de nueva línea.

[4] El carácter que en informática se conoce como LF, de *line feed*.
[5] El carácter EOF, de *end of file*.

Hansl no tiene una limitación práctica para la longitud de un comando[6] pero, para facilitar la lectura del guion, es muy recomendable partir los comandos largos en varias líneas. Usaremos para ello la barra invertida.

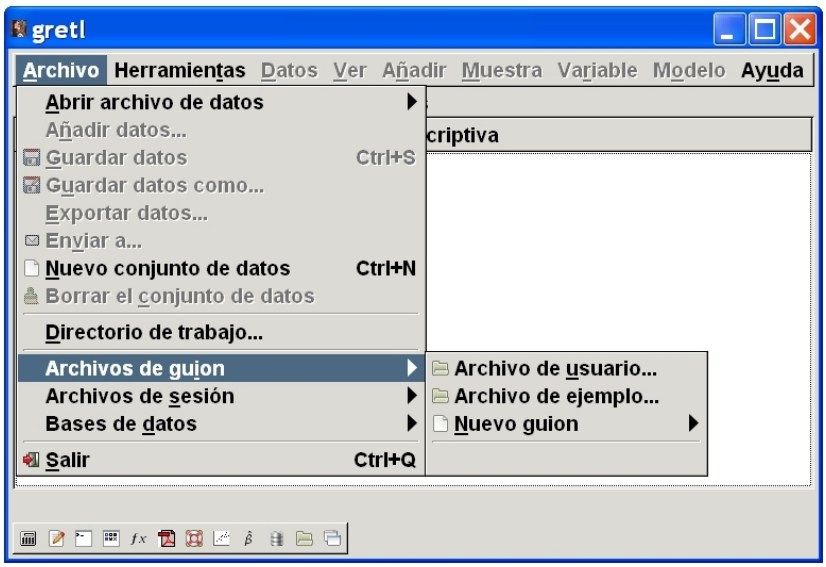
Las primera línea de un programa en *Hansl* suele contener:

- La instrucción `open fichero-de-datos` para abrir un fichero con datos o
- la instrucción `nulldata número` en el caso de que vayamos a trabajar con datos simulados. Por ejemplo, escribiremos `nulldata 150` para trabajar con series de 150 observaciones.

Pero si en nuestro programa estamos definiendo algunas *funciones de usuario* (concepto que explicaremos más adelante), los comandos que las definen se suelen colocar antes de las instrucciones `open` o `nulldata` (ver ejemplo en figura 1, página 5)

3. Cómo crear y editar un guion

Un guion es un archivo de texto plano (ASCII), cuyo nombre lleva extensión .inp, que puede ser creado mediante cualquier editor de textos convencional (el mismo *bloc de notas* de Windows puede servir). Sin embargo, la interfaz gráfica de *Gretl* tiene un editor de texto que está especialmente diseñado para esto y, por eso, presenta algunas ventajas. Está disponible en el menú Archivo → Archivos de guion.



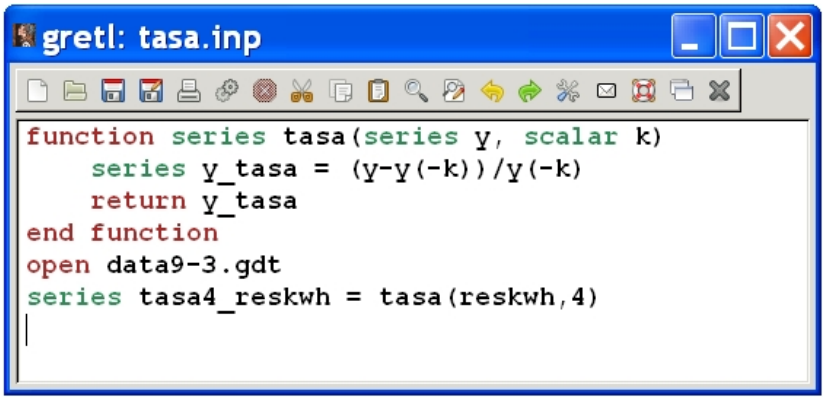
[6] La longitud máxima es $2^{14} = 16384$ bytes ≈ 16384 caracteres, que nadie en su sano juicio utilizaría.

En este menú tenemos tres opciones:

- Archivo de usuario... permite abrir y editar archivos de guion que el usuario haya creado y guardado previamente.
- Archivo de ejemplo... muestra una lista de guiones con diversos ejemplos, que se instalan por defecto al instalar *Gretl*.
- Nuevo guion aquí pulsando sobre Guion de Gretl se abre el editor de guiones con una página en blanco para que el usuario pueda crear el guion que desee[7].

Este editor, además de las características típicas de cualquier otro (copiar, pegar, guardar, etc), presenta la posibilidad de ejecutar el guion, de insertar un sangrado adecuado, de aplicar comentarios en bloque y la capacidad de resaltar en diferentes colores los comandos y funciones, lo cual facilita bastante su lectura. Puede verse un ejemplo de edición de un guion en la figura 1 (el ejemplo se comenta más adelante).

Figura 1:



4. Ejecución de un guion

Una vez que se ha escrito y guardado un archivo guion, las instrucciones que contiene se pueden ejecutar de dos maneras:

- Desde el propio editor de guiones (ver figura 1). Aquí hay tres formas de ejecución:

[7] También nos da la posibilidad de editar y procesar guiones escritos en otros lenguajes externos como Gnuplot, R, Octave y Python.

1. Pulsando sobre el icono de ruedas dentadas  cuando no hay nada seleccionado, se produce la ejecución del guion completo.
2. Si está seleccionado solo un subconjunto de líneas del guion, pulsando ese mismo icono se produce la ejecución de solamente esas líneas.
3. Por último, si se desea ejecutar una sola línea, es posible hacerlo como se señala en el punto 2 o también colocando el cursor en cualquier posición de esa línea y tecleando simultáneamente CTRL+↵ .

- Desde una terminal de texto, es decir, en Windows[8], desde el *símbolo del sistema* de Windows (Pulsando sobre Inicio ->Todos los programas ->Accesorios ->Símbolo del sistema) usando el programa ejecutable `gretlcli` (que en Windows suele estar en C:/Archivos de programa/gretl/gretlcli.exe). Por ejemplo, en la ventana de terminal ejecutaríamos

  ```
  gretlcli -b nombre_de_fichero.inp
  ```

  Esta segunda forma es más incómoda pero tiene la ventaja de que durante la ejecución se utiliza menos memoria, por lo que suele ser lo adecuado cuando nuestro guion incluye muchos cálculos (por ejemplo simulaciones) o estamos utilizando un conjunto de datos muy grande. También suele ser la forma de ejecución en ordenadores centrales de centros de computación que no tienen interfaz gráfica.

5. El conjunto de datos

En Goitisolo y Moral (2013, sección *Carga de datos: abrir o crear ficheros de datos*) se explica cómo se puede crear, usando la interfaz gráfica de *Gretl*, un archivo de datos en el formato nativo de *Gretl* *gdt*. También se describe cómo importar ficheros de otros formatos. En *Hansl*, para abrir o importar ficheros de datos se utiliza el comando `open`. Para abrir un fichero de datos ya existente, que está en el formato nativo de *Gretl*, utilizamos el comando:

```
open nombre-del-fichero.gdt
```

Por ejemplo, en el guion 1 hemos visto cómo, para abrir el fichero de datos data3-6.gdt que contiene las series de consumo y renta, se ha utilizado el comando:

```
open data3-6.gdt
```

Podemos usar también el comando `open` para importar ficheros de otros formatos. En particular, *Gretl* es capaz de reconocer los siguientes tipos de ficheros:

[8] En Linux, se puede utilizar cualquier programa de terminal para esto, como el Terminal de Gnome, Xterm, etc

Extensión	Descripción del tipo de fichero
`.gdtb`	Datos binarios de *Gretl* [1]
`.csv`	Datos ASCII separados por comas
`.txt`	Datos ASCII
`.gnumeric`	Hoja de cálculo Gnumeric
`.ods`	Hoja de cálculo *Open Document*
`.xls` y `.xlsx`	Hoja de cálculo Excel
`.dta`	Datos de Stata
`.wf1`	Datos de Eviews
`.sav`	Datos de SPSS
`.xpt`	Datos en el formato xport de SAS
`.m`	Datos de Octave
`.dat`	Datos de JMulTi

Al importar un fichero de datos ASCII (texto plano), tanto con extensión .txt como .csv, hay que tener en cuenta que, por defecto, un fichero de este tipo se considera un conjunto de datos de corte transversal. Para que sea interpretado, por ejemplo, como una serie temporal y así estén disponibles una serie de características especiales para series temporales, es necesario darle esta información precisando cuál es la periodicidad estacional (cuántas observaciones se recogen por año) y cuál es el periodo al que corresponde la primera observación. Esto se hace con la instrucción `setobs`, por ejemplo, así:

```
open fichero.txt
setobs 4 1981:1 --time-series
```

Cuando se guarda un fichero en el formato gdt queda registrada en el propio fichero la información sobre la estructura de los datos y simplemente con la instrucción `open fichero.gdt` el programa ya sabe si lo que tiene son series temporales.

En el siguiente guion se pide a *Gretl* que lea los datos de un fichero y muestre las observaciones de las variables que contiene:

Guion 2:

```
open data3-6.gdt
print --byobs
```

La clave `--byobs` indica que los datos han de presentarse junto al índice de las observaciones. Esto es especialmente conveniente cuando tenemos datos de series temporales,

[1] Un formato especial para series de muchas observaciones.

ya que el índice es la fecha de cada observación. El comando `print --byobs` es práctico cuando sabemos que el fichero contiene pocas variables, cuando tenemos muchas el formateo en la ventana de resultados impide ver correctamente la estructura de las variables. En ese caso es más práctico averiguar primero las variables que nos interesan y representarlas solo a ellas así:

```
print Ct --byobs
```

Al ejecutar el guion 2, obtenemos el resultado que aparece en el cuadro 1.

Cuadro 1:

```
gretl versión 1.9.90
Sesión actual: 2014-07-11 13:19

? open data3-6.gdt

Leer fichero de datos data3-6.gdt
periodicidad: 1, máx. obs: 36
rango de observaciones: 1959 a 1994

Listando 3 variables:
  0) const    1) Ct       2) Yt

? print --byobs

                Ct           Yt

1959          7876         8641
1960          7926         8660
1961          7954         8794
...
(se omiten líneas para ahorrar espacio)
...
1993         16810        18075
1994         17152        18320
```

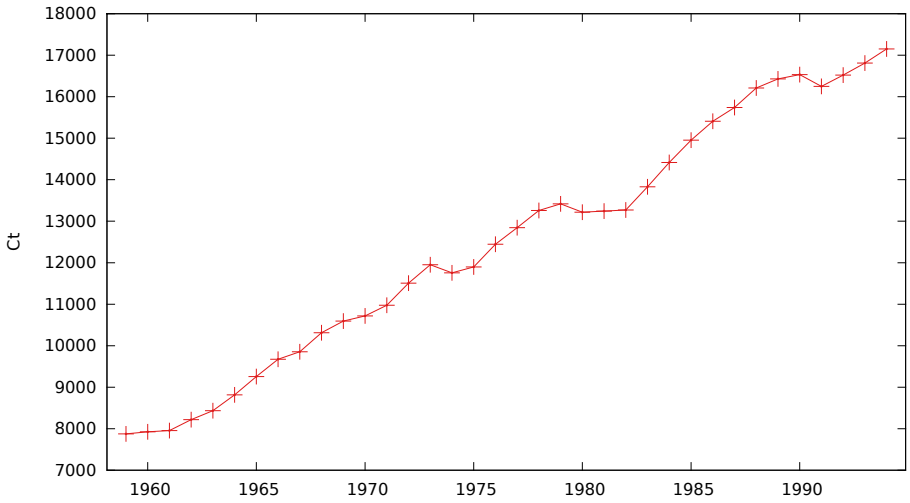
Si deseamos representar gráficamente la serie de consumo, el guion podría ser

```
open data3-6.gdt
gnuplot Ct --time-series --with-lp --output=display
```

La clave `--time-series` indica que la variable `Ct` ha de representarse respecto al tiempo, `--with-lp` indica que deben mostrarse marcas para las observaciones (por

defecto la series temporales se muestran sin marcas y con las observaciones ligadas mediante líneas). La clave `--output=display` indica que si el guion se ejecuta desde la interfaz gráfica, el resultado debe mostrarse en pantalla en una nueva ventana (por defecto los gráficos no se muestran en pantalla sino que se guardan en un fichero de formato `plt` de Gnuplot[9]). Alternativamente, podríamos poner por ejemplo `--output=nombre-de-fichero.pdf` para que el gráfico quede guardado en un fichero de formato `pdf` (también se admiten los formatos: `png`, `eps`, `emf`, `fig` y `svg`). Al ejecutar este guion se obtiene el gráfico de la figura 2.

Figura 2:



6. Conceptos básicos en Hansl

6.1. Tipos de objetos

Gretl y *Hansl* reconocen varios tipos diferentes de objetos (escribimos entre paréntesis la expresión en inglés que hay que poner en los comandos que los utilizan): escalar (`scalar`), serie (`series`), matriz (`matrix`), lista (`list`), cadena de caracteres (`string`), contenedor (`bundle`) y formación (`array`) (este último se ha incorporado en la versión 1.9.91 de *Gretl*).

[9] En la ventana de resultados se indica la ruta en la que se encuentra y el nombre del fichero que se ha asignado al gráfico

Tipo de objeto	Definición
`scalar`:	Un único valor numérico.
`series`:	Un conjunto de n valores numéricos, donde n es el número de observaciones del conjunto de datos actual.
`matrix`:	Una estructura rectangular de valores numéricos.
`list`:	Una lista de series. En concreto el objeto contiene los números de identificación (ID) del conjunto de series.
`string`:	Una cadena de caracteres.
`bundle`:	Un contenedor, que puede contener un número variable de objetos de cualquiera de estos seis tipos (incluyendo bundles).
`array`:	Un contenedor, que puede tener cero o más objetos del mismo tipo (strings, matrices, bundles o listas)

Las observaciones de una variable vienen recogidas por defecto en un objeto de tipo `series`. Cuando hablamos de series **no nos referimos necesariamente a series temporales**. El conjunto de observaciones de sección cruzada de una determinada variable, en la terminología de *Gretl*/*Hansl* se considera también una **serie**.

6.2. Comandos o instrucciones de Hansl

Como ya se ha mencionado, el lenguaje de programación *Hansl* está formado por un conjunto de comandos que sirven para pedir al ordenador que realice diferentes acciones, relacionadas con cálculos de tipo econométrico. La *Guía de Instrucciones de* Gretl (Cottrel y Lucchetti, 2013) presenta la definición y explicación de cada uno de los comandos. En el Apéndice: Lista de comandos de *Hansl* del presente documento se muestra un listado de los comandos y sus descripciones, clasificados por temas.

Casos especiales

Para crear un nuevo objeto de tipo serie podemos utilizar el comando

```
series nombre = (función)
```

Para crear un escalar utilizaremos

```
scalar nombre = (función)
```

y, de forma similar, podemos definir matrices, listas, cadenas de caracteres, contenedores o formaciones. Como se puede apreciar, los comandos `series`, `scalar`, etc. no están en el listado del Apéndice. Estos son en realidad diferentes variantes de un comando más general, de nombre `genr`. Así que para consultar en la *Guía de Instrucciones* más detalles sobre estos comandos, es necesario consultar la información correspondiente a `genr`.

6.3. El historial de instrucciones

Si el usuario ya ha practicado con *Gretl* realizando cálculos, gráficos y estimaciones por medio de la interfaz gráfica (ventanas y menús), ha de saber ahora que, salvo casos muy especiales, todas las transformaciones de variables y las estimaciones que se realizan de esta manera tienen su correspondiente contrapartida mediante un comando de *Hansl* y, al ejecutarse mediante la interfaz gráfica, ese comando queda reflejado en un fichero, que denominamos *historial de comandos*. Dicho fichero contiene una lista con todos los comandos que se hayan ejecutado hasta ese momento.

Así que una manera sencilla de empezar a programar en *Hansl* es, mediante la interfaz gráfica, abrir un fichero y realizar algunas transformaciones o cálculos y luego ir al menú Herramientas Historial de instrucciones para revisar cómo se realizan mediante comandos.

Por ejemplo, supongamos que abrimos *Gretl* y mediante la interfaz gráfica abrimos el fichero data3-6.gdt, que queremos realizar la regresión de `Yt` sobre una constante y `Ct` y que, para tener en cuenta la autocorrelación de las perturbaciones, utilizamos el método de mínimos cuadrados generalizados factibles de Cochrane-Orkutt. Una vez abierto el fichero de datos esto se hace pulsando sobre el menú Modelo Series temporales ▶ AR(1) y dejando allí el método de estimación que está señalado por defecto.

Una vez realizada la estimación, pulsando sobre Herramientas Historial de instrucciones podremos ver el siguiente listado:

```
# Comienza el historial 2014-09-18 13:55
# Registro de las instrucciones de la sesión.  Por favor, tenga en
# cuenta que probablemente necesitará editar este guion antes
# de ejecutarlo.
open (ruta)/data3-6.gdt
# modelo 1
ar1 Ct const Yt
```

La ventana del historial **no es editable ni ejecutable**. El historial se rellena automáticamente y el usuario no puede modificarlo, salvo por la ejecución de comandos vía menú. Si se quieren ejecutar los comandos de ese historial, es necesario seleccionarlos con el ratón, copiarlos al portapapeles mediante `Crtl+C` o pulsando sobre el icono de copiar, crear un nuevo guion de *Gretl* pulsando sobre Archivo->Archivos de guion->Nuevo guion->Guion de Gretl y, una vez abierto el editor de guiones, pegar allí el contenido del portapapeles mediante `Ctrl+V` o pulsando sobre el icono de pegar. Allí sí es posible editar y ejecutar esos comandos.

6.4. Los guiones de ejemplo

Cuando se instala *Gretl*, se instalan también, por defecto, un conjunto de guiones de ejemplo que están disponibles mediante el menú Archivo Archivos de guion ▶ Archivo de ejemplo... Al pulsar sobre Archivo de ejemplo... se abre una ventana que permite acceder a tres colecciones de guiones distribuidas en tres solapas con títulos: Gretl, Greene y Ramanathan. La solapa *Gretl* contiene un listado de guiones preparados para demostrar características específicas de *Gretl*; las solapas de nombres *Greene* y *Ramanathan* contienen ejemplos de ejercicios de los libros de texto de Greene (2003) y Ramanathan (2002).

Para aprender a programar en *Hansl*, es muy recomendable echar un vistazo a estos guiones.

6.5. Funciones

Para crear nuevos objetos mediante `genr` o cualquiera de sus variantes, se pueden utilizar (a la derecha del signo =) varios tipos de *funciones*:

1. **Funciones nativas de *Hansl***. Las hay de dos tipos (puede consultarse el listado completo en la *Guía de funciones*):

   1.1 **Accesores**: son funciones cuyo nombre va precedido del símbolo $ y que dan acceso al contenido de variables internas u ocultas que *Hansl* puede tener en memoria.

   Por ejemplo, `$nobs` contiene el número de observaciones en la muestra actual, `$pd` la periodicidad estacional y, después de realizar una estimación, `$coeff` contiene los coeficientes estimados, `$uhat` la serie de residuos, etc. El comando `varlist --accessors` sirve para presentar el listado de este tipo de variables que en el momento de invocarlo tienen asignado algún valor.

   1.2 **Funciones estándar**: son las funciones matemáticas, funciones de manejo de cadenas de caracteres, etc. Su estructura concreta depende del tipo de objeto que generan.

   Por ejemplo, `abs` es una función mediante la cual se obtiene el valor absoluto del objeto al que se aplica, que puede ser un escalar, una serie o una matriz.

2. **Funciones de usuario**: las define el propio usuario mediante el entorno

   ```
   function
   ...(comandos)
   end function.
   ```

   Se explican de manera detallada en la sección 7.

6.6. Dos ejemplos sencillos con Hansl

Ejemplo 1

El fichero `alturas.gdt` contiene los datos sobre alturas de 10 individuos y de sus padres del ejemplo utilizado en Goitisolo y Moral (2013, Sección *Carga de datos: abrir o crear ficheros de datos*). Crearemos un guion de *Hansl* para:

1) Cargar estos datos.

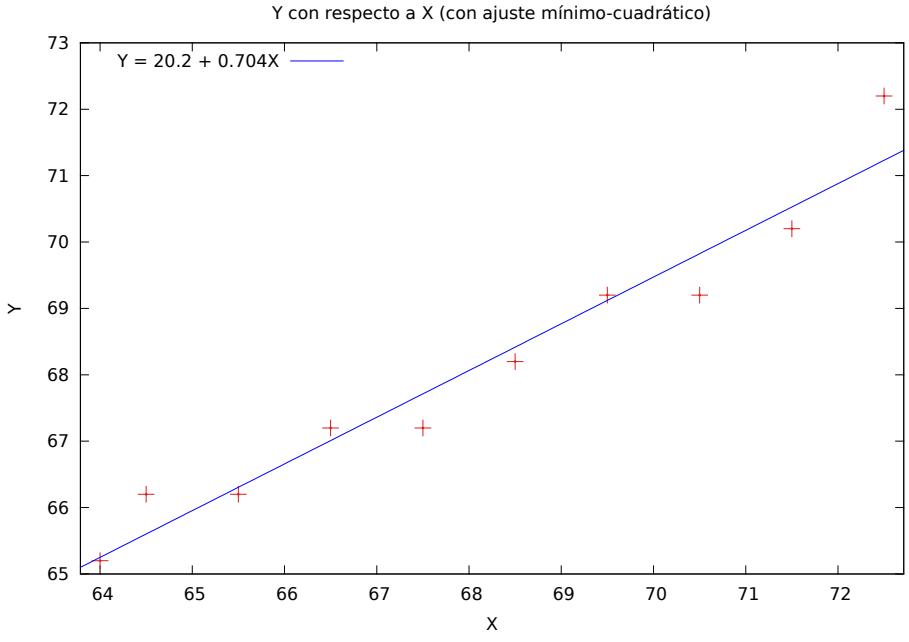
2) Realizar un gráfico de la variable Y (altura de los individuos), respecto a X (altura de los padres).

3) Calcular la regresión mínimo cuadrática ordinaria de Y sobre X.

El guion puede tener esta estructura:

```
open alturas.gdt
gnuplot Y X --output=display
ols Y const X
```

La ejecución de este programa da los siguientes resultados:



```
? model1 <- ols Y X const

model1: MCO, usando las observaciones 1-10
Variable dependiente: Y

             Coeficiente   Desv. Típica   Estadístico t   Valor p
  ------------------------------------------------------------------
  const       20.2105       4.00054          5.052       0.0010   ***
  X            0.703739     0.0587387       11.98        2.17e-06 ***

Media de la vble. dep.  68.10000   D.T. de la vble. dep.   2.131770
Suma de cuad. residuos  2.159156   D.T. de la regresión    0.519514
R-cuadrado              0.947209   R-cuadrado corregido    0.940610
F(1, 8)                 143.5407   Valor p (de F)          2.17e-06
Log-verosimilitud      -6.525048   Criterio de Akaike      17.05010
Criterio de Schwarz     17.65527   Crit. de Hannan-Quinn   16.38622

model1 guardado
```

Si el programa se ejecuta desde el editor de guiones de la interfaz gráfica de *Gretl*, puede ser conveniente asignar un nombre al gráfico y al modelo de esta forma:

```
open alturas.gdt
graph1 <- gnuplot Y X --output=display
model1 <- ols Y const X
```

ya que así el modelo y el gráfico quedarán guardados en la *Vista de iconos de sesión*  y serán más fácilmente accesibles para verlos de nuevo, analizar los residuos o realizar otros cálculos posteriores.

Ejemplo 2

La serie Y contiene $n = 45$ observaciones individuales de un índice que mide el estado de depresión de los individuos, cuyos valores van desde 0 hasta 20 y mayor índice significa mayor depresión (Fuente de datos: El Epicentro (2007)):

Y = { 2, 5, 6, 8, 8, 9, 9, 10, 11, 11, 11, 13, 13, 14, 14, 14, 14, 14, 14, 15, 15, 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20 }

Suponiendo una distribución normal en la variable Y y utilizando como predictor la media aritmética, la tarea que proponemos consiste en realizar una predicción por intervalo a un nivel de confianza del 95 %.

Recordemos que si $Y \sim N(\mu, \sigma^2)$ entonces $\bar{Y} \sim N(\mu, \frac{\sigma^2}{n})$ y si $z_{\alpha/2}$ es el valor que en la distribución normal deja a su derecha una probabilidad de $\alpha/2$,

$$\Pr\left(-z_{\alpha/2} < \frac{\bar{Y} - \mu}{\sigma/\sqrt{n}} < z_{\alpha/2}\right) = 1 - \alpha$$

$$\Pr\left(\bar{Y} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \bar{Y} + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}\right) = 1 - \alpha$$

Como σ es desconocida, la sustituimos por la estimación obtenida a través del estimador muestral $S = \sqrt{\sum_{i=1}^{n-1} \frac{(Y_i - \bar{Y})^2}{n-1}}$ y entonces tenemos

$$\Pr\left(\bar{Y} - t(n-1)_{\alpha/2}\frac{S}{\sqrt{n}} < \mu < \bar{Y} + t(n-1)_{\alpha/2}\frac{S}{\sqrt{n}}\right) = 1 - \alpha$$

donde $t(n-1)_{\alpha/2}$ es el valor que en la distribución t de Student de $n-1$ grados de libertad deja a su derecha una probabilidad de $\alpha/2$.

Suponiendo que los datos ya se han incluido en el fichero `depresion.gdt`, veamos cómo calcular en *Hansl* los límites de este intervalo.

Guion 3:

```
#1: Abrimos el fichero de datos
open depresion.gdt
#2: Calculamos la media de la serie Y
scalar mY = mean(Y)
#3: Calculamos la desviación típica muestral de la serie Y
scalar sY = sd(Y)
#4: Consultamos en la distribución t-de-Student de n-1
# grados de libertad cuál es el valor crítico que deja
# a su derecha una probabilidad de 0.025
scalar ta2 = critical(t,$nobs-1,0.025)
#5: Construimos una matriz (1x2) que contiene el límite
#inferior y el límite superior del intervalo pedido.
matrix intervalo = {mY - ta2*sY/sqrt($nobs), \
   mY + ta2*sY/sqrt($nobs) }
#6: Pedimos que se nos muestre la matriz
print intervalo
```

Los resultados de estos cálculos son (Utilizamos `[...]` para representar las primeras líneas de la ventana de resultados, que son similares a las del cuadro 1):

```
[...]
? scalar mY = mean(Y)
Se ha generado el escalar mY = 14.5556
#3: Calculamos la desviación típica muestral de la serie Y
? scalar sY = sd(Y)
Se ha generado el escalar sY = 4.32517
#4: Consultamos en la distribución t-de-Student de N-1
# grados de libertad cuál es el valor crítico que deja
# a su derecha una probabilidad de 0.025
? scalar ta2 = critical(t,$nobs-1,0.025)
Se ha generado el escalar ta2 = 2.01537
#5: Construimos un matriz (1x2) que contiene el límite
#inferior y el límite superior del intervalo pedido.
? matrix intervalo = {mY - ta2*sY/sqrt($nobs), mY + ta2*sY/sqrt($nobs)}
Se ha generado la matriz intervalo
#6: pedimos que se muestre la matriz
? print intervalo
intervalo (1 x 2)

      13.256       15.855
```

7. Funciones de usuario

La posibilidad de que el usuario cree sus propias funciones hace que el lenguaje de programación *Hansl* sea muy flexible y al mismo tiempo que los programas creados con este lenguaje tengan una estructura muy compacta. Como ya se ha indicado, las funciones de usuario se crean mediante el entorno `function...end function`.

Mediante el comando `function` se determina la estructura de la función que queremos programar. El comando se usa de la siguiente forma:

```
function tipo-de-resultado nombre-de-la-función (tipo parámetro,
tipo parámetro, ...)
```

El `tipo-de-resultado` determina el tipo de objeto que la función generará, que puede ser: `void` (si la función no genera nada), `scalar`, `series`, `matrix`, `list`, `string` o `bundle`. Luego, se establece el nombre de la función y después, entre paréntesis, se colocan los parámetros de la función, que normalmente contienen la información que desde el programa ha de pasarse a la parte interna de la función para que ésta pueda desarrollar sus cálculos u operaciones, o dicho de otra forma, los *inputs* de la función. Cada parámetro ha de ir precedido de una etiqueta que especifica el tipo de parámetro. En las líneas siguientes se colocarán los comandos específicos que desarrollan el cálculo que queremos que la función realice y, para finalizar, la función normalmente

concluirá (excepto en el caso de una función que no genere nada) con un comando `return` que indica cuál es el objeto que la función debe producir como resultado.

Veamos un ejemplo de una función sencilla. Dada una serie temporal y_t, se define la tasa de variación relativa de orden k, $\dot{y}_t(k)$ como:

$$\dot{y}_t(k) \equiv \frac{y_t - y_{t-k}}{y_{t-k}} \tag{1}$$

Hansl no tiene una función específica para calcular una tasa de variación. En este guion se construye una función que la calcula:

Guion 4:

```
function series tasa(series y, scalar k)
    series y_tasa=(y-y(-k))/y(-k)
    return y_tasa
end function
```

Estudiemos línea por línea el código de esta función. En la primera línea,

```
function series tasa(series y, scalar k)
```

se establece que la función generará como resultado una serie, que el nombre de la función es `tasa` y que esta función requiere dos parámetros de entrada: una serie `y` y un valor escalar `k`.

La línea siguiente,

```
series y_tasa=(y-y(-k))/y(-k)
```

contiene la instrucción en la que realmente se realiza el cálculo de la tasa, como en la ecuación (1), situando en la serie `y_tasa` el resultado.

La última línea,

```
return y_tasa
```

indica que la función ha de devolver como resultado la serie `y_tasa`. Por último, la linea

```
end function
```

indica el cierre del entorno de definición de la función.

Para ejecutar esta función es necesario primero que *Gretl* la lea, para ello en el editor de guiones pulsaremos sobre el botón  (Ver sección 4, *Ejecución de un guion*). Al ejecutar la definición de una función, *Gretl* no produce ningún resultado, solo reproduce sus líneas en la ventana de resultados y queda cargada en memoria. Luego, es necesario abrir un fichero de datos y después ejecutar, tal y como indica la función, una instrucción que genere una serie, introduciendo la función `tasa` con dos parámetros, por ejemplo, de esta forma:

```
series tasa1_x = tasa(x,1)
```

donde `x` es una serie ya existente en el conjunto de datos activo. Mediante esta instrucción se crea una nueva serie con nombre `tasa1_x` que contiene la tasa de orden uno de la serie temporal `x`, es decir, $\dot{x}_t(1) = \frac{x_t - x_{t-1}}{x_{t-1}}$.

Si nuestro conjunto de datos contiene datos trimestrales, también puede ser interesante calcular la tasa interanual, es decir, la tasa de orden cuatro. Esto se haría escribiendo la segunda instrucción así:

```
series tasa4_x = tasa(x,4)
```

Pero cuando se programa en *Hansl* lo normal es poner todo el código de la función y del comando que la ejecuta en un solo guion y ejecutar el fichero completo, por ejemplo así:

```
#1: Definimos la función
function series tasa(series y, scalar k)
    series y_tasa=(y-y(-k))/y(-k)
    return y_tasa
end function
#2: Ejecutamos la función
open data9-3.gdt
series tasa1_reskwh = tasa(reskwh,1)
series tasa4_reskwh = tasa(reskwh,4)
print reskwh tasa1_reskwh tasa4_reskwh --byobs
```

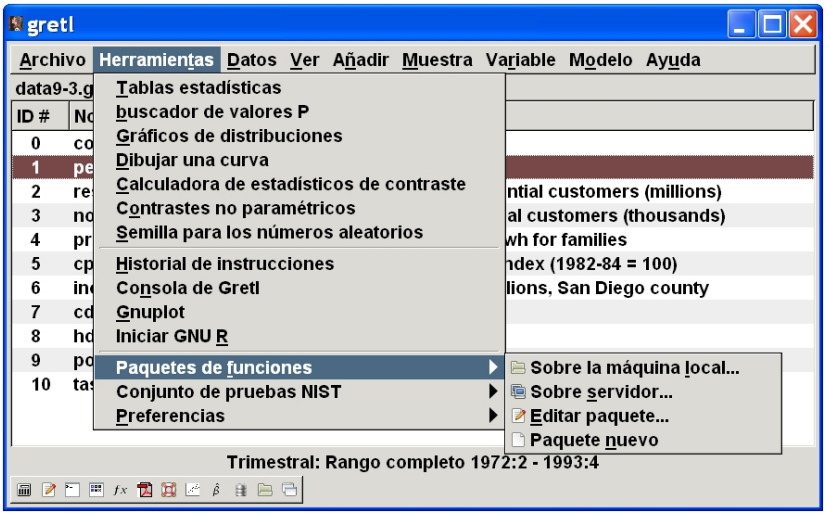
Al ejecutar este guion obtenemos como resultado:

```
[...]
? series tasa1_reskwh = tasa(reskwh,1)
Se ha generado la serie tasa1_reskwh (ID 10)
? series tasa4_reskwh = tasa(reskwh,4)
Se ha generado la serie tasa4_reskwh (ID 11)
? print reskwh tasa1_reskwh tasa4_reskwh --byobs

             reskwh tasa1_reskwh tasa4_reskwh

1972:2       586.608
1972:3       625.797      0.06681
1972:4       704.154    0.1252115
1973:1       817.206    0.1605501
1973:2       667.642   -0.1830187    0.1381400
1973:3       661.827    -0.008710      0.05757
1973:4       732.839    0.1072969      0.04074
1974:1       796.876      0.08738     -0.02488
...
```

8. Paquetes de funciones

Además de poderse programar funciones por parte del usuario, *Gretl* y *Hansl* ofrecen la posibilidad de empaquetar varias funciones conjuntamente y de acceder a la ejecución de las funciones desde la interfaz gráfica de *Gretl* usando ventanas y menús. Esto se hace mediante lo que se denominan **paquetes de funciones**.

Desde la interfaz gráfica de *Gretl* se accede a ellos mediante el menú Herramientas Paquetes de funciones.



Desde allí se accede a cuatro elementos:

- Sobre la máquina local... Da acceso a los paquetes de funciones que el usuario tiene ya instalados en su sistema.
- Sobre servidor... Realiza una conexión con el servidor central de *Gretl* y se accede a un listado con paquetes de funciones que los usuarios han puesto a disposición del público y que están disponibles para que cualquiera los pueda instalar en su sistema (en este momento son 77 paquetes).
- Editar paquete... Permite la modificación de un paquete que el usuario tenga ya instalado en su sistema.
- Paquete nuevo Da acceso a una interfaz que permite la creación de un nuevo paquete de funciones.

Veamos ahora cómo crear un nuevo paquete de funciones. En primer lugar, es necesario crear las funciones en *Hansl*, por ejemplo desde un archivo de guion. Supongamos que queremos crear un paquete de funciones que contenga la función que hemos creado antes, que está en el guion 4 (página 17).

Como vamos a crear un paquete que se ejecutará desde los menús, conviene cambiar la primera línea de la función que hemos propuesto antes, añadiendo para cada parámetro, entre comillas, una explicación más detallada. Esta información ayuda a que el usuario entienda mejor lo que debe seleccionar como *input*, ya que que se muestra junto a cada parámetro en la ventana de ejecución. Por ejemplo:

```
function series tasa(series y "Variable a transformar", scalar k "Orden")
```

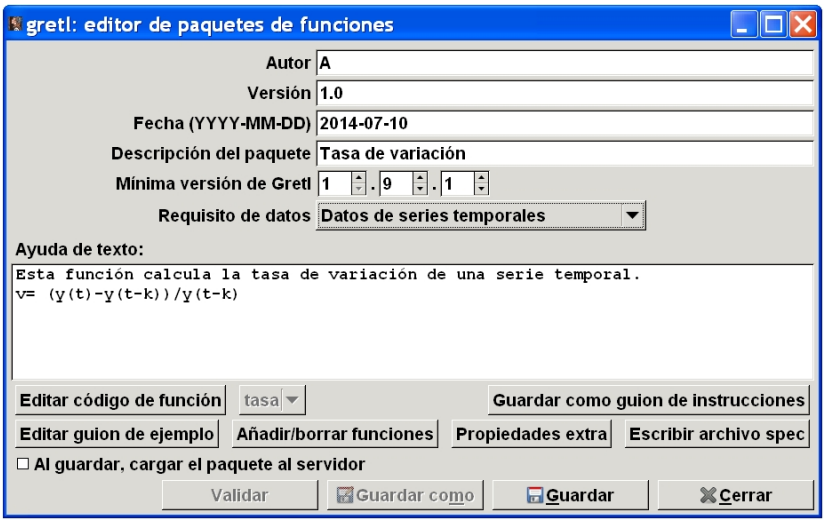
(Puede verse la correspondiente ventana de ejecución más adelante, en la figura 4, página 24,).

A continuación, hay que ejecutar el guión desde la interfaz gráfica de *Gretl* para que las funciones queden cargadas en memoria. Cuando ya se han cargado, al pulsar sobre Herramientas Paquetes de funciones Paquete nuevo el programa detecta si hay funciones cargadas en memoria y ofrece la posibilidad de empaquetarlas. Se abre entonces una ventana de diálogo que, en su parte izquierda, muestra todas las funciones cargadas en memoria y nos pide que seleccionemos de entre ellas, cuáles queremos marcar como *funciones públicas* y cuáles como *funciones de ayuda* o *auxiliares*:

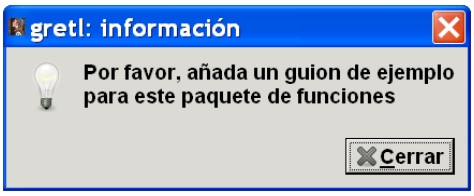
Una función es *pública* si se va permitir al usuario que introduzca los valores de sus parámetros. Una función es "*de ayuda*" (o *auxiliar*) cuando los valores de sus parámetros no los introduce el usuario sino que son transmitidos desde otra función y devuelve sus resultados a esa función. Todo paquete contendrá, al menos, una función pública. En nuestro ejemplo solo hay una, que obviamente debe empaquetarse como función pública; si hubiera más, alguna de ellas podría ser pública y otras podrían ser auxiliares.

Una vez determinadas las funciones *públicas* y las *auxiliares*, pulsando sobre el botón Aceptar se accede al editor de paquetes:

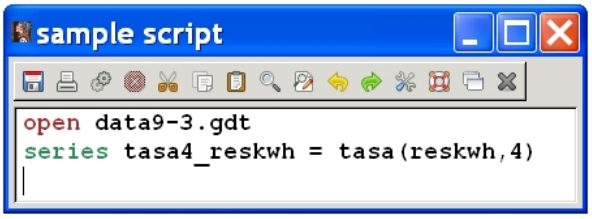
Figura 3:

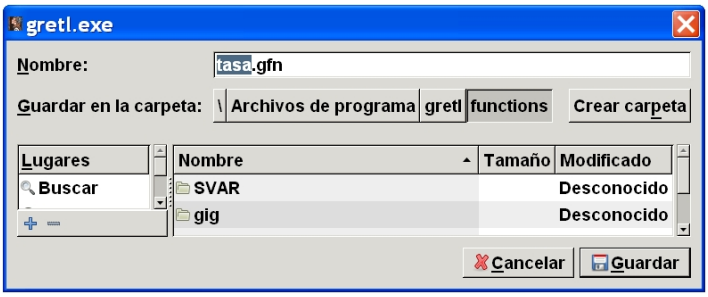


Desde este editor se puede Editar el código de una función (o funciones) y preparar varias otras cosas para completar el paquete. Entre ellas: se puede Editar el guion de ejemplo, o mejor dicho se debe, ya que es obligatorio que todo paquete incluya un guion de ejemplo. Si intentamos guardar el paquete sin haber creado un ejemplo, obtendremos el siguiente mensaje

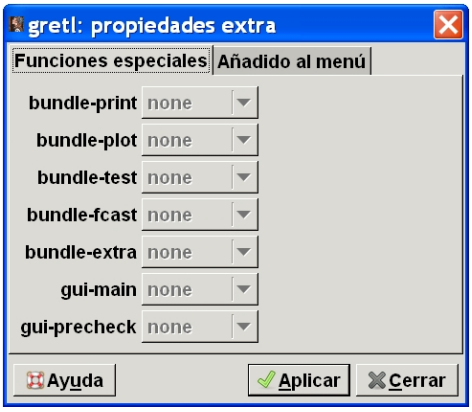


Una vez que hayamos creado un pequeño guión de ejemplo, ya se nos permitirá guardar el paquete. En el momento de guardarlo se nos preguntará el nombre, que por defecto es el nombre de la función pública, con extensión .gfn.



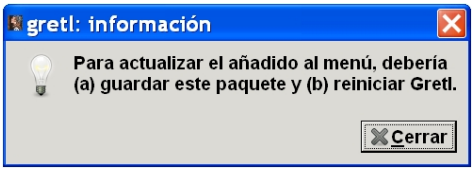


Una vez guardado el paquete, también se puede indicar si queremos **que la función quede incorporada a los menús**. Pulsando sobre el botón Propiedades extra del editor de funciones, se accede a una pestaña con **Funciones especiales**, que permiten configurar una serie de opciones para las estructuras de contenedores (`bundles`). Son funciones avanzadas que aquí no comentaremos ya que su uso requiere un conocimiento avanzado de *Hansl*, que excede el nivel pretendido en este documento.

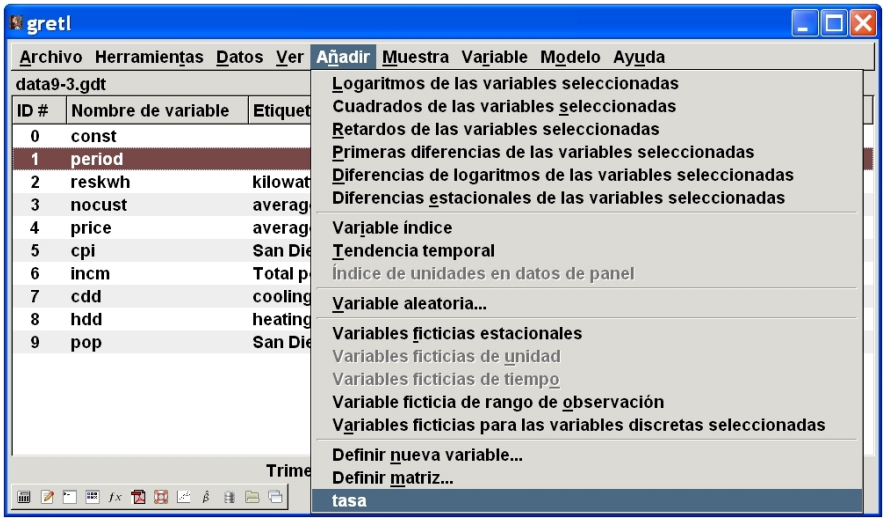


También se da acceso a una segunda pestaña que lleva por nombre Añadido al menú y que permite situar en los menús de *Gretl* un elemento para, al pulsar sobre él, ejecutar el nuevo paquete. Al activar esa pestaña, se abre el siguiente cuadro de diálogo:

En el campo Etiqueta que aparece en este cuadro hay que especificar el nombre que queremos que aparezca en el menú. Además, es posible elegir que la función aparezca en la ventana principal o en la ventana de estimación de un modelo. Por ejemplo, en nuestro caso hemos puesto la etiqueta `tasa` y lo hemos incluido en el menú Añadir de la ventana principal. Al pulsar sobre el botón Aplicar nos aparece un mensaje que indica que para que el elemento se añada al menú hay que guardar el paquete (pulsando sobre el botón Guardar del editor de paquetes) y reiniciar *Gretl*.



La próxima vez que iniciemos *Gretl* podremos ver que el nuevo elemento ha quedado incorporado en el menú en el lugar que hemos indicado:

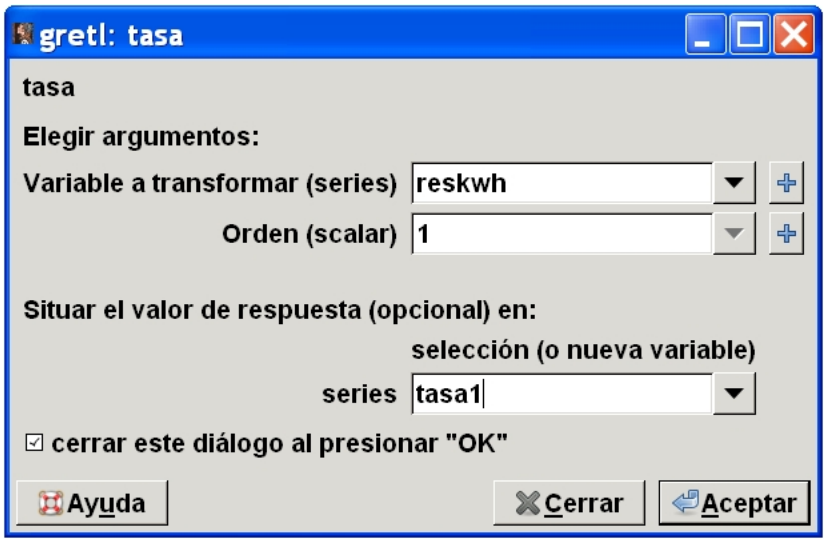


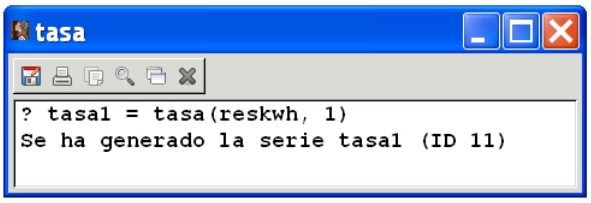
En nuestro ejemplo, al abrir el fichero `data9-3.gdt` con datos de series temporales trimestrales y ejecutar Añadir ▶ tasa nos aparece la ventana de la figura 4. En esta ventana el usuario tiene que elegir una Variable a transformar, es decir, la variable sobre la que se calculará la tasa y el Orden de la tasa que desea.

En la parte de abajo de este cuadro hay un campo[10] Situar el valor de respuesta (opcional) en: en el que se puede seleccionar en dónde se guardará la nueva serie que se va a crear. Es posible sobreescribir una serie ya existente, o dar un nuevo nombre, para que se cree una serie nueva. Después de pulsar sobre el botón Aceptar se realiza el cálculo y, en este caso, se añade la nueva variable *tasa1* al conjunto de datos actual de *Gretl*.

[10] En la versión 1.9.91 se ha mejorado esta traducción para que quede más clara, *Poner el resultado en : (es opcional)*

Figura 4:





Desde este momento, esa nueva variable ya está disponible para visualizarla gráficamente o trabajar con ella en otros nuevos cálculos.

8.1. Paquetes de funciones que publican los usuarios

Si un usuario ha construido un hermoso paquete de funciones, está orgulloso de él y desea compartirlo con otros usuarios, puede hacerlo mediante el editor de paquetes. En la figura 3 en la parte inferior izquierda hay una opción al guardar, cargar el paquete en el servidor. Si se activa esa marca, se intentará cargar el paquete en el servidor[11].

Los paquetes de funciones que han publicado los usuarios están disponibles Herramientas Paquetes de funciones ▶ Sobre servidor y también hay un listado explicativo en la wiki de *Gretl*: http://gretlwiki.econ.univpm.it

[11] La primera vez que un usuario hace esto, le aparece un cuadro de diálogo para crear un nombre de usuario y una clave en ese servidor.

Apéndice: Lista de comandos de Hansl

Presentamos aquí una lista de comandos clasificados por temas. El enlace de cada comando lleva a su definición en una página del servidor web de *Gretl*. Téngase en cuenta que la definición de la página web puede corresponder a una versión de *Gretl* posterior a la 1.9.90 en la que está basado el presente documento.

Estimación

ar	Estimación autorregresiva	ar1	Estimación AR(1)
arbond	Arellano-Bond	arch	Modelo ARCH
arima	Modelo ARIMA	biprobit	Probit bivariante
dpanel	Modelos de panel dinámicos	duration	Modelos de duración
equation	Define las ecuaciones de un sistema	estimate	Estima un sistema de ecuaciones
garch	Modelo GARCH	gmm	Estimación GMM
heckit	Modelo de selección de Heckman	hsk	Estimaciones corregidas de heterocedasticidad
intreg	Modelo de regresión por intervalos	kalman	Filtro de Kalman
lad	Estimación por mínima desviación absoluta	logistic	Regresión logística
logit	Regresión Logit	mle	Estimación por máxima verosimilitud
mpols	MCO de precisión múltiple	negbin	Regresión binomial negativa
nls	Mínimos cuadrados no lineales	ols	Mínimos cuadrados ordinarios
panel	Modelos de panel	poisson	Estimación de Poisson
probit	Modelo Probit	quantreg	Regresión de cuantil
system	Sistemas de equaciones	tobit	Modelo Tobit
tsls	Regresión por variables instrumentales	var	Autorregresión vectorial
vecm	Modelo de Corrección de Error vectorial	wls	Mínimos cuadrados ponderados

Contrastes

add	Añadir variables al modelo	adf	Contraste aumentado de Dickey-Fuller
chow	Contraste de Chow	coeffsum	Suma de coeficientes
coint	Contraste de cointegración de Engle-Granger	coint2	Contraste de cointegración de Johansen
cusum	Contraste CUSUM	difftest	Contrastes no paramétricos para la diferencia entre poblaciones
hausman	Diagnósticos de panel	kpss	Contraste de estacionariedad KPSS
leverage	Observaciones influyentes	levinlin	Contraste Levin-Lin-Chu

meantest	Diferencia de medias	modtest	Contrastes de un modelo
normtest	Contraste de normalidad	omit	Omisión de variables
qlrtest	Contraste de razón de verosimilitudes de Quandt	reset	Contraste RESET de Ramsey
restrict	Contraste de restricciones	runs	Contraste de rachas
vartest	Diferencia de varianzas	vif	Factores de inflación de varianza

Transformaciones

diff	Primeras diferencias	discrete	Marcar la variable como discreta
dummify	Crear conjuntos de variables ficticias	lags	Crear retardos
ldiff	Diferencias de logaritmos	logs	Crear logaritmos
orthdev	Desviaciones ortogonales	sdiff	Diferencias estacionales
square	Cuadrados de las variables		

Estadísticos

anova	Análisis de varianza	corr	Coeficientes de correlación
corrgm	Correlograma	fractint	Integración fraccional
freq	Distribución de frecuencias	hurst	Exponente de Hurst
mahal	Distancias de Mahalanobis	pca	Análisis de componentes principales
pergm	Periodograma	spearman	Rango de correlación de Spearman
summary	Estadísticos descriptivos	xcorrgm	Correlograma cruzado
xtab	Tabulación cruzada de variables		

Conjunto de datos

append	Añadir datos	data	Importar desde una base de datos
dataset	Manipular el conjunto de datos	delete	Borrar variables
genr	Generar una nueva variable	info	Información sobre el conjunto de datos
join	Administrar fuentes de datos	labels	Etiquetas para las variables
markers	Etiquetas de las observaciones	nulldata	Crear un conjunto de datos en blanco
open	Abrir un fichero de datos	rename	Renombrar variables
setinfo	Editar los atributos de una variable	setmiss	Establecer código de valor ausente
setobs	Establecer la frecuencia y la primera observación	smpl	Establecer el rango muestral

`store`	Guardar datos	`varlist`	Ver el listado de las variables

Gráficos

`boxplot`	Gráficos de caja	`gnuplot`	Crear un gráfico de gnuplot
`graphpg`	Página de gráficos de Gretl	`qqplot`	Gráfico Q-Q
`rmplot`	Gráfico Rango-Media	`scatters`	Múltiples gráficos por parejas
`textplot`	Gráfico ASCII		

Presentación

`eqnprint`	Presentar un modelo como ecuación LaTeX	`modprint`	Presentar un modelo definido por el usuario
`outfile`	Dirigir los resultados a un fichero	`print`	Mostrar datos o cadenas de caracteres
`printf`	Mostrar información con formato	`sprintf`	Poner resultados en una cadena de caracteres
`tabprint`	Presentar un modelo en forma de tabla LaTeX		

Predicción

`fcast`	Generar predicciones

Programación

`break`	Cortar un bucle	`catch`	Capturar errores
`clear`	Vaciar	`debug`	Depuración
`elif`	Control de flujo	`else`	Control de flujo
`end`	Final de un bloque de comandos	`endif`	Control de flujo
`endloop`	Final de un bucle de comandos	`flush`	Procesando...
`foreign`	Inicia un guion en un otro lenguaje	`function`	Define una función
`if`	Control de flujo	`include`	Incluye funciones ya definidas
`loop`	Comenzar un bucle de comandos	`makepkg`	Crear un paquete de funciones
`run`	Ejecutar un guion	`set`	Establecer parámetros del programa
`setopt`	Establece las opciones para el siguiente comando		

Utilidades

`help`	Ayuda sobre los comandos	`modeltab`	Tabla de modelos
`pvalue`	Calcular valores p	`quit`	Salir del programa
`shell`	Ejecutar comandos de shell		

Referencias

Cottrel, A. and Lucchetti, R. (2013a), *Gretl User´s Guide*, disponible en http://gretl.sourceforge.net.

Cottrel, A. and Lucchetti, R. (2013b), *Gretl Command Reference*, disponible en http://gretl.sourceforge.net.

El Epicentro. Centro de Epidemiología (2007), *Epidemiología analítica. Intervalos de confianza*. Universidad Católica de Chile. Página web consultada el 18 de septiembre de 2014 en http://escuela.med.puc.cl/recursos/recepidem/epianal9.htm.

Goitisolo, B. and Moral, P. (2013), *Introducción a Gretl 1.9.12 para Econometría*, publicado en ADDI: http://hdl.handle.net/10810/10560.

Greene, W.H. (2003), *Econometric Analysis*, 5ª ed., Prentice Hall.

Ramanathan, R. (2002) *Introductory Econometrics with Applications*, 5ª ed., Harcourt College Publishers: Orlando, FL.