Gretl Command Reference

Gnu Regression, Econometrics and Time-series

Allin Cottrell
Department of Economics
Wake Forest university

November, 2005

Contents

Chapter 1

Gretl commands

1.1 Notation

The commands defined below may be executed in the command-line client program. They may also be placed in a "script" file for execution in the GUI, or entered using the latter's "console mode". In most cases the syntax given below also applies when you are presented with a line to type in a dialog box in the GUI (but see also gretl's online help), except that you should *not* type the initial command word — it is implicit from the context. One other difference is that some commands support option flags, but you cannot enter these in GUI dialog boxes; generally there are menu items which achieve the same effect.

The following conventions are used below:

- A `typewriter font` is used for material that you would type directly, and also for internal names of variables.
- Terms in a *slanted font* are place-holders: you should substitute some specific replacement. For example, you might type `income` in place of the generic *xvar*.
- The construction `[` *arg* `]` means that the argument *arg* is optional: you may supply it or not (but in any case don't type the brackets).
- The phrase "estimation command" means a command that generates estimates for a given model, for example `ols`, `ar` or `wls`.

Section and Chapter references below are to Ramanathan (2002).

1.2 Commands

add

Argument:	*varlist*
Options:	`--vcv` (print covariance matrix)
	`--quiet` (don't print estimates for augmented model)
	`--silent` (don't print anything)
	`--inst` (add as instrument, TSLS only)
	`--both` (add as both regressor and instrument, TSLS only)
Examples:	`add 5 7 9`
	`add xx yy zz --quiet`

Must be invoked after an estimation command. The variables in *varlist* are added to the previous model and the new model is estimated. A test statistic for the joint significance of the added variables is printed, along with its p-value. The test statistic is F in the case of OLS estimation, an asymptotic Wald chi-square value otherwise. A p-value below 0.05 means that the coefficients are jointly significant at the 5 percent level.

If the `--quiet` option is given the printed results are confined to the test for the joint significance of the added variables, otherwise the estimates for the augmented model are also printed. In the

latter case, the `--vcv` flag causes the covariance matrix for the coefficients to be printed also. If the `--silent` option is given, nothing is printed; nonetheless, the results of the test can be retrieved using the special variables `$test` and `$pvalue`.

If the original model was estimated using two-stage least squares, an ambiguity arises: should the new variables be added as regressors, as instruments, or as both? This is resolved as follows: by default the new variables are added as endogenous regressors, but if the `--inst` flag is given they are added as instruments, or if the `--both` flag is given they are added as exogenous regressors.

Menu path: Model window, /Tests/add variables

addobs

Argument: *nobs*
Example: `addobs 10`

Adds the specified number of extra observations to the end of the working dataset. This is primarily intended for forecasting purposes. The values of most variables over the additional range will be set to missing, but certain deterministic variables are recognized and extended, namely, a simple linear trend and periodic dummy variables.

This command is not available if the dataset is currently subsampled by selection of cases on some Boolean criterion.

Menu path: /Data/Add observations

addto

Arguments: *modelID varlist*
Option: `--quiet` (don't print estimates for augmented model)
Example: `addto 2 5 7 9`

Works like the `add` command, except that you specify a previous model (using its ID number, which is printed at the start of the model output) to take as the base for adding variables. The example above adds variables number 5, 7 and 9 to Model 2.

Menu path: Model window, /Tests/add variables

adf

Arguments: *order varname*
Options: `--nc` (test without a constant)
`--c` (with constant only)
`--ct` (with constant and trend)
`--ctt` (with constant, trend and trend squared)
`--verbose` (print regression results)
Examples:
```
adf 0 y
adf 2 y --nc --c --ct
```

Computes statistics for a set of Dickey–Fuller tests on the specified variable, the null hypothesis being that the variable has a unit root.

By default, three variants of the test are shown: one based on a regression containing a constant, one using a constant and linear trend, and one using a constant and a quadratic trend. You can control the variants that are presented by specifying one or more of the option flags.

In all cases the dependent variable is the first difference of the specified variable, y, and the key independent variable is the first lag of y. The model is constructed so that the coefficient on lagged

y equals 1 minus the root in question. For example, the model with a constant may be written as

$$(1 - L)y_t = \beta_0 + (1 - \alpha)y_{t-1} + \epsilon_t$$

If the lag order, k, is greater than 0, then k lags of the dependent variable are included on the right-hand side of each test regression.

If the given lag order is prefaced with a minus sign, it is taken as the maximum lag and the actual lag order used is obtained by testing down. Let the given order be $-k$: the testing-down algorithm is then:

1. Estimate the Dickey–Fuller regression with k lags of the dependent variable.
2. Is the last lag significant? If so, execute the test with lag order k. Otherwise, let $k = k - 1$; if $k = 0$, execute the test with lag order 0, else go to step 1.

In the context of step 2 above, "significant" means that the t-statistic for the last lag has an asymptotic two-sided p-value, against the normal distribution, of 0.10 or less.

P-values for the Dickey–Fuller tests are based on MacKinnon (1996). The relevant code is included by kind permission of the author.

Menu path: /Variable/Augmented Dickey-Fuller test

append

Argument: *datafile*

Opens a data file and appends the content to the current dataset, if the new data are compatible. The program will try to detect the format of the data file (native, plain text, CSV, Gnumeric, Excel, etc.).

Menu path: /File/Append data

ar

Arguments: *lags* ; *depvar indepvars*
Option: `--vcv` (print covariance matrix)
Example: `ar 1 3 4 ; y 0 x1 x2 x3`

Computes parameter estimates using the generalized Cochrane–Orcutt iterative procedure (see Section 9.5 of Ramanathan). Iteration is terminated when successive error sums of squares do not differ by more than 0.005 percent or after 20 iterations.

lags is a list of lags in the residuals, terminated by a semicolon. In the above example, the error term is specified as

$$u_t = \rho_1 u_{t-1} + \rho_3 u_{t-3} + \rho_4 u_{t-4} + e_t$$

Menu path: /Model/Time series/Autoregressive estimation

arch

Arguments: *order depvar indepvars*
Example: `arch 4 y 0 x1 x2 x3`

Tests the model for ARCH (Autoregressive Conditional Heteroskedasticity) of the specified lag order. If the LM test statistic has a p-value below 0.10, then ARCH estimation is also carried out. If the predicted variance of any observation in the auxiliary regression is not positive, then the corresponding squared residual is used instead. Weighted least squares estimation is then performed on the original model.

See also garch.

Menu path: Model window, /Tests/ARCH

arma

Arguments:	*p q* ; [*P Q* ;] *depvar* [*indepvars*]
Options:	`--native` (Use native plugin (default))
	`--x-12-arima` (use X-12-ARIMA for estimation)
	`--verbose` (print details of iterations)
	`--vcv` (print covariance matrix)
	`--nc` (do not include a constant)
Examples:	`arma 1 2 ; y`
	`arma 2 2 ; y 0 x1 x2 --verbose`
	`arma 1 1 ; 1 0 ; y 0 x1 x2`

If no *indepvars* list is given, estimates a univariate ARMA (Autoregressive, Moving Average) model. The integer values p and q represent the AR and MA orders respectively. The optional integer values P and Q represent seasonal AR and MA orders; these are relevant only if the data have a frequency greater than 1 (for example, quarterly or monthly data).

In the univariate case the default is to include an intercept in the model but this can be suppressed with the `--nc` flag. If *indepvars* are added, the model becomes ARMAX; in this case the constant should be included explicitly if you want an intercept (as in the second example above).

The default is to use the "native" gretl ARMA function; in the case of a univariate ARMA model X-12-ARIMA may be used instead (if the X-12-ARIMA package for gretl is installed).

The options given above may be combined, except that the covariance matrix is not available when estimation is by X-12-ARIMA.

The native gretl ARMA algorithm is largely due to Riccardo "Jack" Lucchetti. It uses a conditional maximum likelihood procedure, implemented via iterated least squares estimation of the outer product of the gradient (OPG) regression. See the *Gretl User's Guide* for the logic of the procedure. The AR coefficients (and those for any additional regressors) are initialized using an OLS auto-regression, and the MA coefficients are initialized at zero.

The AIC value given in connection with ARMA models is calculated according to the definition used in X-12-ARIMA, namely

$$\mathrm{AIC} = -2\ell + 2k$$

where ℓ is the log-likelihood and k is the total number of parameters estimated. The "frequency" figure printed in connection with AR and MA roots is the λ value that solves

$$z = re^{i2\pi\lambda}$$

where z is the root in question and r is its modulus.

Menu path: /Variable/ARMA model, /Model/Time series/ARMAX

Other access: Main window pop-up menu (single selection)

boxplot

Argument:	*varlist*
Option:	`--notches` (show 90 percent interval for median)

These plots (after Tukey and Chambers) display the distribution of a variable. The central box encloses the middle 50 percent of the data, i.e. it is bounded by the first and third quartiles. The "whiskers" extend to the minimum and maximum values. A line is drawn across the box at the median.

In the case of notched boxes, the notch shows the limits of an approximate 90 percent confidence interval for the median. This is obtained by the bootstrap method.

After each variable specified in the boxplot command, a parenthesized Boolean expression may be added, to limit the sample for the variable in question. A space must be inserted between the variable name or number and the expression. Suppose you have salary figures for men and women, and you have a dummy variable GENDER with value 1 for men and 0 for women. In that case you could draw comparative boxplots with the following *varlist*:

```
salary (GENDER=1) salary (GENDER=0)
```

Some details of gretl's boxplots can be controlled via a (plain text) file named `.boxplotrc`. For details on this see the *Gretl User's Guide*.

Menu path: /Data/Graph specified vars/Boxplots

break

Break out of a loop. This command can be used only within a loop; it causes command execution to break out of the current (innermost) loop. See also loop.

chow

Argument: *obs*
Examples: `chow 25`
`chow 1988:1`

Must follow an OLS regression. Creates a dummy variable which equals 1 from the split point specified by *obs* to the end of the sample, 0 otherwise, and also creates interaction terms between this dummy and the original independent variables. An augmented regression is run including these terms and an F statistic is calculated, taking the augmented regression as the unrestricted and the original as restricted. This statistic is appropriate for testing the null hypothesis of no structural break at the given split point.

Menu path: Model window, /Tests/Chow test

coeffsum

Argument: *varlist*
Example: `coeffsum xt xt_1 xr_2`

Must follow a regression. Calculates the sum of the coefficients on the variables in *varlist*. Prints this sum along with its standard error and the p-value for the null hypothesis that the sum is zero.

Note the difference between this and omit, which tests the null hypothesis that the coefficients on a specified subset of independent variables are *all* equal to zero.

Menu path: Model window, /Tests/sum of coefficients

coint

Arguments: *order depvar indepvars*
Option: `--nc` (do not include a constant)
Example: `coint 4 y x1 x2`

The Engle–Granger cointegration test. Carries out Augmented Dickey–Fuller tests on the null hypothesis that each of the variables listed has a unit root, using the given lag order. The cointegrating regression is estimated, and an ADF test is run on the residuals from this regression. The Durbin–Watson statistic for the cointegrating regression is also given.

P-values for this test are based on MacKinnon (1996). The relevant code is included by kind permission of the author.

By default, the cointegrating regression contains a constant. If you wish to suppress the constant, add the `--nc` flag.

Menu path: /Model/Time series/Cointegration test/Engle-Granger

coint2

Arguments:	*order depvar indepvars*
Options:	`--nc` (no constant)
	`--rc` (restricted constant)
	`--crt` (constant and restricted trend)
	`--ct` (constant and unrestricted trend)
	`--seasonals` (include centered seasonal dummies)
	`--verbose` (print details of auxiliary regressions)
Examples:	`coint2 2 y x`
	`coint2 4 y x1 x2 --verbose`
	`coint2 3 y x1 x2 --rc`

Carries out the Johansen test for cointegration among the listed variables for the given lag order. Critical values are computed via J. Doornik's gamma approximation (Doornik, 1998). For details of this test see Hamilton, *Time Series Analysis* (1994), Chapter 20.

The inclusion of deterministic terms in the model is controlled by the option flags. The default if no option is specified is to include an "unrestricted constant", which allows for the presence of a non-zero intercept in the cointegrating relations as well as a trend in the levels of the endogenous variables. In the literature stemming from the work of Johansen (see for example his 1995 book) this is often referred to as "case 3". The first four options given above, which are mutually exclusive, produce cases 1, 2, 4 and 5 respectively. The meaning of these cases and the criteria for selecting a case are explained in the *Gretl User's Guide*.

The `--seasonals` option, which may be combined with any of the other options, specifies the inclusion of a set of centered seasonal dummy variables. This option is available only for quarterly or monthly data.

The following table is offered as a guide to the interpretation of the results shown for the test, for the 3-variable case. `H0` denotes the null hypothesis, `H1` the alternative hypothesis, and `c` the number of cointegrating relations.

Rank	Trace test		Lmax test	
	H0	H1	H0	H1
0	c = 0	c = 3	c = 0	c = 1
1	c = 1	c = 3	c = 1	c = 2
2	c = 2	c = 3	c = 2	c = 3

See also the vecm command.

Menu path: /Model/Time series/Cointegration test/Johansen

corc

Arguments:	*depvar indepvars*
Option:	`--vcv` (print covariance matrix)
Example:	`corc 1 0 2 4 6 7`

Computes parameter estimates using the Cochrane–Orcutt iterative procedure (see Section 9.4 of Ramanathan). Iteration is terminated when successive estimates of the autocorrelation coefficient do not differ by more than 0.001 or after 20 iterations.

Menu path: /Model/Time series/Cochrane-Orcutt

corr

Argument: `[` *varlist* `]`
Example: `corr y x1 x2 x3`

Prints the pairwise correlation coefficients for the variables in *varlist*, or for all variables in the data set if *varlist* is not given.

Menu path: /Data/Correlation matrix

Other access: Main window pop-up menu (multiple selection)

corrgm

Arguments: *variable* `[` *maxlag* `]`
Example: `corrgm x 12`

Prints the values of the autocorrelation function for *variable*, which may be specified by name or number. The values are defined as $\hat{\rho}(u_t, u_{t-s})$ where u_t is the tth observation of the variable u and s is the number of lags.

The partial autocorrelations (calculated using the Durbin–Levinson algorithm) are also shown: these are net of the effects of intervening lags. The command also graphs the correlogram and prints the Box–Pierce Q statistic for testing the null hypothesis that the series is "white noise": this is asymptotically distributed as chi-square with degrees of freedom equal to the number of lags used.

If a *maxlag* value is specified the length of the correlogram is limited to at most that number of lags, otherwise the length is determined automatically, as a function of the frequency of the data and the number of observations.

Menu path: /Variable/Correlogram

Other access: Main window pop-up menu (single selection)

criteria

Arguments: *ess T k*
Example: `criteria 23.45 45 8`

Computes the Akaike Information Criterion (AIC) and Schwarz's Bayesian Information Criterion (BIC), given *ess* (error sum of squares), the number of observations (T), and the number of coefficients (k). T, k, and *ess* may be numerical values or names of previously defined variables.

The AIC is computed as in Akaike's original (1974) formulation, namely

$$\mathrm{AIC} = -2\ell + 2k$$

where ℓ denotes the maximized log-likelihood. The BIC is computed as

$$\mathrm{BIC} = -2\ell + k \log T$$

Please see the *Gretl User's Guide* for further details.

critical

Arguments: *dist param1* [*param2*]
Examples:
```
critical t 20
critical X 5
critical F 3 37
critical d 50
```

If *dist* is t, X or F, prints out the critical values for the student's *t*, chi-square or *F* distribution respectively, for the common significance levels and using the specified degrees of freedom, given as *param1* for t and chi-square, or *param1* and *param2* for *F*. If *dist* is d, prints the upper and lower values of the Durbin–Watson statistic at 5 percent significance, for the given number of observations, *param1*, and for the range of 1 to 5 explanatory variables.

Menu path: /Utilities/Statistical tables

cusum

Must follow the estimation of a model via OLS. Performs the CUSUM test for parameter stability. A series of (scaled) one-step ahead forecast errors is obtained by running a series of regressions: the first regression uses the first k observations and is used to generate a prediction of the dependent variable at observation $k + 1$; the second uses the first $k + 1$ observations and generates a prediction for observation $k + 2$, and so on (where k is the number of parameters in the original model).

The cumulated sum of the scaled forecast errors is printed and graphed. The null hypothesis of parameter stability is rejected at the 5 percent significance level if the cumulated sum strays outside of the 95 percent confidence band.

The Harvey–Collier *t*-statistic for testing the null hypothesis of parameter stability is also printed. See Chapter 7 of Greene's *Econometric Analysis* for details.

Menu path: Model window, /Tests/CUSUM

data

Argument: *varlist*

Reads the variables in *varlist* from a database (gretl or RATS 4.0), which must have been opened previously using the open command. In addition, a data frequency and sample range must be established using the setobs and smpl commands prior to using this command. Here is a full example:

```
open macrodat.rat
setobs 4 1959:1
smpl ; 1999:4
data GDP_JP GDP_UK
```

These commands open a database named macrodat.rat, establish a quarterly data set starting in the first quarter of 1959 and ending in the fourth quarter of 1999, and then import the series named GDP_JP and GDP_UK.

If the series to be read are of higher frequency than the working data set, you must specify a compaction method as below:

```
data (compact=average) LHUR PUNEW
```

The four available compaction methods are "average" (takes the mean of the high frequency observations), "last" (uses the last observation), "first" and "sum".

Menu path: /File/Browse databases

delete

Argument: [*varlist*]

Removes the listed variables (given by name or number) from the dataset. *Use with caution*: no confirmation is asked, and any variables with higher ID numbers will be re-numbered.

If no *varlist* is given with this command, it deletes the last (highest numbered) variable from the dataset.

Menu path: Main window pop-up (single selection)

diff

Argument: *varlist*

The first difference of each variable in *varlist* is obtained and the result stored in a new variable with the prefix `d_`. Thus `diff x y` creates the new variables

```
d_x = x(t) - x(t-1)
d_y = y(t) - y(t-1)
```

Menu path: /Data/Add variables/first differences

else

See if.

end

Ends a block of commands of some sort. For example, `end system` terminates an equation system.

endif

See if.

endloop

Marks the end of a command loop. See loop.

eqnprint

Argument: [*-f filename*]
Option: `--complete` (Create a complete document)

Must follow the estimation of a model. Prints the estimated model in the form of a LaTeX equation. If a filename is specified using the `-f` flag output goes to that file, otherwise it goes to a file with a name of the form `equation_N.tex`, where `N` is the number of models estimated to date in the current session. See also tabprint.

If the `--complete` flag is given, the LaTeX file is a complete document, ready for processing; otherwise it must be included in a document.

Menu path: Model window, /LaTeX

equation

Arguments: *depvar indepvars*
Example: `equation y x1 x2 x3 const`

Specifies an equation within a system of equations (see system). The syntax for specifying an equation within an SUR system is the same as that for, e.g., ols. For an equation within a Three-Stage Least Squares system you may either (a) give an OLS-type equation specification and provide a common list of instruments using the `instr` keyword (again, see system), or (b) use the same equation syntax as for tsls.

estimate

Arguments: *systemname estimator*
Options: `--iterate` (iterate to convergence)
`--no-df-corr` (no degrees of freedom correction)
`--geomean` (see below)
Examples: `estimate "Klein Model 1" method=fiml`
`estimate Sys1 method=sur`
`estimate Sys1 method=sur --iterate`

Calls for estimation of a system of equations, which must have been previously defined using the system command. The name of the system should be given first, surrounded by double quotes if the name contains spaces. The estimator, which must be one of `ols`, `tsls`, `sur`, `3sls`, `fiml` or `liml`, is preceded by the string `method=`.

If the system in question has had a set of restrictions applied (see the restrict command), estimation will be subject to the specified restrictions.

If the estimation method is `sur` or `3sls` and the `--iterate` flag is given, the estimator will be iterated. In the case of SUR, if the procedure converges the results are maximum likelihood estimates. Iteration of three-stage least squares, however, does not in general converge on the full-information maximum likelihood results. The `--iterate` flag is ignored for other methods of estimation.

If the equation-by-equation estimators `ols` or `tsls` are chosen, the default is to apply a degrees of freedom correction when calculating standard errors. This can be suppressed using the `--no-df-corr` flag. This flag has no effect with the other estimators; no degrees of freedom correction is applied in any case.

By default, the formula used in calculating the elements of the cross-equation covariance matrix is

$$\hat{\sigma}_{i,j} = \frac{\hat{u}_i' \hat{u}_j}{T}$$

If the `--geomean` flag is given, a degrees of freedom correction is applied: the formula is

$$\hat{\sigma}_{i,j} = \frac{\hat{u}_i' \hat{u}_j}{\sqrt{(T - k_i)(T - k_j)}}$$

where the ks denote the number of independent parameters in each equation.

fcast

Arguments: `[` *startobs endobs* `]` *fitvar*
Options: `--dynamic` (create dynamic forecast)
`--static` (create static forecast)
Examples: `fcast 1997:1 2001:4 f1`
`fcast fit2`

Must follow an estimation command. Forecasts are generated for the specified range (or the largest possible range if no *startobs* and *endobs* are given) and the values saved as *fitvar*, which can be printed, graphed, or plotted. The right-hand side variables are those in the original model. There is no provision to substitute other variables. If an autoregressive error process is specified the forecast incorporates the predictable fraction of the error process.

The choice between a static and a dynamic forecast applies only in the case of dynamic models, with an autoregressive error process or including one or more lagged values of the dependent variable as regressors. See fcasterr for more details.

Menu path: Model window, /Model data/Forecasts

fcasterr

Arguments:	*startobs endobs*
Options:	`--plot` (display graph)
	`--dynamic` (create dynamic forecast)
	`--static` (create static forecast)

After estimating a model you can use this command to print out fitted values over the specified observation range, along with (depending on the nature of the model and the available data) estimated standard errors of those predictions and 95 percent confidence intervals.

The choice between a static and a dynamic forecast applies only in the case of dynamic models, with an autoregressive error process or including one or more lagged values of the dependent variable as regressors. Static forecasts are one step ahead, based on realized values from the previous period, while dynamic forecasts employ the chain rule of forecasting. For example, if a forecast for y in 2008 requires as input a value of y for 2007, a static forecast is impossible without actual data for 2007. A dynamic forecast for 2008 is possible if a prior forecast can be substituted for y in 2007.

The default is to give a static forecast for any portion of the forecast range that lies with the sample range over which the model was estimated, and a dynamic forecast (if relevant) out of sample. The `dynamic` option requests a dynamic forecast from the earliest possible date, and the `static` option requests a static forecast even out of sample.

The nature of the forecast standard errors (if available) depends on the nature of the model and the forecast. For static linear models standard errors are computed using the method outlined by Davidson and MacKinnon (2004); they incorporate both uncertainty due to the error process and parameter uncertainty (summarized in the covariance matrix of the parameter estimates). For dynamic models, forecast standard errors are computed only in the case of a dynamic forecast, and they do not incorporate parameter uncertainty. For nonlinear models, forecast standard errors are not presently available.

Menu path: Model window, /Model data/Forecasts

fit

A shortcut to fcast. Must follow an estimation command. Generates fitted values, in a series called `autofit`, for the current sample, based on the last regression. In the case of time-series models, also pops up a graph of fitted and actual values of the dependent variable against time.

freq

Argument:	*var*
Options:	`--quiet` (suppress printing of histogram)
	`--gamma` (test for gamma distribution)

With no options given, displays the frequency distribution for *var* (given by name or number) and shows the results of the Doornik–Hansen chi-square test for normality.

If the `--quiet` option is given, the histogram is not shown. If the `--gamma` option is given, the test for normality is replaced by Locke's nonparametric test for the null hypothesis that the variable follows the gamma distribution; see Locke (1976), Shapiro and Chen (2001).

In interactive mode a graph of the distribution is displayed.

Menu path: /Variable/Frequency distribution

function

Argument: *fnname*

Opens a block of statements in which a function is defined. This block must be closed with `end function`. Please see the *Gretl User's Guide* for details.

garch

Arguments: *p q* `;` *depvar* `[` *indepvars* `]`
Options: `--robust` (robust standard errors)
`--verbose` (print details of iterations)
`--vcv` (print covariance matrix)
`--arma-init` (initial variance parameters from ARMA)
Examples:
```
garch 1 1 ; y
garch 1 1 ; y 0 x1 x2 --robust
```

Estimates a GARCH model (GARCH = Generalized Autoregressive Conditional Heteroskedasticity), either a univariate model or, if *indepvars* are specified, including the given exogenous variables. The integer values p and q represent the lag orders in the conditional variance equation:

$$h_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_i h_{t-j}$$

The gretl GARCH algorithm is basically that of Fiorentini, Calzolari and Panattoni (1996), used by kind permission of Professor Fiorentini.

Several variant estimates of the coefficient covariance matrix are available with this command. By default, the Hessian is used unless the `--robust` option is given, in which case the QML (White) covariance matrix is used. Other possibilities (e.g. the information matrix, or the Bollerslev–Wooldridge estimator) can be specified using the set command.

By default, the estimates of the variance parameters are initialized using the unconditional error variance from initial OLS estimation for the constant, and small positive values for the coefficients on the past values of the squared error and the error variance. The flag `--arma-init` calls for the starting values of these parameters to be set using an initial ARMA model, exploiting the relationship between GARCH and ARMA set out in Chapter 21 of Hamilton's *Time Series Analysis*. In some cases this may improve the chances of convergence.

Menu path: /Model/Time series/GARCH model

genr

Arguments: *newvar = formula*

Creates new variables, usually through transformations of existing variables. See also diff, logs, lags, ldiff, multiply, sdiff and square for shortcuts. In the context of a `genr` formula, existing variables must be referenced by name, not ID number. The formula should be a well-formed combination of variable names, constants, operators and functions (described below). Note that further details on some aspects of this command can be found in the *Gretl User's Guide*.

This command may yield either a series or a scalar result. For example, the formula `x2 = x * 2` naturally yields a series if the variable `x` is a series and a scalar if `x` is a scalar. The formulae `x = 0` and `mx = mean(x)` naturally return scalars. Under some circumstances you may want to have a scalar result expanded into a series or vector. You can do this by using `series` as an "alias" for the `genr` command. For example, `series x = 0` produces a series all of whose values are set to 0. You can also use `scalar` as an alias for `genr`. It is not possible to coerce a vector result into a scalar, but use of this keyword indicates that the result *should be* a scalar: if it is not, an error occurs.

When a formula yields a series or vector result, the range over which the result is written to the target variable depends on the current sample setting. It is possible, therefore, to define a series piecewise using the `smpl` command in conjunction with `genr`.

Supported *arithmetical operators* are, in order of precedence: `^` (exponentiation); `*`, `/` and `%` (modulus or remainder); `+` and `-`.

The available *Boolean operators* are (again, in order of precedence): `!` (negation), `&` (logical AND), `|` (logical OR), `>`, `<`, `=`, `>=` (greater than or equal), `<=` (less than or equal) and `!=` (not equal). The Boolean operators can be used in constructing dummy variables: for instance `(x > 10)` returns 1 if $x > 10$, 0 otherwise.

Built-in constants are `pi` and `NA`. The latter is the missing value code: you can initialize a variable to the missing value with `scalar x = NA`.

Supported *functions* fall into these groups:

- Standard mathematical functions: `abs`, `cos`, `exp`, `int` (integer part), `ln` (natural logarithm: `log` is a synonym), `sin`, `sqrt`. All of these take a single argument, which may be either a series or a scalar.
- Standard statistical functions taking a single argument and yielding a scalar result: `max` (maximum value in a series), `min` (minimum value in series), `mean` (arithmetic mean), `median`, `var` (variance), `sd` (standard deviation), `sst` (sum of squared deviations from the mean), `sum`.
- Statistical functions taking one series as argument and yielding a series or vector result: `sort` (sort a series is ascending order of magnitude), `cum` (cumulate, or running sum).
- Statistical functions taking two series as arguments and yielding a scalar result: `cov` (covariance), `corr` (correlation coefficient).
- Special statistical functions: `pvalue` and `critical` (see below), `cnorm` (standard normal CDF), `dnorm` (standard normal PDF), `resample` (resample a series with replacement, for bootstrap purposes), `hpfilt` (Hodrick–Prescott filter: this function returns the "cycle" component of the series), `bkfilt` (Baxter–King bandpass filter).
- Time-series functions: `diff` (first difference), `ldiff` (log-difference, or first difference of natural logs), `sdiff` (seasonal difference), `fracdiff` (fractional difference). To generate lags of a variable `x`, use the syntax `x(-N)`, where `N` represents the desired lag length; to generate leads, use `x(+N)`.
- Dataset functions yielding a series: `misszero` (replaces the missing observation code in a given series with zeros); `zeromiss` (the inverse operation to `misszero`); `missing` (at each observation, 1 if the argument has a missing value, 0 otherwise); `ok` (the opposite of `missing`).
- Dataset functions yielding a scalar: `nobs` (gives the number of valid observations in a data series), `firstobs` (gives the 1-based observation number of the first non-missing value in a series), `lastobs` (observation number of the last non-missing observation in a series).
- Pseudo-random numbers: `uniform` and `normal`. These functions do not take an argument and should be written with empty parentheses: `uniform()`, `normal()`. They create pseudo-random series drawn from the uniform (0–1) and standard normal distributions respectively.

See also the set command, seed option. Uniform series are generated using the Mersenne Twister;[1] for normal series the method of Box and Muller (1958) is used, taking input from the Mersenne Twister.

All of the above functions with the exception of cov, corr, pvalue, critical, fracdiff, uniform and normal take as their single argument either the name of a variable or an expression that evaluates to a variable (e.g. ln((x1+x2)/2)).

The pvalue function takes the same arguments as the pvalue command, but in this context commas should be placed between the arguments. It returns a one-tailed p-value, and in the case of the normal and t distributions, it is for the "short tail". With the normal, for example, both 1.96 and −1.96 will give a result of around 0.025.

The critical function returns the critical value for a specified probability distribution and a specified proportion in the right-hand tail (with specified degrees of freedom where applicable). The distribution parameter, which must come first, can be z or N (Normal distribution), t (Student's t), X (chi-square) or F. The last parameter is the right-hand tail proportion. If the first parameter is t or X, a second parameter must give the degrees of freedom. For the F distribution, the second and third parameters must give the numerator and denominator degrees of freedom.

Here are some examples of use of the pvalue and critical functions (spaces between the arguments are optional):

```
genr p1 = pvalue(z, 2.2)
genr p2 = pvalue(X, 3, 5.67)
genr c1 = critical(t, 20, 0.025)
genr c2 = critical(F, 4, 48, 0.05)
```

The fracdiff function takes two arguments: the name of the series and a fraction, in the range −1 to 1.

Besides the operators and functions noted above there are some special uses of genr:

- genr time creates a time trend variable (1,2,3,...) called time. genr index does the same thing except that the variable is called index.
- genr dummy creates dummy variables up to the periodicity of the data. For example, in the case of quarterly data (periodicity 4), the program creates dummy_1 = 1 for first quarter and 0 in other quarters, dummy_2 = 1 for the second quarter and 0 in other quarters, and so on.
- genr paneldum creates a set of special dummy variables for use with a panel data set — see panel.

Various internal variables defined in the course of running a regression can be retrieved using genr, as follows:

- $ess: error sum of squares
- $rsq: unadjusted R-squared
- $T: number of observations used
- $df: degrees of freedom
- $ncoeff: total number of estimated coefficients

[1] See Matsumoto and Nishimura (1998). The implementation is provided by glib, if available, or by the C code written by Nishimura and Matsumoto.

- $trsq: *TR*-squared (sample size times *R*-squared)
- $sigma: standard error of residuals
- $aic: Akaike Information Criterion
- $bic: Schwarz's Bayesian Information Criterion
- $lnl: log-likelihood (where applicable)
- coeff(*var*): estimated coefficient for variable *var*
- stderr(*var*): estimated standard error for variable *var*
- rho(*i*): *i*th order autoregressive coefficient for residuals
- vcv(*x1*,*x2*): estimated covariance between coefficients for the variables *x1* and *x2*

Note: In the command-line program, genr commands that retrieve model-related data always reference the model that was estimated most recently. This is also true in the GUI program, if one uses genr in the "gretl console" or enters a formula using the "Define new variable" option under the Variable menu in the main window. With the GUI, however, you have the option of retrieving data from any model currently displayed in a window (whether or not it's the most recent model). You do this under the "Model data" menu in the model's window.

The internal series $uhat and $yhat hold, respectively, the residuals and fitted values from the last regression.

Three "internal" dataset variables are available: $nobs holds the number of observations in the current sample range (which may or may not equal the value of $T, the number of observations used in estimating the last model); $nvars holds the number of variables in the dataset (including the constant); and $pd holds the frequency or periodicity of the data (e.g. 4 for quarterly data).

Two special internal scalars, $test and $pvalue, hold respectively the value and the p-value of the test statistic that was generated by the last explicit hypothesis-testing command, if any (e.g. chow). Please see the *Gretl User's Guide* for details on this.

The variable t serves as an index of the observations. For instance genr dum = (t=15) will generate a dummy variable that has value 1 for observation 15, 0 otherwise. The variable obs is similar but more flexible: you can use this to pick out particular observations by date or name. For example, genr d = (obs>1986:4) or genr d = (obs="CA"). The last form presumes that the observations are labeled; the label must be put in double quotes.

Scalar values can be pulled from a series in the context of a genr formula, using the syntax *varname*[*obs*]. The *obs* value can be given by number or date. Examples: x[5], CPI[1996:01]. For daily data, the form *YYYY/MM/DD* should be used, e.g. ibm[1970/01/23].

An individual observation in a series can be modified via genr. To do this, a valid observation number or date, in square brackets, must be appended to the name of the variable on the left-hand side of the formula. For example, genr x[3] = 30 or genr x[1950:04] = 303.7.

Here are a couple of tips on dummy variables:

- Suppose x is coded with values 1, 2, or 3 and you want three dummy variables, d1 = 1 if x = 1, 0 otherwise, d2 = 1 if x = 2, and so on. To create these, use the commands:

```
genr d1 = (x=1)
genr d2 = (x=2)
genr d3 = (x=3)
```

- To create z = max(x,y) do

Table 1.1: Examples of use of genr command

Formula	*Comment*
`y = x1^3`	`x1` cubed
`y = ln((x1+x2)/x3)`	
`z = x>y`	`z(t)` = 1 if `x(t) > y(t)`, otherwise 0
`y = x(-2)`	`x` lagged 2 periods
`y = x(+2)`	`x` led 2 periods
`y = diff(x)`	`y(t) = x(t) - x(t-1)`
`y = ldiff(x)`	`y(t) = log x(t) - log x(t-1)`, the instantaneous rate of growth of `x`
`y = sort(x)`	sorts `x` in increasing order and stores in `y`
`y = -sort(-x)`	sort `x` in decreasing order
`y = int(x)`	truncate `x` and store its integer value as `y`
`y = abs(x)`	store the absolute values of `x`
`y = sum(x)`	sum `x` values excluding missing −999 entries
`y = cum(x)`	cumulation: $y_t = \sum_{\tau=1}^{t} x_\tau$
`aa = $ess`	set `aa` equal to the Error Sum of Squares from last regression
`x = coeff(sqft)`	grab the estimated coefficient on the variable `sqft` from the last regression
`rho4 = rho(4)`	grab the 4th-order autoregressive coefficient from the last model (presumes an `ar` model)
`cvx1x2 = vcv(x1, x2)`	grab the estimated coefficient covariance of vars `x1` and `x2` from the last model
`foo = uniform()`	uniform pseudo-random variable in range 0–1
`bar = 3 * normal()`	normal pseudo-random variable, $\mu = 0$, $\sigma = 3$
`samp = ok(x)`	= 1 for observations where `x` is not missing.

```
genr d = x>y
genr z = (x*d)+(y*(1-d))
```

Menu path: /Variable/Define new variable

Other access: Main window pop-up menu

gnuplot

Arguments:	*yvars xvar* [*dumvar*]
Options:	`--with-lines` (use lines, not points)
	`--with-impulses` (use vertical lines)
	`--suppress-fitted` (don't show least squares fit)
	`--dummy` (see below)
Examples:	`gnuplot y1 y2 x`
	`gnuplot x time --with-lines`
	`gnuplot wages educ gender --dummy`

Without the `--dummy` option, the *yvars* are graphed against *xvar*. With `--dummy`, *yvar* is graphed against *xvar* with the points shown in different colors depending on whether the value of *dumvar* is 1 or 0.

The `time` variable behaves specially: if it does not already exist then it will be generated automatically.

In interactive mode the result is displayed immediately. In batch mode a gnuplot command file is written, with a name on the pattern `gptmpN.plt`, starting with N = `01`. The actual plots may be generated later using `gnuplot` (under MS Windows, `wgnuplot`).

A further option to this command is available: following the specification of the variables to be plotted and the option flag (if any), you may add literal gnuplot commands to control the appearance of the plot (for example, setting the plot title and/or the axis ranges). These commands should be enclosed in braces, and each gnuplot command must be terminated with a semi-colon. A backslash may be used to continue a set of gnuplot commands over more than one line. Here is an example of the syntax:

```
set title 'My Title'; set yrange [0:1000];
```

Menu path: /Data/Graph specified vars

Other access: Main window pop-up menu, graph button on toolbar

graph

Arguments:	*yvars xvar*
Option:	`--tall` (use 40 rows)

ASCII graphics. The *yvars* (which may be given by name or number) are graphed against *xvar* using ASCII symbols. The `--tall` flag will produce a graph with 40 rows and 60 columns. Without it, the graph will be 20 by 60 (for screen output). See also gnuplot.

hausman

This test is available only after estimating a model using the pooled command (see also `panel` and `setobs`). It tests the simple pooled model against the principal alternatives, the fixed effects and random effects models.

The fixed effects model adds a dummy variable for all but one of the cross-sectional units, allowing the intercept of the regression to vary across the units. An *F*-test for the joint significance of

these dummies is presented. The random effects model decomposes the residual variance into two parts, one part specific to the cross-sectional unit and the other specific to the particular observation. (This estimator can be computed only if the number of cross-sectional units in the data set exceeds the number of parameters to be estimated.) The Breusch–Pagan LM statistic tests the null hypothesis (that the pooled OLS estimator is adequate) against the random effects alternative.

The pooled OLS model may be rejected against both of the alternatives, fixed effects and random effects. Provided the unit- or group-specific error is uncorrelated with the independent variables, the random effects estimator is more efficient than the fixed effects estimator; otherwise the random effects estimator is inconsistent and the fixed effects estimator is to be preferred. The null hypothesis for the Hausman test is that the group-specific error is not so correlated (and therefore the random effects model is preferable). A low p-value for this test counts against the random effects model and in favor of fixed effects.

Menu path: Model window, /Tests/panel diagnostics

hccm

Arguments: *depvar indepvars*
Option: `--vcv` (print covariance matrix)

Heteroskedasticity-Consistent Covariance Matrix: this command runs a regression where the coefficients are estimated via the standard OLS procedure, but the standard errors of the coefficient estimates are computed in a manner that is robust in the face of heteroskedasticity, namely using the MacKinnon–White "jackknife" procedure.

Menu path: /Model/HCCM

help

Gives a list of available commands. `help` *command* describes *command* (e.g. `help smpl`). You can type `man` instead of `help` if you like.

Menu path: /Help

hilu

Arguments: *depvar indepvars*
Options: `--vcv` (print covariance matrix)
`--no-corc` (do not fine-tune results with Cochrane-Orcutt)

Computes parameter estimates for the specified model using the Hildreth–Lu search procedure. The results are fine-tuned using the Cochrane–Orcutt iterative method, unless the `--no-corc` flag is specified.

This procedure is designed to correct for serial correlation of the error term. The error sum of squares of the transformed model is graphed against the value of rho from −0.99 to 0.99.

Menu path: /Model/Time Series/Hildreth-Lu

hsk

Arguments: *depvar indepvars*
Option: `--vcv` (print covariance matrix)

An OLS regression is run and the residuals are saved. The logs of the squares of these residuals then become the dependent variable in an auxiliary regression, on the right-hand side of which are the original independent variables plus their squares. The fitted values from the auxiliary regression are then used to construct a weight series, and the original model is re-estimated using weighted least squares. This final result is reported.

The weight series is formed as $1/\sqrt{e^{y^*}}$, where y^* denotes the fitted values from the auxiliary regression.

Menu path: /Model/Heteroskedasticity corrected

hurst

Argument: *varname*

Calculates the Hurst exponent (a measure of persistence or long memory) for a time-series variable having at least 128 observations.

The Hurst exponent is discussed by Mandelbrot. In theoretical terms it is the exponent, H, in the relationship

$$\mathrm{RS}(x) = an^H$$

where RS is the "rescaled range" of the variable x in samples of size n and a is a constant. The rescaled range is the range (maximum minus minimum) of the cumulated value or partial sum of x over the sample period (after subtraction of the sample mean), divided by the sample standard deviation.

As a reference point, if x is white noise (zero mean, zero persistence) then the range of its cumulated "wandering" (which forms a random walk), scaled by the standard deviation, grows as the square root of the sample size, giving an expected Hurst exponent of 0.5. Values of the exponent significantly in excess of 0.5 indicate persistence, and values less than 0.5 indicate anti-persistence (negative autocorrelation). In principle the exponent is bounded by 0 and 1, although in finite samples it is possible to get an estimated exponent greater than 1.

In gretl, the exponent is estimated using binary sub-sampling: we start with the entire data range, then the two halves of the range, then the four quarters, and so on. For sample sizes smaller than the data range, the RS value is the mean across the available samples. The exponent is then estimated as the slope coefficient in a regression of the log of RS on the log of sample size.

Menu path: /Variable/Hurst exponent

if

Flow control for command execution. The syntax is:

```
if condition
  commands1
else
  commands2
endif
```

condition must be a Boolean expression, for the syntax of which see genr. The `else` block is optional; `if ... endif` blocks may be nested.

import

Argument: *filename*
Option: `--box1` (BOX1 data)

Brings in data from a comma-separated values (CSV) format file, such as can easily be written from a spreadsheet program. The file should have variable names on the first line and a rectangular data matrix on the remaining lines. Variables should be arranged "by observation" (one column per variable; each row represents an observation). See the *Gretl User's Guide* for details.

With the `--box1` flag, reads a data file in BOX1 format, which could at one time be obtained via the Data Extraction Service of the US Bureau of the Census.

Menu path: /File/Open data/import

include

Argument: *inputfile*

Intended for use in a command script, primarily for including definitions of functions. Executes the commands in *inputfile* then returns control to the main script.

See also run.

info

Prints out any supplementary information stored with the current datafile.

Menu path: /Data/Read info

Other access: Data browser windows

label

Arguments: *varname* `-d` *description* `-n` *displayname*
Example: `label x1 -d "Description of x1" -n "Graph name"`

Sets the descriptive label for the given variable (if the `-d` flag is given, followed by a string in double quotes) and/or the "display name" for the variable (if the `-n` flag is given, followed by a quoted string). If a variable has a display name, this is used when generating graphs.

If neither flag is given, this command prints the current descriptive label for the specified variable, if any.

Menu path: /Variable/Edit attributes

Other access: Main window pop-up menu

kpss

Arguments: *order varname*
Options: `--trend` (include a trend)
`--verbose` (print regression results)
Examples: `kpss 8 y`
`kpss 4 x1 --trend`

Computes the KPSS test (Kwiatkowski, Phillips, Schmidt and Shin, 1992) for stationarity of a variable. The null hypothesis is that the variable in question is stationary, either around a level or, if the `--trend` option is given, around a deterministic linear trend.

The order argument determines the size of the window used for Bartlett smoothing. If the `--verbose` option is chosen the results of the auxiliary regression are printed, along with the estimated variance of the random walk component of the variable.

Menu path: /Variable/KPSS test

labels

Prints out the informative labels for any variables that have been generated using `genr`, and any labels added to the data set via the GUI.

lad

Arguments: *depvar indepvars*
Option: `--vcv` (print covariance matrix)

Calculates a regression that minimizes the sum of the absolute deviations of the observed from the fitted values of the dependent variable. Coefficient estimates are derived using the Barrodale–Roberts simplex algorithm; a warning is printed if the solution is not unique.

Standard errors are derived using the bootstrap procedure with 500 drawings. The covariance matrix for the parameter estimates, printed when the `--vcv` flag is given, is based on the same bootstrap.

Menu path: /Model/Least Absolute Deviation

lags

Variants: `lags` *varlist*
`lags` *order* `;` *varlist*
Examples: `lags x y`
`lags 12 ; x y`

Creates new variables which are lagged values of each of the variables in *varlist*. By default the number of lagged variables equals the periodicity of the data. For example, if the periodicity is 4 (quarterly), the command `lags x` creates

```
x_1 = x(t-1)
x_2 = x(t-2)
x_3 = x(t-3)
x_4 = x(t-4)
```

The number of lags created can be controlled by the optional first parameter.

Menu path: /Data/Add variables/lags of selected variables

ldiff

Argument: *varlist*

The first difference of the natural log of each variable in *varlist* is obtained and the result stored in a new variable with the prefix `ld_`. Thus `ldiff x y` creates the new variables

```
ld_x = log(x) - log(x(-1))
ld_y = log(y) - log(y(-1))
```

Menu path: /Data/Add variables/log differences

leverage

Option: `--save` (save variables)

Must immediately follow an `ols` command. Calculates the leverage (h, which must lie in the range 0 to 1) for each data point in the sample on which the previous model was estimated. Displays the residual (u) for each observation along with its leverage and a measure of its influence on the estimates, $uh/(1-h)$. "Leverage points" for which the value of h exceeds $2k/n$ (where k is the number of parameters being estimated and n is the sample size) are flagged with an asterisk. For details on the concepts of leverage and influence see Davidson and MacKinnon (1993, Chapter 2).

DFFITS values are also shown: these are "studentized residuals" (predicted residuals divided by their standard errors) multiplied by $\sqrt{h/(1-h)}$. For a discussion of studentized residuals and DFFITS see G. S. Maddala, *Introduction to Econometrics*, chapter 12; also Belsley, Kuh and Welsch (1980).

Briefly, a "predicted residual" is the difference between the observed value of the dependent variable at observation t, and the fitted value for observation t obtained from a regression in which that

observation is omitted (or a dummy variable with value 1 for observation t alone has been added); the studentized residual is obtained by dividing the predicted residual by its standard error.

If the `--save` flag is given with this command, then the leverage, influence and DFFITS values are added to the current data set.

Menu path: Model window, /Tests/influential observations

lmtest

Options: `--logs` (non-linearity, logs)
`--autocorr` (serial correlation)
`--squares` (non-linearity, squares)
`--white` (heteroskedasticity (White's test))

Must immediately follow an `ols` command. Depending on the options given, this command carries out some combination of the following: Lagrange Multiplier tests for nonlinearity (logs and squares); White's test for heteroskedasticity; and the LMF test for serial correlation up to the periodicity (see Kiviet, 1986). The corresponding auxiliary regression coefficients are also printed out. See Ramanathan, Chapters 7, 8, and 9 for details.

Menu path: Model window, /Tests

logistic

Arguments: *depvar indepvars* `[ ymax=`*value*` ]`
Option: `--vcv` (print covariance matrix)
Examples:
```
logistic y const x
logistic y const x ymax=50
```

Logistic regression: carries out an OLS regression using the logistic transformation of the dependent variable,

$$\log\left(\frac{y}{y^* - y}\right)$$

The dependent variable must be strictly positive. If it is a decimal fraction, between 0 and 1, the default is to use a y^* value (the asymptotic maximum of the dependent variable) of 1. If the dependent variable is a percentage, between 0 and 100, the default y^* is 100.

If you wish to set a different maximum, use the optional `ymax=`*value* syntax following the list of regressors. The supplied value must be greater than all of the observed values of the dependent variable.

The fitted values and residuals from the regression are automatically transformed using

$$y = \frac{y^*}{1 + e^{-x}}$$

where x represents either a fitted value or a residual from the OLS regression using the transformed dependent variable. The reported values are therefore comparable with the original dependent variable.

Note that if the dependent variable is binary, you should use the logit command instead.

Menu path: /Model/Logistic

logit

Arguments: *depvar indepvars*
Options: `--robust` (robust standard errors)
`--vcv` (print covariance matrix)

Binomial logit regression. The dependent variable should be a binary variable. Maximum likelihood estimates of the coefficients on *indepvars* are obtained via the "binary response model regression" (BRMR) method outlined by Davidson and MacKinnon (2004). As the model is nonlinear the slopes depend on the values of the independent variables: the reported slopes are evaluated at the means of those variables. The chi-square statistic tests the null hypothesis that all coefficients are zero apart from the constant.

By default, standard errors are computed using the negative inverse of the Hessian. If the `--robust` flag is given, then QML or Huber–White standard errors are calculated instead. In this case the estimated covariance matrix is a "sandwich" of the inverse of the estimated Hessian and the outer product of the gradient. See Davidson and MacKinnon (2004, Chapter 10) for details.

If you want to use logit for analysis of proportions (where the dependent variable is the proportion of cases having a certain characteristic, at each observation, rather than a 1 or 0 variable indicating whether the characteristic is present or not) you should not use the `logit` command, but rather construct the logit variable, as in

```
genr lgt_p = log(p/(1 - p))
```

and use this as the dependent variable in an OLS regression. See Ramanathan, Chapter 12.

Menu path: /Model/Logit

logs

Argument: *varlist*

The natural log of each of the variables in *varlist* is obtained and the result stored in a new variable with the prefix `l_` which is "el" underscore. `logs x y` creates the new variables `l_x` = ln(x) and `l_y` = ln(y).

Menu path: /Data/Add variables/logs of selected variables

loop

Argument: *control*
Options: `--progressive` (enable special forms of certain commands)
`--verbose` (report details of genr commands)
Examples:
```
loop 1000
loop 1000 --progressive
loop while essdiff > .00001
loop for i=1991..2000
loop for (r=-.99; r<=.99; r+=.01)
```

The parameter *control* must take one of four forms, as shown in the examples: an integer number of times to repeat the commands within the loop; "`while`" plus a numerical condition; "`for`" plus a range of values for the internal index variable `i`; or "`for`" plus three expressions in parentheses, separated by semicolons. In the last form the left-hand expression initializes a variable, the middle expression sets a condition for iteration to continue, and the right-hand expression sets an increment or decrement to be applied at the start of the second and subsequent iterations. (This is a restricted form of the `for` statement in the C programming language.)

This command opens a special mode in which the program accepts commands to be executed repeatedly. Within a loop, only certain commands can be used: `genr`, `ols`, `print`, `printf`, `pvalue`, `sim`, `smpl`, `store`, `summary`, `if`, `else` and `endif`. You exit the mode of entering loop commands with `endloop`: at this point the stacked commands are executed.

See the *Gretl User's Guide* for further details and examples. The effect of the `--progressive` option (which is designed for use in Monte Carlo simulations) is explained there.

mahal

Argument: *varlist*
Options: `--save` (add distances to the dataset)
`--vcv` (print covariance matrix)

The Mahalanobis distance is the distance between two points in an k-dimensional space, scaled by the statistical variation in each dimension of the space. For example, if p and q are two observations on a set of k variables with covariance matrix C, then the Mahalanobis distance between the observations is given by

$$\sqrt{(p-q)'C^{-1}(p-q)}$$

where $(p-q)$ is a k-vector. This reduces to Euclidean distance if the covariance matrix is the identity matrix.

The space for which distances are computed is defined by the selected variables. For each observation in the current sample range, the distance is computed between the observation and the centroid of the selected variables. This distance is the multidimensional counterpart of a standard z-score, and can be used to judge whether a given observation "belongs" with a group of other observations.

If the `--vcv` option is given, the covariance matrix and its inverse are printed. If the `--save` option is given, the distances are saved to the dataset under the name `mdist` (or `mdist1`, `mdist2` and so on if there is already a variable of that name).

Menu path: /Data/Mahalanobis distances

meantest

Arguments: *var1 var2*
Option: `--unequal-vars` (assume variances are unequal)

Calculates the t statistic for the null hypothesis that the population means are equal for the variables *var1* and *var2*, and shows its p-value.

By default the test statistic is calculated on the assumption that the variances are equal for the two variables; with the `--unequal-vars` option the variances are assumed to be different. This will make a difference to the test statistic only if there are different numbers of non-missing observations for the two variables.

Menu path: /Data/Difference of means

mle

Arguments: *log-likelihood function derivatives*
Options: `--vcv` (print covariance matrix)
`--verbose` (print details of iterations)

Performs Maximum Likelihood (ML) estimation using the BFGS (Broyden, Fletcher, Goldfarb, Shanno) algorithm. The user must specify the log-likelihood function. The parameters of this function must be declared and given starting values (using the `genr` command) prior to estimation. Optionally, the user may specify the derivatives of the log-likelihood function with respect to each of the parameters; if analytical derivatives are not supplied, a numerical approximation is computed.

Simple example: Suppose we have a series `X` with values 0 or 1 and we wish to obtain the maximum likelihood estimate of the probability, `p`, that `X` = 1. (In this simple case we can guess in advance that the ML estimate of `p` will simply equal the proportion of Xs equal to 1 in the sample.)

The parameter p must first be added to the dataset and given an initial value. This can be done using the genr command. For example, genr p = 0.5.

We then construct the MLE command block:

```
mle loglik = X*log(p) + (1-X)*log(1-p)
deriv p = X/p - (1-X)/(1-p)
end mle
```

The first line above specifies the log-likelihood function. It starts with the keyword mle, then a dependent variable is specified and an expression for the log-likelihood is given (using the same syntax as in the genr command). The next line (which is optional) starts with the keyword deriv and supplies the derivative of the log-likelihood function with respect to the parameter p. If no derivatives are given, you should include a statement using the keyword params which identifies the free parameters: these are listed on one line, separated by spaces. For example, the above could be changed to:

```
mle loglik = X*log(p) + (1-X)*log(1-p)
params p
end mle
```

in which case numerical derivatives would be used.

The option flags, --vcv or --verbose, should be appended to the ending line of the MLE block.

Menu path: /Model/Maximum likelihood

modeltab

Arguments: *add* or *show* or *free*

Manipulates the gretl "model table". See the *Gretl User's Guide* for details. The sub-commands have the following effects: add adds the last model estimated to the model table, if possible; show displays the model table in a window; and free clears the table.

Menu path: Session window, Model table icon

mpols

Arguments: *depvar indepvars*

Computes OLS estimates for the specified model using multiple precision floating-point arithmetic. This command is available only if gretl is compiled with support for the Gnu Multiple Precision library (GMP).

To estimate a polynomial fit, using multiple precision arithmetic to generate the required powers of the independent variable, use the form, e.g. mpols y 0 x ; 2 3 4 This does a regression of y on x, x squared, x cubed and x to the fourth power. That is, the numbers (which must be positive integers) to the right of the semicolon specify the powers of x to be used. If more than one independent variable is specified, the last variable before the semicolon is taken to be the one that should be raised to various powers.

Menu path: /Model/High precision OLS

multiply

Arguments: *x suffix varlist*

Examples:
```
multiply invpop pc 3 4 5 6
multiply 1000 big x1 x2 x3
```

The variables in *varlist* (referenced by name or number) are multiplied by *x*, which may be either a numerical value or the name of a variable already defined. The products are named with the specified *suffix* (maximum 3 characters). The original variable names are truncated first if need be. For instance, suppose you want to create per capita versions of certain variables, and you have the variable pop (population). A suitable set of commands is then:

```
genr invpop = 1/pop
multiply invpop pc income
```

which will create incomepc as the product of income and invpop, and expendpc as expend times invpop.

nls

Arguments: *function derivatives*
Option: --vcv (print covariance matrix)

Performs Nonlinear Least Squares (NLS) estimation using a modified version of the Levenberg–Marquandt algorithm. The user must supply a function specification. The parameters of this function must be declared and given starting values (using the genr command) prior to estimation. Optionally, the user may specify the derivatives of the regression function with respect to each of the parameters; if analytical derivatives are not supplied, a numerical approximation to the Jacobian is computed.

It is easiest to show what is required by example. The following is a complete script to estimate the nonlinear consumption function set out in William Greene's *Econometric Analysis* (Chapter 11 of the 4th edition, or Chapter 9 of the 5th). The numbers to the left of the lines are for reference and are not part of the commands. Note that the --vcv option, for printing the covariance matrix of the parameter estimates, attaches to the final command, end nls.

```
1   open greene11_3.gdt
2   ols C 0 Y
3   genr alpha = coeff(0)
4   genr beta = coeff(Y)
5   genr gamma = 1.0
6   nls C = alpha + beta * Y^gamma
7   deriv alpha = 1
8   deriv beta = Y^gamma
9   deriv gamma = beta * Y^gamma * log(Y)
10  end nls --vcv
```

It is often convenient to initialize the parameters by reference to a related linear model; that is accomplished here on lines 2 to 5. The parameters alpha, beta and gamma could be set to any initial values (not necessarily based on a model estimated with OLS), although convergence of the NLS procedure is not guaranteed for an arbitrary starting point.

The actual NLS commands occupy lines 6 to 10. On line 6 the nls command is given: a dependent variable is specified, followed by an equals sign, followed by a function specification. The syntax for the expression on the right is the same as that for the genr command. The next three lines specify the derivatives of the regression function with respect to each of the parameters in turn. Each line begins with the keyword deriv, gives the name of a parameter, an equals sign, and an expression whereby the derivative can be calculated (again, the syntax here is the same as for genr). These deriv lines are optional, but it is recommended that you supply them if possible. Line 10, end nls, completes the command and calls for estimation.

For further details on NLS estimation please see the *Gretl User's Guide*.

Menu path: /Model/Nonlinear Least Squares

noecho

Obsolete command. See set.

nulldata

Argument:	*series-length*
Example:	`nulldata 500`

Establishes a "blank" data set, containing only a constant and an index variable, with periodicity 1 and the specified number of observations. This may be used for simulation purposes: some of the `genr` commands (e.g. `genr uniform()`, `genr normal()`) will generate dummy data from scratch to fill out the data set. This command may be useful in conjunction with `loop`. See also the "seed" option to the set command.

Menu path: /File/Create data set

ols

Arguments:	*depvar indepvars*
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)
	`--quiet` (suppress printing of results)
	`--no-df-corr` (suppress degrees of freedom correction)
	`--print-final` (see below)
Examples:	`ols 1 0 2 4 6 7`
	`ols y 0 x1 x2 x3 --vcv`
	`ols y 0 x1 x2 x3 --quiet`

Computes ordinary least squares (OLS) estimates with *depvar* as the dependent variable and *indepvars* as the list of independent variables. Variables may be specified by name or number; use the number zero for a constant term.

Besides coefficient estimates and standard errors, the program also prints p-values for t (two-tailed) and F-statistics. A p-value below 0.01 indicates statistical significance at the 1 percent level and is marked with ***. ** indicates significance between 1 and 5 percent and * indicates significance between the 5 and 10 percent levels. Model selection statistics (the Akaike Information Criterion or AIC and Schwarz's Bayesian Information Criterion) are also printed. The formula used for the AIC is that given by Akaike (1974), namely minus two times the maximized log-likelihood plus two times the number of parameters estimated.

If the option `--no-df-corr` is given, the usual degrees of freedom correction is not applied when calculating the estimated error variance (and hence also the standard errors of the parameter estimates).

The option `--print-final` is applicable only in the context of a loop. It arranges for the regression to be run silently on all but the final iteration of the loop. See the *Gretl User's Guide* for details.

Various internal variables may be retrieved using the genr command, provided `genr` is invoked immediately after this command.

The specific formula used for generating robust standard errors (when the `--robust` option is given) can be adjusted via the set command.

Menu path: /Model/Ordinary Least Squares

Other access: Beta-hat button on toolbar

omit

Argument: *varlist*
Options: `--vcv` (print covariance matrix)
`--quiet` (don't print estimates for reduced model)
`--silent` (don't print anything)
`--inst` (omit as instrument, TSLS only)
`--both` (omit as both regressor and instrument, TSLS only)
Examples:
```
omit 5 7 9
omit seasonals --quiet
```

This command must follow an estimation command. The selected variables are omitted from the previous model and the new model estimated. A test statistic for the joint significance of the omitted variables is printed, along with its p-value. The test statistic is F in the case of OLS estimation, an asymptotic Wald chi-square value otherwise. A p-value below 0.05 means that the coefficients are jointly significant at the 5 percent level.

If the `--quiet` option is given the printed results are confined to the test for the joint significance of the omitted variables, otherwise the estimates for the reduced model are also printed. In the latter case, the `--vcv` flag causes the covariance matrix for the coefficients to be printed also. If the `--silent` option is given, nothing is printed; nonetheless, the results of the test can be retrieved using the special variables `$test` and `$pvalue`.

If the original model was estimated using two-stage least squares, an ambiguity arises: should the selected variables be omitted as regressors, as instruments, or as both? This is resolved as follows: by default the variables are dropped from the list of regressors, but if the `--inst` flag is given they are dropped as instruments, or if the `--both` flag is given they are dropped from the model altogether.

Menu path: Model window, /Tests/omit variables

omitfrom

Arguments: *modelID varlist*
Option: `--quiet` (don't print estimates for reduced model)
Example:
```
omitfrom 2 5 7 9
```

Works like omit, except that you specify a previous model (using its ID number, which is printed at the start of the model output) to take as the base for omitting variables. The example above omits variables number 5, 7 and 9 from Model 2.

Menu path: Model window, /Tests/omit variables

open

Argument: *datafile*

Opens a data file. If a data file is already open, it is replaced by the newly opened one. The program will try to detect the format of the data file (native, plain text, CSV or BOX1).

This command can also be used to open a database (gretl or RATS 4.0) for reading. In that case it should be followed by the data command to extract particular series from the database.

Menu path: /File/Open data

Other access: Drag a data file into gretl (MS Windows or Gnome)

outfile

Arguments:	*filename option*
Options:	`--append` (append to file)
	`--close` (close file)
	`--write` (overwrite file)
Examples:	`outfile --write regress.txt`
	`outfile --close`

Diverts output to *filename*, until further notice. Use the flag `--append` to append output to an existing file or `--write` to start a new file (or overwrite an existing one). Only one file can be opened in this way at any given time.

The `--close` flag is used to close an output file that was previously opened as above. Output will then revert to the default stream.

In the first example command above, the file `regress.txt` is opened for writing, and in the second it is closed. This would make sense as a sequence only if some commands were issued before the `--close`. For example if an `ols` command intervened, its output would go to `regress.txt` rather than the screen.

panel

Options:	`--cross-section` (stacked cross sections)
	`--time-series` (stacked time series)

Request that the current data set be interpreted as a panel (pooled cross section and time series). By default, or with the `--time-series` flag, the data set is taken to be in the form of stacked time series (successive blocks of data contain time series for each cross-sectional unit). With the `--cross-section` flag, the data set is read as stacked cross-sections (successive blocks contain cross sections for each time period). See also setobs.

pca

Argument:	*varlist*
Options:	`--save` (Save major components)
	`--save-all` (Save all components)

Principal Components Analysis. Prints the eigenvalues of the correlation matrix for the variables in *varlist* along with the proportion of the joint variance accounted for by each component. Also prints the corresponding eigenvectors (or "component loadings").

If the `--save` flag is given, components with eigenvalues greater than 1.0 are saved to the dataset as variables, with names `PC1`, `PC2` and so on. These artificial variables are formed as the sum of (component loading) times (standardized `Xi`), where `Xi` denotes the *i*th variable in *varlist*.

If the `--save-all` flag is given, all of the components are saved as described above.

Menu path: Main window pop-up (multiple selection)

pergm

Argument:	*varname*
Option:	`--bartlett` (use Bartlett lag window)

Computes and displays (and if not in batch mode, graphs) the spectrum of the specified variable. Without the `--bartlett` flag the sample periodogram is given; with the flag a Bartlett lag window of length two times the square root of the sample size is used in estimating the spectrum (see Chapter 18 of Greene's *Econometric Analysis*).

When the sample periodogram is printed, a t-test for fractional integration of the series ("long memory") is also given: the null hypothesis is that the integration order is zero.

Menu path: /Variable/spectrum

Other access: Main window pop-up menu (single selection)

poisson

Arguments:	*depvar indepvars* `[ ; offset ]`
Options:	`--vcv` (print covariance matrix)
	`--verbose` (print details of iterations)
Examples:	`poisson y 0 x1 x2`
	`poisson y 0 x1 x2 ; S`

Estimates a poisson regression. The dependent variable is taken to represent the occurrence of events of some sort, and must take on only non-negative integer values.

If a discrete random variable Y follows the Poisson distribution, then

$$\Pr(Y = y) = \frac{e^{-v} v^y}{y!}$$

for $y = 0, 1, 2, \ldots$. The mean and variance of the distribution are both equal to v. In the Poisson regression model, the parameter v is represented as a function of one or more independent variables. The most common version (and the only one supported by gretl) has

$$v = \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots)$$

or in other words the log of v is a linear function of the independent variables.

Optionally, you may add an "offset" variable to the specification. This is a scale variable, the log of which is added to the linear regression function (implicitly, with a coefficient of 1.0). This makes sense if you expect the number of occurrences of the event in question to be proportional, other things equal, to some known factor. For example, the number of traffic accidents might be supposed to be proportional to traffic volume, other things equal, and in that case traffic volume could be specified as an "offset" in a Poisson model of the accident rate. The offset variable must be strictly positive.

Menu path: /Model/Poisson

plot

Argument:	*varlist*
Option:	`--one-scale` (force a single scale)

Plots the values for specified variables, for the range of observations currently in effect, using ASCII symbols. Each line stands for an observation and the values are plotted horizontally. By default the variables are scaled appropriately. See also gnuplot.

pooled

Arguments:	*depvar indepvars*
Options:	`--unit-weights` (feasible GLS)
	`--iterate` (iterate to Maximum Likelihood solution)
	`--vcv` (print covariance matrix)

By default, estimates a model via OLS (see ols for details on syntax), and flags it as a pooled or panel model, so that the hausman test item becomes available.

If the `--unit-weights` flag is given, estimation is by feasible GLS, using weights constructed from the specific error variances per cross-sectional unit. This offers a gain in efficiency over OLS if the error variance differs across units.

If, in addition, the `--iterate` flag is given, the GLS estimator is iterated: if this procedure converges it yields Maximum Likelihood estimates.

Menu path: /Model/Pooled OLS

print

Arguments:	*varlist* or *string-literal*
Options:	`--byobs` (by observations)
	`--ten` (use 10 significant digits)
	`--no-dates` (use simple observation numbers)
Examples:	`print x1 x2 --byobs`
	`print "This is a string"`

If *varlist* is given, prints the values of the specified variables; if no list is given, prints the values of all variables in the current data file. If the `--byobs` flag is given the data are printed by observation, otherwise they are printed by variable. If the `--ten` flag is given the data are printed by variable to 10 significant digits.

If the `--byobs` flag is given and the data are printed by observation, the default is to show the date (with time-series data) or the observation marker string (if any) at the start of each line. The `--no-dates` option suppresses the printing of dates or markers; a simple observation number is shown instead.

If the argument to `print` is a literal string (which must start with a double-quote, "), the string is printed as is. See also printf.

Menu path: /Data/Display values

printf

Arguments: *format args*

Prints scalar values under the control of a format string (providing a small subset of the `printf()` statement in the C programming language). Recognized formats are `%g` and `%f`, in each case with the various modifiers available in C. Examples: the format `%.10g` prints a value to 10 significant figures; `%12.6f` prints a value to 6 decimal places, with a width of 12 characters.

The format string itself must be enclosed in double quotes. The values to be printed must follow the format string, separated by commas. These values should take the form of either (a) the names of variables in the dataset, (b) expressions that are valid for the `genr` command, or (c) the special functions `varname()` or `date()`. The following example prints the values of two variables plus that of a calculated expression:

```
ols 1 0 2 3
genr b = coeff(2)
genr se_b = stderr(2)
printf "b = %.8g, standard error %.8g, t = %.4f\n", b, se_b, b/se_b
```

The next lines illustrate the use of the varname and date functions, which respectively print the name of a variable, given its ID number, and a date string, given a 1-based observation number.

```
printf "The name of variable %d is %s\n", i, varname(i)
printf "The date of observation %d is %s\n", j, date(j)
```

The maximum length of a format string is 127 characters. The escape sequences \n (newline), \t (tab), \v (vertical tab) and \\ (literal backslash) are recognized. To print a literal percent sign, use %%.

probit

Arguments: *depvar indepvars*
Options: --robust (robust standard errors)
--vcv (print covariance matrix)

The dependent variable should be a binary variable. Maximum likelihood estimates of the coefficients on *indepvars* are obtained via the "binary response model regression" (BRMR) method outlined by Davidson and MacKinnon (2004). As the model is nonlinear the slopes depend on the values of the independent variables: the reported slopes are evaluated at the means of those variables. The chi-square statistic tests the null hypothesis that all coefficients are zero apart from the constant.

By default, standard errors are computed using the negative inverse of the Hessian. If the --robust flag is given, then QML or Huber–White standard errors are calculated instead. In this case the estimated covariance matrix is a "sandwich" of the inverse of the estimated Hessian and the outer product of the gradient. See Davidson and MacKinnon (2004, Chapter 10) for details.

Probit for analysis of proportions is not implemented in gretl at this point.

Menu path: /Model/Probit

pvalue

Arguments: *dist* [*params*] *xval*
Examples:
```
pvalue z zscore
pvalue t 25 3.0
pvalue X 3 5.6
pvalue F 4 58 fval
pvalue G xbar varx x
pvalue B bprob 10 6
```

Computes the area to the right of *xval* in the specified distribution (z for Gaussian, t for Student's *t*, X for chi-square, F for *F*, G for gamma, or B for binomial).

For the *t* and chi-square distributions the degrees of freedom must be given; for *F* numerator and denominator degrees of freedom are required; for gamma the mean and variance are needed; and for the binomial distribution the "success" probability and the number of trials must be given. In each case, these extra values are provided before the *xval*.

As shown in the examples above, the numerical parameters may be given in numeric form or as the names of variables.

Menu path: /Utilities/p-value finder

pwe

Arguments: *depvar indepvars*
Option: --vcv (print covariance matrix)
Example: pwe 1 0 2 4 6 7

Computes parameter estimates using the Prais–Winsten procedure, an implementation of feasible GLS which is designed to handle first-order autocorrelation of the error term. The procedure is iterated, as with corc; the difference is that while Cochrane–Orcutt discards the first observation,

Prais–Winsten makes use of it. See, for example, Chapter 13 of Greene's *Econometric Analysis* (2000) for details.

Menu path: /Model/Time series/Prais-Winsten

quit

Exits from the program, giving you the option of saving the output from the session on the way out.

Menu path: /File/Exit

rename

Arguments: *varnumber newname*

Changes the name of the variable with identification number *varnumber* to *newname*. The *varnumber* must be between 1 and the number of variables in the dataset. The new name must be of 8 characters maximum, must start with a letter, and must be composed of only letters, digits, and the underscore character.

Menu path: /Variable/Edit attributes

Other access: Main window pop-up menu (single selection)

reset

Must follow the estimation of a model via OLS. Carries out Ramsey's RESET test for model specification (non-linearity) by adding the square and the cube of the fitted values to the regression and calculating the F statistic for the null hypothesis that the parameters on the two added terms are zero.

Menu path: Model window, /Tests/Ramsey's RESET

restrict

Imposes a set of linear restrictions on either (a) the model last estimated or (b) a system of equations previously defined and named. The syntax and effects of the command differ slightly in the two cases.

In both cases the set of restrictions should be started with the keyword "restrict" and terminated with "end restrict". In the single equation case the restrictions are implicitly to be applied to the last model, and they are evaluated as soon as the `restrict` command is terminated. In the system case the initial "restrict" must be followed by the name of a previously defined system of equations (see system). The restrictions are evaluated when the system is next estimated, using the estimate command.

Each restriction in the set should be expressed as an equation, with a linear combination of parameters on the left and a numeric value to the right of the equals sign. In the single-equation case, parameters are referenced in the form b*N*, where *N* represents the position in the list of regressors, starting at zero. For example, `b1` denotes the second regression parameter. In the system case, parameters are referenced using `b` plus two numbers in square brackets. The leading number represents the position of the equation within the system, starting from 1, and the second number indicates position in the list of regressors, starting at zero. For example `b[2,0]` denotes the first parameter in the second equation, and `b[3,1]` the second parameter in the third equation.

The `b` terms in the equation representing a restriction equation may be prefixed with a numeric multiplier, using `*` to represent multiplication, for example `3.5*b4`.

Here is an example of a set of restrictions for a previously estimated model:

```
restrict
 b1 = 0
 b2 - b3 = 0
 b4 + 2*b5 = 1
end restrict
```

And here is an example of a set of restrictions to be applied to a named system. (If the name of the system does not contain spaces, the surrounding quotes are not required.)

```
restrict "System 1"
 b[1,1] = 0
 b[1,2] - b[2,2] = 0
 b[3,4] + 2*b[3,5] = 1
end restrict
```

In the single-equation case the restrictions are evaluated via a Wald *F*-test, using the coefficient covariance matrix of the model in question. In the system case, the full results of estimating the system subject to the restrictions are shown; the test statistic depends on the estimator chosen (a Likelihood Ratio test if the system is estimated using a Maximum Likelihood method, or an asymptotic *F*-test otherwise.)

Menu path: Model window, /Tests/linear restrictions

rhodiff

Arguments: *rholist* ; *varlist*
Examples: `rhodiff .65 ; 2 3 4`
`rhodiff r1 r2 ; x1 x2 x3`

Creates rho-differenced counterparts of the variables (given by number or by name) in *varlist* and adds them to the data set, using the suffix # for the new variables. Given variable `v1` in *varlist*, and entries `r1` and `r2` in *rholist*, the new variable

```
v1# = v1 - r1*v1(-1) - r2*v1(-2)
```

is created. The *rholist* entries can be given as numerical values or as the names of variables previously defined.

rmplot

Argument: *varname*

Range–mean plot: this command creates a simple graph to help in deciding whether a time series, *y*(t), has constant variance or not. We take the full sample t=1,...,T and divide it into small subsamples of arbitrary size *k*. The first subsample is formed by *y*(1),...,*y*(k), the second is *y*(k+1), ..., *y*(2k), and so on. For each subsample we calculate the sample mean and range (= maximum minus minimum), and we construct a graph with the means on the horizontal axis and the ranges on the vertical. So each subsample is represented by a point in this plane. If the variance of the series is constant we would expect the subsample range to be independent of the subsample mean; if we see the points approximate an upward-sloping line this suggests the variance of the series is increasing in its mean; and if the points approximate a downward sloping line this suggests the variance is decreasing in the mean.

Besides the graph, gretl displays the means and ranges for each subsample, along with the slope coefficient for an OLS regression of the range on the mean and the p-value for the null hypothesis that this slope is zero. If the slope coefficient is significant at the 10 percent significance level then the fitted line from the regression of range on mean is shown on the graph.

Menu path: /Variable/Range-mean graph

run

Argument: *inputfile*

Execute the commands in *inputfile* then return control to the interactive prompt. This command is intended for use with the command-line program gretlcli, or at the "gretl console" in the GUI program.

See also include.

Menu path: Run icon in script window

runs

Argument: *varname*

Carries out the nonparametric "runs" test for randomness of the specified variable. If you want to test for randomness of deviations from the median, for a variable named `x1` with a non-zero median, you can do the following:

```
genr signx1 = x1 - median(x1)
runs signx1
```

Menu path: /Variable/Runs test

scatters

Arguments: *yvar* ; *xvarlist* or *yvarlist ; xvar*
Examples:
```
scatters 1 ; 2 3 4 5
scatters 1 2 3 4 5 6 ; 7
```

Plots pairwise scatters of *yvar* against all the variables in *xvarlist*, or of all the variables in *yvarlist* against *xvar*. The first example above puts variable 1 on the *y*-axis and draws four graphs, the first having variable 2 on the *x*-axis, the second variable 3 on the *x*-axis, and so on. The second example plots each of variables 1 through 6 against variable 7 on the *x*-axis. Scanning a set of such plots can be a useful step in exploratory data analysis. The maximum number of plots is six; any extra variable in the list will be ignored.

Menu path: /Data/Multiple scatterplots

sdiff

Argument: *varlist*

The seasonal difference of each variable in *varlist* is obtained and the result stored in a new variable with the prefix `sd_`. This command is available only for seasonal time series.

Menu path: /Data/Add variables/seasonal differences

seed

Obsolete command. See set.

set

Arguments: *variable value*
Examples:
```
set qr on
set csv_delim tab
set horizon 10
```

Set the values of various program parameters. The given value remains in force for the duration of the gretl session unless it is changed by a further call to `set`. The parameters that can be set in this way are enumerated below. Note that the settings of `hac_lag` and `hc_version` are used when the `--robust` option is given to the `ols` command.

- `echo`: `off` or `on` (the default). Suppress or resume the echoing of commands in gretl's output.
- `qr`: `on` or `off` (the default). Use QR rather than Cholesky decomposition in calculating OLS estimates.
- `seed`: an unsigned integer. Sets the seed for the pseudo-random number generator. By default this is set from the system time; if you want to generate repeatable sequences of random numbers you must set the seed manually.
- `hac_lag`: `nw1` (the default) or `nw2`, or an integer. Sets the maximum lag value, p, used when calculating HAC (Heteroskedasticity and Autocorrelation Consistent) standard errors using the Newey-West approach, for time series data. `nw1` and `nw2` represent two variant automatic calculations based on the sample size, T: for nw1, $p = 0.75 \times T^{1/3}$, and for nw2, $p = 4 \times (T/100)^{2/9}$.
- `hc_version`: 0 (the default), 1, 2 or 3. Sets the variant used when calculating Heteroskedasticity Consistent standard errors with cross-sectional data. The options correspond to the HC0, HC1, HC2 and HC3 discussed by Davidson and MacKinnon in *Econometric Theory and Methods*, chapter 5. HC0 produces what are usually called "White's standard errors".
- `force_hc`: `off` (the default) or `on`. By default, with time-series data and when the `--robust` option is given with `ols`, the HAC estimator is used. If you set `force_hc` to "on", this forces calculation of the regular Heteroskedasticity Consistent Covariance Matrix (which does not take autocorrelation into account).
- `garch_vcv`: `unset`, `hessian`, `im` (information matrix) , `op` (outer product matrix), `qml` (QML estimator), `bw` (Bollerslev–Wooldridge). Specifies the variant that will be used for estimating the coefficient covariance matrix, for GARCH models. If `unset` is given (the default) then the Hessian is used unless the "robust" option is given for the garch command, in which case QML is used.
- `hp_lambda`: `auto` (the default), or a numerical value. Sets the smoothing parameter for the Hodrick–Prescott filter (see the `hpfilt` function under the `genr` command). The default is to use 100 times the square of the periodicity, which gives 100 for annual data, 1600 for quarterly data, and so on.
- `bkbp_limits`: two integers, the second greater than the first (the defaults are 8 and 32). Sets the frequency bounds for the Baxter–King bandpass filter (see the `bkfilt` function under the `genr` command).
- `bkbp_k`: one integer (the default is 8). Sets the approximation order for the Baxter–King bandpass filter.
- `horizon`: one integer (the default is based on the frequency of the data). Sets the horizon for impulse responses and forecast variance decompositions in the context of vector autoregressions.
- `csv_delim`: either `comma` (the default), `space` or `tab`. Sets the column delimiter used when saving data to file in CSV format.

setobs

Arguments:	*periodicity startobs*
Options:	`--cross-section` (interpret as cross section)
	`--time-series` (interpret as time series)
	`--stacked-cross-section` (interpret as panel data)
	`--stacked-time-series` (interpret as panel data)
Examples:	`setobs 4 1990:1 --time-series`
	`setobs 12 1978:03`
	`setobs 1 1 --cross-section`
	`setobs 20 1:1 --stacked-time-series`

Force the program to interpret the current data set as having a specified structure.

The *periodicity*, which must be an integer, represents frequency in the case of time-series data (1 = annual; 4 = quarterly; 12 = monthly; 52 = weekly; 5, 6, or 7 = daily; 24 = hourly). In the case of panel data the periodicity means the number of lines per data block: this corresponds to the number of cross-sectional units in the case of stacked cross-sections, or the number of time periods in the case of stacked time series. In the case of simple cross-sectional data the periodicity should be set to 1.

The starting observation represents the starting date in the case of time series data. Years may be given with two or four digits; subperiods (for example, quarters or months) should be separated from the year with a colon. In the case of panel data the starting observation should be given as 1:1; and in the case of cross-sectional data, as 1. Starting observations for daily or weekly data should be given in the form YY/MM/DD or YYYY/MM/DD (or simply as 1 for undated data).

If no explicit option flag is given to indicate the structure of the data the program will attempt to guess the structure from the information given.

Menu path: /Sample/Dataset structure

setmiss

Arguments:	*value* [*varlist*]
Examples:	`setmiss -1`
	`setmiss 100 x2`

Get the program to interpret some specific numerical data value (the first parameter to the command) as a code for "missing", in the case of imported data. If this value is the only parameter, as in the first example above, the interpretation will be applied to all series in the data set. If *value* is followed by a list of variables, by name or number, the interpretation is confined to the specified variable(s). Thus in the second example the data value 100 is interpreted as a code for "missing", but only for the variable `x2`.

Menu path: /Sample/Set missing value code

shell

Argument:	*shellcommand*
Examples:	`! ls -al`
	`! notepad`

A ! at the beginning of a command line is interpreted as an escape to the user's shell. Thus arbitrary shell commands can be executed from within gretl.

sim

Arguments: `[` *startobs endobs* `]` *varname a0 a1 a2 ...*
Examples:
```
sim 1979.2 1983.1 y 0 0.9
sim 15 25 y 10 0.8 x
```

Simulates values for *varname* for the current sample range, or for the range *startobs* through *endobs* if these optional arguments are given. The variable *y* must have been defined earlier with appropriate initial values. The formula used is

$$y_t = a_{0t} + a_{1t}y_{t-1} + a_{2t}y_{t-2} + \ldots$$

The `ai(t)` terms may be either numerical constants or variable names previously defined; these terms may be prefixed with a minus sign.

This command is deprecated. You should use genr instead.

smpl

Variants:
```
smpl startobs endobs
smpl +i -j
smpl dumvar --dummy
smpl condition --restrict
smpl --no-missing [ varlist ]
smpl n --random
smpl full
```
Examples:
```
smpl 3 10
smpl 1960:2 1982:4
smpl +1 -1
smpl x > 3000 --restrict
smpl y > 3000 --restrict --replace
smpl 100 --random
```

Resets the sample range. The new range can be defined in several ways. In the first alternate form (and the first two examples) above, *startobs* and *endobs* must be consistent with the periodicity of the data. Either one may be replaced by a semicolon to leave the value unchanged. In the second form, the integers *i* and *j* (which may be positive or negative, and should be signed) are taken as offsets relative to the existing sample range. In the third form *dummyvar* must be an indicator variable with values 0 or 1 at each observation; the sample will be restricted to observations where the value is 1. The fourth form, using `--restrict`, restricts the sample to observations that satisfy the given Boolean condition (which is specified according to the syntax of the genr command).

With the `--no-missing` form, if *varlist* is specified observations are selected on condition that all variables in *varlist* have valid values at that observation; otherwise, if no *varlist* is given, observations are selected on condition that *all* variables have valid (non-missing) values.

With the `--random` flag, the specified number of cases are selected from the full dataset at random. If you wish to be able to replicate this selection you should set the seed for the random number generator first (see the set command).

The final form, `smpl full`, restores the full data range.

Note that sample restrictions are, by default, cumulative: the baseline for any `smpl` command is the current sample. If you wish the command to act so as to replace any existing restriction you can add the option flag `--replace` to the end of the command.

The internal variable `obs` may be used with the `--restrict` form of `smpl` to exclude particular observations from the sample. For example

```
smpl obs!=4 --restrict
```

will drop just the fourth observation. If the data points are identified by labels,

```
smpl obs!="USA" --restrict
```

will drop the observation with label "USA".

One point should be noted about the `--dummy`, `--restrict` and `--no-missing` forms of `smpl`: Any "structural" information in the data file (regarding the time series or panel nature of the data) is lost when this command is issued. You may reimpose structure with the setobs command.

Please see the *Gretl User's Guide* for further details.

Menu path: /Sample

spearman

Arguments: *x y*
Option: `--verbose` (print ranked data)

Prints Spearman's rank correlation coefficient for the two variables *x* and *y*. The variables do not have to be ranked manually in advance; the function takes care of this.

The automatic ranking is from largest to smallest (i.e. the largest data value gets rank 1). If you need to invert this ranking, create a new variable which is the negative of the original first. For example:

```
genr altx = -x
spearman altx y
```

Menu path: /Model/Rank correlation

square

Argument: *varlist*
Option: `--cross` (generate cross-products as well as squares)

Generates new variables which are squares of the variables in *varlist* (plus cross-products if the `--cross` option is given). For example, `square x y` will generate `sq_x` = `x` squared, `sq_y` = `y` squared and (optionally) `x_y` = `x` times `y`. If a particular variable is a dummy variable it is not squared because we will get the same variable.

Menu path: /Data/Add variables/squares of variables

store

Arguments: *datafile* `[` *varlist* `]`
Options: `--csv` (use CSV format)
`--omit-obs` (see below, on CSV format)
`--gnu-octave` (use GNU Octave format)
`--gnu-R` (use GNU R format)
`--traditional` (use traditional ESL format)
`--gzipped` (apply gzip compression)
`--dat` (use PcGive ASCII format)
`--database` (use gretl database format)
`--overwrite` (see below, on database format)

Saves either the entire dataset or, if a *varlist* is supplied, a specified subset of the variables in the current dataset, to the file given by *datafile*.

By default the data are saved in "native" gretl format, but the option flags permit saving in several alternative formats. CSV (Comma-Separated Values) data may be read into spreadsheet programs, and can also be manipulated using a text editor. The formats of Octave, R and PcGive are designed for use with the respective programs. Gzip compression may be useful for large datasets. See the *Gretl User's Guide* for details on the various formats.

The option flag `--omit-obs` is applicable only when saving data in CSV format. By default, if the data are time series or panel or if the dataset includes specific observation markers, the CSV file includes a first column identifying the observations (e.g. by date). If the `--omit-obs` flag is given this column is omitted; only the actual data are printed.

Note that any scalar variables will not be saved automatically: if you wish to save scalars you must explicitly list them in *varlist*.

The option of saving in gretl database format is intended to help with the construction of large sets of series, possibly having mixed frequencies and ranges of observations. At present this option is available only for annual, quarterly or monthly time-series data. If you save to a file that already exists, the default action is to append the newly saved series to the existing content of the database. In this context it is an error if one or more of the variables to be saved has the same name as a variable that is already present in the database. The `--overwrite` flag has the effect that, if there are variable names in common, the newly saved variable replaces the variable of the same name in the original dataset.

Menu path: /File/Save data; /File/Export data

summary

Argument: `[` *varlist* `]`

Print summary statistics for the variables in *varlist*, or for all the variables in the data set if *varlist* is omitted. Output consists of the mean, standard deviation (sd), coefficient of variation (= sd/mean), median, minimum, maximum, skewness coefficient, and excess kurtosis.

Menu path: /Data/Summary statistics

Other access: Main window pop-up menu

system

Variants: `system method=`*estimator*
`system name=`*sysname*
Argument: *savevars*
Examples:
```
system name="Klein Model 1"
system method=sur
system method=sur save=resids
system method=3sls save=resids,fitted
```

Starts a system of equations. Either of two forms of the command may be given, depending on whether you wish to save the system for estimation in more than one way or just estimate the system once.

To save the system you should give it a name, as in the first example (if the name contains spaces it must be surrounded by double quotes). In this case you estimate the system using the estimate command. With a saved system of equations, you are able to impose restrictions (including cross-equation restrictions) using the restrict command.

Alternatively you can specify an estimator for the system using `method=` followed by a string identifying one of the supported estimators: `ols` (Ordinary Least Squares), `tsls` (Two-Stage Least

Squares) `sur` (Seemingly Unrelated Regressions), `3sls` (Three-Stage Least Squares), `fiml` (Full Information Maximum Likelihood) or `liml` (Limited Information Maximum Likelihood). In this case the system is estimated once its definition is complete.

An equation system is terminated by the line `end system`. Within the system four sorts of statement may be given, as follows.

- equation: specify an equation within the system. At least two such statements must be provided.
- `instr`: for a system to be estimated via Three-Stage Least Squares, a list of instruments (by variable name or number). Alternatively, you can put this information into the `equation` line using the same syntax as in the tsls command.
- `endog`: for a system of simultaneous equations, a list of endogenous variables. This is primarily intended for use with FIML estimation, but with Three-Stage Least Squares this approach may be used instead of giving an `instr` list; then all the variables not identified as endogenous will be used as instruments.
- `identity`: for use with FIML, an identity linking two or more of the variables in the system. This sort of statement is ignored when an estimator other than FIML is used.

In the optional `save=` field of the command you can specify whether to save the residuals (`resids`) and/or the fitted values (`fitted`).

For full examples of the specification and estimation of systems of equations, please see the scripts `klein.inp` and `greene14_2.inp` (supplied with the gretl distribution).

Menu path: /Model/Simultaneous equations

tabprint

Argument: `[` *-f filename* `]`
Option: `--complete` (Create a complete document)

Must follow the estimation of a model. Prints the estimated model in the form of a LaTeX table. If a filename is specified using the `-f` flag output goes to that file, otherwise it goes to a file with a name of the form `model_N.tex`, where `N` is the number of models estimated to date in the current session. See also eqnprint.

If the `--complete` flag is given the LaTeX file is a complete document, ready for processing; otherwise it must be included in a document.

Menu path: Model window, /LaTeX

testuhat

Must follow a model estimation command. Gives the frequency distribution for the residual from the model along with a chi-square test for normality, based on the procedure suggested by Doornik and Hansen (1984).

Menu path: Model window, /Tests/normality of residual

tobit

Arguments: *depvar indepvars*
Options: `--vcv` (print covariance matrix)
`--verbose` (print details of iterations)

Estimates a Tobit model. This model may be appropriate when the dependent variable is "truncated". For example, positive and zero values of purchases of durable goods on the part of individual households are observed, and no negative values, yet decisions on such purchases may be thought of as outcomes of an underlying, unobserved disposition to purchase that may be negative in some cases. For details see Greene's *Econometric Analysis*, Chapter 20.

Menu path: /Model/Tobit

transpos

Transposes the current data set. That is, each observation (row) in the current data set will be treated as a variable (column), and each variable as an observation. This command may be useful if data have been read from some external source in which the rows of the data table represent variables.

Menu path: /Sample/Transpose data

tsls

Arguments:	*depvar indepvars* ; *instruments*
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)
Example:	`tsls y1 0 y2 y3 x1 x2 ; 0 x1 x2 x3 x4 x5 x6`

Computes two-stage least squares (TSLS or IV) estimates: *depvar* is the dependent variable, *indepvars* is the list of independent variables (including right-hand side endogenous variables) in the structural equation for which TSLS estimates are needed; and *instruments* is the combined list of exogenous and predetermined variables in all the equations. If the *instruments* list is not at least as long as *indepvars*, the model is not identified.

In the above example, the `y`s are the endogenous variables and the `x`s are the exogenous and predetermined variables.

Output includes the Hausman test and, if the model is over-identified, the Sargan over-identification test. In the Hausman test, the null hypothesis is that OLS estimates are consistent, or in other words estimation by means of instrumental variables is not required. A model of this sort is over-identified if there are more instruments than are strictly required. The Sargan test is based on an auxiliary regression of the residuals from the two-stage least squares model on the full list of instruments. The null hypothesis is that all the instruments are valid, and suspicion is thrown on this hypothesis if the auxiliary regression has a significant degree of explanatory power. Davidson and MacKinnon (2004, chapter 8) give a good explanation of both tests.

Menu path: /Model/Two-Stage least Squares

var

Arguments:	*order varlist* ; *detlist*
Options:	`--nc` (do not include a constant)
	`--seasonals` (include seasonal dummy variables)
	`--robust` (robust standard errors)
	`--impulse-responses` (print impulse responses)
	`--variance-decomp` (print forecast variance decompositions)
Examples:	`var 4 x1 x2 x3 ; time mydum`
	`var 4 x1 x2 x3 --seasonals`

Sets up and estimates (using OLS) a vector autoregression (VAR). The first argument specifies the lag order, then follows the setup for the first equation. Don't include lags among the elements of

varlist — they will be added automatically. The semi-colon separates the stochastic variables, for which *order* lags will be included, from deterministic or exogenous terms in *detlist*.

In fact, gretl is able to recognize the more common deterministic variables (time trend, dummy variables with no values other than 0 and 1) as such, so these do not have to placed after the semi-colon. More complex deterministic variables (e.g. a time trend interacted with a dummy variable) must be put after the semi-colon. Note that a constant is included automatically unless you give the `--nc` flag, and seaonal dummy variables may be added using the `--seasonals` flag.

A separate regression is run for each variable in *varlist*. Output for each equation includes *F*-tests for zero restrictions on all lags of each of the variables, an *F*-test for the significance of the maximum lag, and, if the `--impulse-responses` flag is given, forecast variance decompositions and impulse responses.

Forecast variance decompositions and impulse responses are based on the Cholesky decomposition of the contemporaneous covariance matrix, and in this context the order in which the (stochastic) variables are given matters. The first variable in the list is assumed to be "most exogenous" within-period. The horizon for variance decompositions and impulse responses can be set using the set command.

Menu path: /Model/Time series/Vector autoregression

varlist

Prints a listing of variables currently available. `list` and `ls` are synonyms.

vartest

Arguments: *var1 var2*

Calculates the *F* statistic for the null hypothesis that the population variances for the variables *var1* and *var2* are equal, and shows its p-value.

Menu path: /Data/Difference of variances

vecm

Arguments:	*order rank varlist*
Options:	`--nc` (no constant)
	`--rc` (restricted constant)
	`--crt` (constant and restricted trend)
	`--ct` (constant and unrestricted trend)
	`--seasonals` (include centered seasonal dummies)
	`--impulse-responses` (print impulse responses)
	`--variance-decomp` (print forecast variance decompositions)
Examples:	`vecm 4 1 Y1 Y2 Y3`
	`vecm 3 2 Y1 Y2 Y3 --rc`

A VECM is a form of vector autoregression or VAR (see var), applicable where the variables in the model are individually integrated of order 1 (that is, are random walks, with or without drift), but exhibit cointegration. This command is closely related to the Johansen test for cointegration (see coint2).

The *order* parameter to this command represents the lag order of the VAR system. The number of lags in the VECM itself (where the dependent variable is given as a first difference) is one less than *order*.

The *rank* parameter represents the cointegration rank, or in other words the number of cointegrating vectors. This must be greater than zero and less than or equal to (generally, less than) the

number of endogenous variables given in *varlist*.

varlist supplies the list of endogenous variables, in levels. The inclusion of deterministic terms in the model is controlled by the option flags. The default if no option is specified is to include an "unrestricted constant", which allows for the presence of a non-zero intercept in the cointegrating relations as well as a trend in the levels of the endogenous variables. In the literature stemming from the work of Johensen (see for example his 1995 book) this is often referred to as "case 3". The first four options given above, which are mutually exclusive, produce cases 1, 2, 4 and 5 respectively. The meaning of these cases and the criteria for selecting a case are explained in the *Gretl User's Guide*.

The `--seasonals` option, which may be combined with any of the other options, specifies the inclusion of a set of centered seasonal dummy variables. This option is available only for quarterly or monthly data.

The first example above specifies a VECM with lag order 4 and a single cointegrating vector. The endogenous variables are `Y1`, `Y2` and `Y3`. The second example uses the same variables but specifies a lag order of 3 and two cointegrating vectors; it also specifies a "restricted constant", which is appropriate if the cointegrating vectors may have a non-zero intercept but the `Y` variables have no trend.

Menu path: /Model/Time series/VECM

vif

Must follow the estimation of a model which includes at least two independent variables. Calculates and displays the Variance Inflation Factors (VIFs) for the regressors. The VIF for regressor j is defined as

$$\frac{1}{1 - R_j^2}$$

where R_j is the coefficient of multiple correlation between regressor j and the other regressors. The factor has a minimum value of 1.0 when the variable in question is orthogonal to the other independent variables. Neter, Wasserman, and Kutner (1990) suggest inspecting the largest VIF as a diagnostic for collinearity; a value greater than 10 is sometimes taken as indicating a problematic degree of collinearity.

Menu path: Model window, /Tests/collinearity

wls

Arguments:	*wtvar depvar indepvars*
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)

Computes weighted least squares estimates using *wtvar* as the weight, *depvar* as the dependent variable, and *indepvars* as the list of independent variables. Specifically, an OLS regression is run on *wtvar* * *depvar* against *wtvar* * *indepvars*. If the *wtvar* is a dummy variable, this is equivalent to eliminating all observations with value zero for *wtvar*.

Menu path: /Model/Weighted Least Squares

1.3 Estimators and tests: summary

Table 1.2 shows the estimators available under the Model menu in gretl's main window. The corresponding script command (if there is one available) is shown in parentheses. For details consult the command's entry in chapter 1.

Table 1.3 shows the tests that are available under the Tests menu in a model window, after estimation.

Table 1.2: Estimators

Estimator	*Comment*
Ordinary Least Squares (`ols`)	The workhorse estimator
Weighted Least Squares (`wls`)	Heteroskedasticity, exclusion of selected observations
HCCM (`hccm`)	Heteroskedasticity corrected covariance matrix
Heteroskedasticity corrected (`hsk`)	Weighted Least Squares based on predicted error variance
Cochrane–Orcutt (`corc`)	First-order autocorrelation
Hildreth–Lu (`hilu`)	First-order autocorrelation
Prais–Winsten (`pwe`)	First-order autocorrelation
Autoregressive Estimation (`ar`)	Higher-order autocorrelation (generalized Cochrane–Orcutt)
ARMAX (`arma`)	Time-series model with ARMA error
GARCH (`garch`)	Generalized Autoregressive Conditional Heteroskedasticity
Vector Autoregression (`var`)	Systems of time-series equations
Cointegration test (`coint`)	Long-run relationships between series
Two-Stage Least Squares (`tsls`)	Simultaneous equations
Nonlinear Least Squares (`nls`)	Nonlinear models
Logit (`logit`)	Binary dependent variable (logistic distribution)
Probit (`probit`)	Binary dependent variable (normal distribution)
Tobit (`tobit`)	Truncated dependent variable
Logistic (`logistic`)	OLS, with logistic transformation of dependent variable
Least Absolute Deviation (`lad`)	Alternative to Least Squares
Rank Correlation (`spearman`)	Correlation with ordinal data
Pooled OLS (`pooled`)	OLS estimation for pooled cross-section, time series data
Multiple precision OLS (`mpols`)	OLS estimation using multiple precision arithmetic

Table 1.3: Tests for models

Test	*Corresponding command*
Omit variables (*F*-test if OLS)	`omit`
Add variables (*F*-test if OLS)	`add`
Sum of coefficients (*t*-test if OLS)	`coeffsum`
Nonlinearity (squares)	`lmtest --squares`
Nonlinearity (logs)	`lmtest --logs`
Nonlinearity (Ramsey's RESET)	`reset`
Heteroskedasticity (White's test)	`lmtest --white`
Influential observations	`leverage`
Autocorrelation up to the data frequency	`lmtest --autocorr`
Chow (structural break)	`chow`
CUSUM (parameter stability)	`cusum`
ARCH (conditional heteroskedasticity)	`arch`
Normality of residual	`testuhat`
Panel diagnostics	`hausman`

Chapter 2

Options, arguments and path-searching

2.1 gretl

`gretl` (under MS Windows, `gretlw32.exe`)[1]

— Opens the program and waits for user input.

`gretl` *datafile*

— Starts the program with the specified datafile in its workspace. The data file may be in native gretl format, CSV format, or BOX1 format (see the *Gretl User's Guide*). The program will try to detect the format of the file and treat it appropriately. See also Section 2.3 below for path-searching behavior.

`gretl --help` (or `gretl -h`)

— Print a brief summary of usage and exit.

`gretl --version` (or `gretl -v`)

— Print version identification for the program and exit.

`gretl --english` (or `gretl -e`)

— Force use of English instead of translation.

`gretl --run` *scriptfile* (or `gretl -r` *scriptfile*)

— Start the program and open a window displaying the specified script file, ready to run. See Section 2.3 below for path-searching behavior.

`gretl --db` *database* (or `gretl -d` *database*)

— Start the program and open a window displaying the specified database. If the database files (the `.bin` file and its accompanying `.idx` file) are not in the default system database directory, you must specify the full path. See also the *Gretl User's Guide* for details on databases.

`gretl --dump` (or `gretl -c`)

— Dump the program's configuration information to a plain text file (the name of the file is printed on standard output). May be useful for trouble-shooting.

Various things in gretl are configurable under the "File, Preferences" menu.

- The base directory for gretl's shared files.
- The user's base directory for gretl-related files.
- The command to launch gnuplot.
- The command to launch GNU R.
- The command with which to view TEX DVI files.
- The directory in which to start looking for native gretl databases.

[1] On Linux, a "wrapper" script named `gretl` is installed. This script checks whether the `DISPLAY` environment variable is set; if so, it launches the GUI program, `gretl_x11`, and if not it launches the command-line program, `gretlcli`.

- The directory in which to start looking for RATS 4 databases.
- The host name of the gretl database server to access.
- The IP number and port number of the HTTP proxy server to use when contacting the database server, if applicable (if you're behind a firewall).
- The calculator and editor programs to launch from the toolbar.
- The monospaced font to be used in gretl screen output.
- The font to be used for menus and other messages. (Note: this item is not present when gretl is compiled for the gnome desktop, since the choice of fonts is handled centrally by gnome.)

There are also some check boxes. Checking the "expert" box quells some warnings that are otherwise issued. Checking "Tell me about gretl updates" makes gretl attempt to query the update server at start-up. Unchecking "Show gretl toolbar" turns the icon toolbar off. If your native language setting is not English and the local decimal point character is not the period ("`.`"), unchecking "Use locale setting for decimal point" will make gretl use the period regardless.

Finally, there are some binary choices: Under the "Open/Save path" tab you can set where gretl looks by default when you go to open or save a file — either the gretl user directory or the current working directory. Under the "Data files" tab you can set the default filename suffix for data files. The standard suffix is `.gdt` but if you wish you can set this to `.dat`, which was standard in earlier versions of the program. If you set the default to `.dat` then data files will be saved in the "traditional" format (see the *Gretl User's Guide*). Also under the "Data files" tab you can select the action for the little folder icon on the toolbar: whether it should open a listing of the data files associated with Ramanathan's textbook, or those associated with Wooldridge's text.

Under the "General" tab you may select the algorithm used by gretl for calculating least squares estimates. The default is Cholesky decomposition, which is fast, relatively economical in terms of memory requirements, and accurate enough for most purposes. The alternative is QR decomposition, which is computationally more expensive and requires more temporary storage, but which is more accurate. You are unlikely to need the extra accuracy of QR decomposition unless you are dealing with very ill-conditioned data and are concerned with coefficient or standard error values to more than 7 digits of precision.[2]

Settings chosen in this way are handled differently depending on the context. Under MS Windows they are stored in the Windows registry. Under the gnome desktop they are stored in `.gnome/gretl` in the user's home directory. Otherwise they are stored in a file named `.gretlrc` in the user's home directory.

2.2 gretlcli

`gretlcli`

— Opens the program and waits for user input.

`gretlcli` *datafile*

— Starts the program with the specified datafile in its workspace. The data file may be in any format supported by gretl (see the *Gretl User's Guide* for details). The program will try to detect the format of the file and treat it appropriately. See also Section 2.3 for path-searching behavior.

`gretlcli --help` (or `gretlcli -h`)

— Prints a brief summary of usage.

`gretlcli --version` (or `gretlcli -v`)

[2] The option of using QR decomposition can also be activated by setting the environment variable `GRETL_USE_QR` to any non-NULL value.

— Prints version identification for the program.

`gretlcli --pvalue` (or `gretlcli -p`)

— Starts the program in a mode in which you can interactively determine p-values for various common statistics.

`gretlcli --english` (or `gretlcli -e`)

— Force use of English instead of translation.

`gretlcli --run` *scriptfile* (or `gretlcli -r` *scriptfile*)

— Execute the commands in *scriptfile* then hand over input to the command line. See Section 2.3 for path-searching behavior.

`gretlcli --batch` *scriptfile* (or `gretlcli -b` *scriptfile*)

— Execute the commands in *scriptfile* then exit. When using this option you will probably want to redirect output to a file. See Section 2.3 for path-searching behavior.

When using the `--run` and `--batch` options, the script file in question must call for a data file to be opened. This can be done using the `open` command within the script.

2.3 Path searching

When the name of a data file or script file is supplied to gretl or gretlcli on the command line, the file is looked for as follows:

1. "As is". That is, in the current working directory or, if a full path is specified, at the specified location.
2. In the user's gretl directory (see Table 2.1 for the default values).
3. In any immediate sub-directory of the user's gretl directory.
4. In the case of a data file, search continues with the main gretl data directory. In the case of a script file, the search proceeds to the system script directory. See Table 2.1 for the default settings. (`PREFIX` denotes the base directory chosen at the time gretl is installed.)
5. In the case of data files the search then proceeds to all immediate sub-directories of the main data directory.

Table 2.1: Default path settings

	Linux	*MS Windows*
User directory	`$HOME/gretl`	`PREFIX\gretl\user`
System data directory	`PREFIX/share/gretl/data`	`PREFIX\gretl\data`
System script directory	`PREFIX/share/gretl/scripts`	`PREFIX\gretl\scripts`

Thus it is not necessary to specify the full path for a data or script file unless you wish to override the automatic searching mechanism. (This also applies within gretlcli, when you supply a filename as an argument to the `open` or `run` commands.)

When a command script contains an instruction to open a data file, the search order for the data file is as stated above, except that the directory containing the script is also searched, immediately after trying to find the data file "as is".

MS Windows

Under MS Windows configuration information for gretl and gretlcli is stored in the Windows registry. A suitable set of registry entries is created when gretl is first installed, and the settings can be changed under gretl's "File, Preferences" menu. In case anyone needs to make manual adjustments to this information, the entries can be found (using the standard Windows program regedit.exe) under Software\gretl in HKEY_CLASSES_ROOT (the main gretl directory and the command to invoke gnuplot) and HKEY_CURRENT_USER (all other configurable variables).