Gretl Command Reference

Gnu Regression, Econometrics and Time-series Library

Allin Cottrell
Department of Economics
Wake Forest University

Riccardo “Jack” Lucchetti
Dipartimento di Economia
Università Politecnica delle Marche

March, 2025

Contents

Chapter 1

Gretl commands

1.1 Introduction

The commands defined below may be executed interactively in the command-line client program or in the console window of the GUI program. They may also be placed in a "script" or batch file for non-interactive execution.

The following notational 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`.

In general, each line of a command script should contain one and only one complete gretl command. There are, however, two means of continuing a long command from one line of input to another. First, if the last non-space character on a line is a backslash, this is taken as an indication that the command is continued on the following line. In addition, if the comma is a valid character in a given command (for instance, as a separator between function arguments, or as punctuation in the command `printf`) then a trailing comma also indicates continuation. To emphasize the point: a backslash may be inserted "arbitrarily" to indicate continuation, but a comma works in this capacity only if it is syntactically valid as part of the command.

1.2 Commands

add

Argument:	*varlist*
Options:	`--lm` (do an LM test, OLS only)
	`--quiet` (print only the basic test result)
	`--silent` (don't print anything)
	`--vcv` (print covariance matrix for augmented model)
	`--both` (IV estimation only, see below)
Examples:	`add 5 7 9`
	`add xx yy zz --quiet`

Must be invoked after an estimation command. Performs a joint test for the addition of the specified variables to the last model, the results of which may be retrieved using the accessors $test and $pvalue.

By default an augmented version of the original model is estimated, including the variables in *varlist*. The test is a Wald test on the augmented model, which replaces the original as the "current model" for the purposes of, for example, retrieving the residuals as `$uhat` or doing further tests.

Alternatively, given the `--lm` option (available only for the models estimated via OLS), an LM test is performed. An auxiliary regression is run in which the dependent variable is the residual from the last model and the independent variables are those from the last model plus *varlist*. Under the null hypothesis that the added variables have no additional explanatory power, the sample size times the unadjusted R-squared from this regression is distributed as chi-square with degrees of freedom equal to the number of added regressors. In this case the original model is not replaced.

The `--both` option is specific to two-stage least squares: it specifies that the new variables should be added both to the list of regressors and the list of instruments, the default in this case being to add to the regressors only.

Menu path: Model window, /Tests/Add variables

adf

Arguments:	*order varlist*
Options:	`--nc` (test without a constant)
	`--c` (with constant only)
	`--ct` (with constant and trend)
	`--ctt` (with constant, trend and trend squared)
	`--seasonals` (include seasonal dummy variables)
	`--gls` (de-mean or de-trend using GLS)
	`--verbose` (print regression results)
	`--quiet` (suppress printing of results)
	`--difference` (use first difference of variable)
	`--test-down[`=*criterion*`]` (automatic lag order)
	`--perron-qu` (see below)
Examples:	`adf 0 y`
	`adf 2 y --nc --c --ct`
	`adf 12 y --c --test-down`
	See also `jgm-1996.inp`

The options shown above and the discussion which follows mostly pertain to the use of the `adf` command with regular time series data. For use of this command with panel data please see the section titled "Panel data" below.

This command computes a set of Dickey–Fuller tests on each of the listed variables, the null hypothesis being that the variable in question has a unit root. (But if the `--difference` flag is given, the first difference of the variable is taken prior to testing, and the discussion below must be taken as referring to the transformed variable.)

By default, two variants of the test are shown: one based on a regression containing a constant and one using a constant and linear trend. You can control the variants that are presented by specifying one or more of the option flags `--nc`, `--c`, `--ct`, `--ctt`.

The `--gls` option can be used in conjunction with one or other of the flags `--c` and `--ct`. The effect of this option is that the series to be tested is demeaned or detrended using the GLS procedure proposed by Elliott *et al.* (1996), which gives a test of greater power than the standard Dickey–Fuller approach. This option is not compatible with `--nc`, `--ctt` or `--seasonals`.

In all cases the dependent variable in the test regression is the first difference of the specified series, y, and the key independent variable is the first lag of y. The regression is constructed such that the coefficient on lagged y equals the root in question, α, minus 1. For example, the model

with a constant may be written as

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

Under the null hypothesis of a unit root the coefficient on lagged y equals zero. Under the alternative that y is stationary this coefficient is negative. So the test is inherently one-sided.

Selecting the lag order

The simplest version of the Dickey–Fuller test assumes that the error term in the test regression is serially uncorrelated. In practice this is unlikely to be the case and the specification is often extended by including one or more lags of the dependent variable, giving an Augmented Dickey–Fuller (ADF) test. The *order* argument governs the number of such lags, k, possibly depending on the sample size, T.

- For a fixed, user-specified k: give a non-negative value for *order*.
- For T-dependent k: give *order* as -1. The order is then set following the recommendation of Schwert (1989), namely the integer part of $12(T/100)^{0.25}$.

In general, however, we don't know how many lags will be required to "whiten" the Dickey–Fuller residual. It's therefore common to specify the *maximum* value of k and let the data decide the actual number of lags to include. This can be done via the `--test-down` option. The criterion for selecting optimal k may be set using the parameter to this option, which should be one of `AIC`, `BIC` or `tstat`, `AIC` being the default.

When testing down via AIC or BIC, the final lag order for the ADF equation is that which optimizes the chosen information criterion (Akaike or Schwarz Bayesian). The exact procedure depends on whether or not the `--gls` option is given. When GLS is specified, AIC and BIC are the "modified" versions described in Ng and Perron (2001), otherwise they are the standard versions. In the GLS case a refinement is available. If the additional option `--perron-qu` is given, lag-order selection is performed via the revised method recommended by Perron and Qu (2007). In this case the data are first demeaned or detrended via OLS; GLS is applied once the lag order is determined.

When testing down via the t-statistic method is called for, the procedure is as follows:

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 equals 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.

To sum up, if we accept the various arguments of Perron, Ng, Qu and Schwert referenced above, the favored command for testing a series `y` is likely to be:

```
adf -1 y --c --gls --test-down --perron-qu
```

(Or substitute `--ct` for `--c` if the series seems to display a trend.) The lag order for the test will then be determined by testing down via modified AIC from the Schwert maximum, with the Perron–Qu refinement.

P-values for the Dickey–Fuller tests are based on response-surface estimates. When GLS is not applied these are taken from MacKinnon (1996). Otherwise they are taken from Cottrell (2015) or, when testing down is performed, Sephton (2021). The *P*-values are specific to the sample size unless they are labeled as asymptotic.

Panel data

When the `adf` command is used with panel data, to produce a panel unit root test, the applicable options and the results shown are somewhat different.

First, while you may give a list of variables for testing in the regular time-series case, with panel data only one variable may be tested per command. Second, the options governing the inclusion of deterministic terms become mutually exclusive: you must choose between no-constant, constant only, and constant plus trend; the default is constant only. In addition, the `--seasonals` option is not available. Third, the `--verbose` option has a different meaning: it produces a brief account of the test for each individual time series (the default being to show only the overall result).

The overall test (null hypothesis: the series in question has a unit root for all the panel units) is calculated in one or both of two ways: using the method of Im *et al.* (2003) or that of Choi (2001). The Choi test requires that *P*-values are available for the individual tests; if this is not the case (depending on the options selected) it is omitted. The particular statistic given for the Im, Pesaran, Shin test varies as follows: if the lag order for the test is non-zero their *W* statistic is shown; otherwise if the time-series lengths differ by individual, their *Z* statistic; otherwise their *t*-bar statistic. See also the levinlin command.

Menu path: /Variable/Unit root tests/Augmented Dickey-Fuller test

anova

Arguments: *response treatment* [*block*]
Option: `--quiet` (don't print results)

Analysis of Variance: *response* is a series measuring some effect of interest and *treatment* must be a discrete variable that codes for two or more types of treatment (or non-treatment). For two-way ANOVA, the *block* variable (which should also be discrete) codes for the values of some control variable.

Unless the `--quiet` option is given, this command prints a table showing the sums of squares and mean squares along with an *F*-test. The *F*-test and its p-value can be retrieved using the accessors $test and $pvalue respectively.

The null hypothesis for the *F*-test is that the mean response is invariant with respect to the treatment type, or in words that the treatment has no effect. Strictly speaking, the test is valid only if the variance of the response is the same for all treatment types.

Note that the results shown by this command are in fact a subset of the information given by the following procedure, which is easily implemented in gretl. Create a set of dummy variables coding for all but one of the treatment types. For two-way ANOVA, in addition create a set of dummies coding for all but one of the "blocks". Then regress *response* on a constant and the dummies using ols. For a one-way design the ANOVA table is printed via the `--anova` option to `ols`. In the two-way case the relevant *F*-test is found by using the omit command. For example (assuming `y` is the response, `xt` codes for the treatment, and `xb` codes for blocks):

```
# one-way
list dxt = dummify(xt)
ols y 0 dxt --anova
# two-way
list dxb = dummify(xb)
ols y 0 dxt dxb
# test joint significance of dxt
omit dxt --quiet
```

Menu path: /Model/Other linear models/ANOVA

append

Argument:	*filename*
Options:	`--time-series` (see below)
	`--fixed-sample` (see below)
	`--update-overlap` (see below)
	`--quiet` (print less confirmation details, see below)
	See below for additional specialized options

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.). Please note that the join command offers much more control over the matching of supplementary data to the current dataset. Also note that appending data to an existing panel dataset is potentially quite tricky; see the section headed "Panel data" below.

The appended data may take the form of either additional observations on series already present in the dataset, and/or new series. In the case of adding series, compatibility requires either (a) that the number of observations for the new data equals that for the current data, or (b) that the new data carries clear observation information so that gretl can work out how to place the values. Note that if there's a "perfect match" of observation information (that is, conditions (a) and (b) are both satisfied), it is assumed that series, rather than observations, are to be added. And if it happens that there are no series names in the file whose data are to be appended that are not already present in the current dataset then nothing is done, and a warning is shown.

One case that is not supported is where the new data start earlier and also end later than the original data. To add new series in such a case you can use the `--fixed-sample` option; this has the effect of suppressing the adding of observations, and so restricting the operation to the addition of new series.

When a data file is selected for appending, there may be an area of overlap with the existing dataset; that is, one or more series may have one or more observations in common across the two sources. If the option `--update-overlap` is given, the `append` operation will replace any overlapping observations with the values from the selected data file, otherwise the values currently in place will be unaffected.

The additional specialized options `--sheet`, `--coloffset`, `--rowoffset` and `--fixed-cols` work in the same way as with open; see that command for explanations.

By default some information about the appended dataset is printed. The `--quiet` option reduces that printout to a confirmatory message stating just the path to the file. If you want the operation to be completely silent, then issue the command `set verbose off` before appending the data, in combination with the `--quiet` option.

Panel data

When new data are appended to a panel dataset, the result will be correct only if both the "units" or "individuals" and the time-periods are properly matched.

Two relatively simple cases should be handled correctly by `append`. Let n denote the number of cross-sectional units and T denote the number of time periods in the current panel, and let m denote the number of observations for the new data. If $m = n$ the new data are taken to be time-invariant, and are copied into place for each time period. On the other hand, if $m = T$ the data are treated as invariant across the panel units, and are copied into place for each unit. If $T = n$ an ambiguity arises. In that case the new data are treated as time-invariant by default, but you can force gretl to treat them as time series (invariant across the units) via the `--time-series` option.

If both the current dataset and the incoming data are recognized as panel data two cases arise. (1) The time-series length, T, differs between the two. Then an error is flagged. (2) T matches. Then a very simple assumption is made, namely that the units match up, *starting with the first unit* in both

datasets. If that assumption is not correct you must use join instead of append.

Menu path: /File/Append data

ar

Arguments:	*lags* ; *depvar indepvars*
Options:	--vcv (print covariance matrix)
	--quiet (don't print parameter estimates)
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 (2002). 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/Univariate time series/AR Errors (GLS)

ar1

Arguments:	*depvar indepvars*
Options:	--hilu (use Hildreth–Lu procedure)
	--pwe (use Prais–Winsten estimator)
	--vcv (print covariance matrix)
	--no-corc (do not fine-tune results with Cochrane-Orcutt)
	--loose (use looser convergence criterion)
	--quiet (don't print anything)
Examples:	ar1 1 0 2 4 6 7
	ar1 y 0 xlist --pwe
	ar1 y 0 xlist --hilu --no-corc

Computes feasible GLS estimates for a model in which the error term is assumed to follow a first-order autoregressive process.

The default method is the Cochrane–Orcutt iterative procedure; see for example section 9.4 of Ramanathan (2002). The criterion for convergence is that successive estimates of the autocorrelation coefficient do not differ by more than 1e-6, or if the --loose option is given, by more than 0.001. If this is not achieved within 100 iterations an error is flagged.

If the --pwe option is given, the Prais–Winsten estimator is used. This involves an iteration similar to Cochrane–Orcutt; the difference is that while Cochrane–Orcutt discards the first observation, Prais–Winsten makes use of it. See, for example, Chapter 13 of Greene (2000) for details.

If the --hilu option is given, the Hildreth–Lu search procedure is used. The results are then fine-tuned using the Cochrane–Orcutt method, unless the --no-corc flag is specified. The --no-corc option is ignored for estimators other than Hildreth–Lu.

Menu path: /Model/Univariate time series/AR Errors (GLS)

arch

Arguments:	*order depvar indepvars*
Option:	--quiet (don't print anything)
Example:	arch 4 y 0 x1 x2 x3

This command is retained at present for backward compatibility, but you are better off using the maximum likelihood estimator offered by the garch command; for a plain ARCH model, set the first GARCH parameter to 0.

Estimates the given model specification allowing for ARCH (Autoregressive Conditional Heteroskedasticity). The model is first estimated via OLS, then an auxiliary regression is run, in which the squared residual from the first stage is regressed on its own lagged values. The final step is weighted least squares estimation, using as weights the reciprocals of the fitted error variances from the auxiliary regression. (If the predicted variance of any observation in the auxiliary regression is not positive, then the corresponding squared residual is used instead).

The `alpha` values displayed below the coefficients are the estimated parameters of the ARCH process from the auxiliary regression.

See also garch and modtest (the `--arch` option).

arima

Arguments:	*p d q* `[ ;` *P D Q* `] ;` *depvar* `[` *indepvars* `]`
Options:	`--verbose` (print details of iterations)
	`--quiet` (don't print out results)
	`--vcv` (print covariance matrix)
	`--hessian` (see below)
	`--opg` (see below)
	`--nc` (do not include a constant)
	`--conditional` (use conditional maximum likelihood)
	`--x-12-arima` (use X-12-ARIMA, or X13, for estimation)
	`--lbfgs` (use L-BFGS-B maximizer)
	`--y-diff-only` (ARIMAX special, see below)
	`--lagselect` (see below)
Examples:	`arima 1 0 2 ; y`
	`arima 2 0 2 ; y 0 x1 x2 --verbose`
	`arima 0 1 1 ; 0 1 1 ; y --nc`
	See also `armaloop.inp`, `auto_arima.inp`, `bjg.inp`

Note: `arma` is an acceptable alias for this command.

If no *indepvars* list is given, estimates a univariate ARIMA (Autoregressive, Integrated, Moving Average) model. The values *p*, *d* and *q* represent the autoregressive (AR) order, the differencing order, and the moving average (MA) order respectively. These values may be given in numerical form, or as the names of pre-existing scalar variables. A *d* value of 1, for instance, means that the first difference of the dependent variable should be taken before estimating the ARMA parameters.

If you wish to include only specific AR or MA lags in the model (as opposed to all lags up to a given order) you can substitute for *p* and/or *q* either (a) the name of a pre-defined matrix containing a set of integer values or (b) an expression such as `{1,4}`; that is, a set of lags separated by commas and enclosed in braces.

The optional integer values *P*, *D* and *Q* represent the seasonal AR order, the order for seasonal differencing, and the seasonal MA order, respectively. These are applicable only if the data have a frequency greater than 1 (for example, quarterly or monthly data). These orders may be given in numerical form or as scalar variables.

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).

An alternative form of syntax is available for this command: if you do not want to apply differencing (either seasonal or non-seasonal), you may omit the *d* and *D* fields altogether, rather than explicitly entering 0. In addition, `arma` is a synonym or alias for `arima`. Thus for example the following command is a valid way to specify an ARMA(2, 1) model:

```
arma 2 1 ; y
```

The default is to use the "native" gretl ARMA functionality, with estimation by exact ML; estimation via conditional ML is available as an option. (If X-12-ARIMA is installed you have the option of using it instead of native code. Note that the newer X13 works as a drop-in replacement in exactly the same way.) For details regarding these options, please see chapter 31 of the *Gretl User's Guide*.

When native exact ML code is used, estimated standard errors are by default based on a numerical approximation to the (negative inverse of) the Hessian, with a fallback to the outer product of the gradient (OPG) if calculation of the numerical Hessian should fail. Two (mutually exclusive) option flags can be used to force the issue: the `--opg` option forces use of the OPG method, with no attempt to compute the Hessian, while the `--hessian` flag disables the fallback to OPG. Note that failure of the numerical Hessian computation is generally an indicator of a misspecified model.

The option `--lbfgs` is specific to estimation using native ARMA code and exact ML: it calls for use of the "limited memory" L-BFGS-B algorithm in place of the regular BFGS maximizer. This may help in some instances where convergence is difficult to achieve.

The option `--y-diff-only` is specific to estimation of ARIMAX models (models with a non-zero order of integration and including exogenous regressors), and applies only when gretl's native exact ML is used. For such models the default behavior is to difference both the dependent variable and the regressors, but when this option is specified only the dependent variable is differenced, the regressors remaining in level form.

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

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

where ℓ is the log-likelihood and k is the total number of parameters estimated. Note that X-12-ARIMA does not produce information criteria such as AIC when estimation is by conditional ML.

The AR and MA roots shown in connection with ARMA estimation are based on the following representation of an ARMA(p, q) process:

$$(1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p) Y = c + (1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q)\varepsilon_t$$

The AR roots are therefore the solutions to

$$1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p L^p = 0$$

and stability requires that these roots lie outside the unit circle.

The "frequency" figure printed in connection with the 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.

Lag selection

When the `--lagselect` option is given, this command does not give specific estimates, but instead produces a table showing information criteria and log-likelihood for a number of ARMA or ARIMA specifications. The lag orders p and q are taken as maxima; and if a seasonal specification is provided P and Q are also taken as maxima. In each case the minimum order is taken to be 0, and results are shown for all specifications from mimima to maxima. The degrees of differencing in the command, d and/or D, are respected but not treated as subject to search. A matrix holding the results is available via the $test accessor.

Menu path: /Model/Univariate time series/ARIMA

arma

See arima; `arma` is an alias.

bds

Arguments:	*order x*
Options:	`--corr1`=*rho* (see below)
	`--sdcrit`=*multiple* (see below)
	`--boot`=*N* (see below)
	`--matrix`=*m* (use matrix input)
	`--quiet` (suppress printing of results)
Examples:	`bds 5 x`
	`bds 3 --matrix=m`
	`bds 4 --sdcrit=2.0`

Performs the BDS (Brock *et al.* (1996)) test for nonlinearity of the series *x*. In an econometric context this is typically used to test a regression residual for violation of the IID condition. The test is based on a set of correlation integrals, designed to detect nonlinearity of progressively higher dimensionality, and the *order* argument sets the number of such integrals. This must be at least 2; the first integral establishes a baseline but does not support a test. The BDS test is of the portmanteau type: able to detect all manner of departures from linearity but not informative about how exactly the condition was violated.

Instead of giving *x* as a series, the `--matrix` option can be used to specify a matrix as input. The matrix must be a vector (column or row).

Criterion for closeness

The correlation integrals are based on a measure of "closeness" of data points, where two points are considered close if they lie within ϵ of each other. The test requires a specification of ϵ. By default gretl follows the recommendation of Kanzler (1999): ϵ is chosen such that the first-order correlation integral is around 0.7. A common alternative (requiring less computation) is to specify ϵ as a multiple of the standard deviation of the target series. The `--sdcrit` option supports the latter method; in the third example above ϵ is set to twice the standard deviation of *x*. The `--corr1` option implies use of Kanzler's method but allows for a target correlation other than 0.7. It should be clear that these two options are mutually exclusive.

Bootstrapping

BDS test statistics are asymptotically distributed as $N(0,1)$ but the test over-rejects quite markedly in small to moderate-sized samples. For that reason *P*-values are by default obtained via bootstrapping when *x* is of length less than 600 (but by reference to the normal distribution otherwise). If you want to use the bootstrap for larger samples you can force the issue by giving a non-zero value for the `--boot` option, Conversely, if you don't want bootstrapping for smaller samples, give a zero value for `--boot`.

P-values

When bootstrapping is performed the default number of iterations is 1999, but you can specify a different number by giving a value greater than 1 with `--boot`.

Accessor matrix

On successful completion of this command, $result retrieves the test results in the form of a matrix with two rows and *order* − 1 columns. The first row contains test statistics and the second *P*-values

for each of the per-dimension tests under the null that x is linear/IID.

biprobit

Arguments:	*depvar1 depvar2 indepvars1* `[ ; ` *indepvars2* `]`
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)
	`--cluster=`*clustvar* (see logit for explanation)
	`--opg` (see below)
	`--save-xbeta` (see below)
	`--verbose` (print extra information)
Examples:	`biprobit y1 y2 0 x1 x2`
	`biprobit y1 y2 0 x11 x12 ; 0 x21 x22`
	See also `biprobit.inp`

Estimates a bivariate probit model, using the Newton–Raphson method to maximize the likelihood.

The argument list starts with the two (binary) dependent variables, followed by a list of regressors. If a second list is given, separated by a semicolon, this is interpreted as a set of regressors specific to the second equation, with *indepvars1* being specific to the first equation; otherwise *indepvars1* is taken to represent a common set of regressors.

By default, standard errors are computed using the analytical Hessian at convergence. But if the `--opg` option is given the covariance matrix is based on the Outer Product of the Gradient (OPG), or if the `--robust` option is given QML standard errors are calculated, using a "sandwich" of the inverse of the Hessian and the OPG.

Note that the estimate of rho, the correlation of the error terms across the two equations, is included in the coefficient vector; it's the last element in the accessors `coeff`, `stderr` and `vcv`.

After successful estimation, the accessor `$uhat` retrieves a matrix with two columns holding the generalized residuals for the two equations; that is, the expected values of the disturbances conditional on the observed outcomes and covariates. By default `$yhat` retrieves a matrix with four columns, holding the estimated probabilities of the four possible joint outcomes for (y_1, y_2), in the order (1,1), (1,0), (0,1), (0,0). Alternatively, if the option `--save-xbeta` is given, `$yhat` has two columns and holds the values of the index functions for the respective equations.

The output includes a test of the null hypothesis that the disturbances in the two equations are uncorrelated. This is a likelihood ratio test unless the QML variance estimator is requested, in which case it's a Wald test.

bkw

Option:	`--quiet` (don't print anything)
Example:	`longley.inp`

Must follow the estimation of a model which includes at least two independent variables. Calculates and displays diagnostic information pertaining to collinearity, namely the BKW Table, based on the work of Belsley *et al.* (1980). This table presents a sophisticated analysis of the degree and sources of collinearity, via eigenanalysis of the inverse correlation matrix. For a thorough account of the BKW approach with reference to gretl, and with several examples, see Adkins *et al.* (2015).

Following this command the $result accessor may be used to retrieve the BKW table as a matrix. See also the vif command for a simpler approach to diagnosing collinearity.

There is also a function named bkw which offers greater flexibility.

Menu path: Model window, /Analysis/Collinearity

boxplot

Argument: *varlist*
Options: `--notches` (show 90 percent interval for median)
`--factorized` (see below)
`--panel` (see below)
`--matrix=`*name* (plot columns of named matrix)
`--output=`*filename* (send output to specified file)

These plots 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 from each end of the box for a range equal to 1.5 times the interquartile range. Observations outside that range are considered outliers and represented via dots. A line is drawn across the box at the median. A "+" sign is used to indicate the mean. If the option of showing a confidence interval for the median is selected, this is computed via the bootstrap method and shown in the form of dashed horizontal lines above and/or below the median.

The `--factorized` option allows you to examine the distribution of a chosen variable conditional on the value of some discrete factor. For example, if a data set contains wages and a gender dummy variable you can select the wage variable as the target and gender as the factor, to see side-by-side boxplots of male and female wages, as in

```
boxplot wage gender --factorized
```

Note that in this case you must specify exactly two variables, with the factor given second.

If the current data set is a panel, and just one variable is specified, the `--panel` option produces a series of side-by-side boxplots, one for each panel "unit" or group.

Generally, the argument *varlist* is required, and refers to one or more series in the current dataset (given either by name or ID number). But if a named matrix is supplied via the `--matrix` option this argument becomes optional: by default a plot is drawn for each column of the specified matrix.

Gretl's boxplots are generated using gnuplot, and it is possible to specify the plot more fully by appending additional gnuplot commands, enclosed in braces. For details, please see the help for the gnuplot command.

In interactive mode the result is displayed immediately. In batch mode the default behavior is that a gnuplot command file is written in the user's working directory, with a name on the pattern `gpttmpN.plt`, starting with N = `01`. The actual plots may be generated later using gnuplot (under MS Windows, wgnuplot). This behavior can be modified by use of the `--output=`*filename* option. For details, please see the gnuplot command.

Menu path: /View/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, continue.

catch

Syntax: `catch` *command*

This is not a command in its own right but can be used as a prefix to most regular commands: the effect is to prevent termination of a script if an error occurs in executing the command. If an error does occur, this is registered in an internal error code which can be accessed as $error (a zero value indicates success). The value of `$error` should always be checked immediately after using `catch`, and appropriate action taken if the command failed.

The `catch` keyword cannot be used before `if`, `elif` or `endif`. In addition it should not be used on calls to user-defined functions; it is intended for use only with gretl commands and calls to "built-in" functions or operators. Furthermore, `catch` cannot be used in conjunction with "back-arrow" assignment of models or plots to session icons (see chapter 3 of the *Gretl User's Guide*).

chow

Variants: `chow` *obs*
`chow` *dummyvar* `--dummy`
Options: `--dummy` (use a pre-existing dummy variable)
`--quiet` (don't print estimates for augmented model)
`--limit-to=`*list* (limit test to subset of regressors)
Examples:
```
chow 25
chow 1988:1
chow female --dummy
```

Must follow an OLS regression. If an observation number or date is given, provides a test for the null hypothesis of no structural break at the given split point. The procedure is to create a dummy variable which equals 1 from the split point specified by *obs* to the end of the sample, 0 otherwise, and also interaction terms between this dummy and the original regressors. If a dummy variable is given, tests the null hypothesis of structural homogeneity with respect to that dummy. Again, interaction terms are added. In either case an augmented regression is run including the additional terms.

By default an F statistic is calculated, taking the augmented regression as the unrestricted model and the original as the restricted. But if the original model used a robust estimator for the covariance matrix, the test statistic is a Wald chi-square value based on a robust estimator of the covariance matrix for the augmented regression.

The `--limit-to` option can be used to limit the set of interactions with the split dummy variable to a subset of the original regressors. The parameter for this option must be a named list, all of whose members are among the original regressors. The list should not include the constant.

Menu path: Model window, /Tests/Chow test

clear

Options: `--dataset` (clear dataset only)
`--functions` (clear functions (only))
`--all` (clear everything)

By default this command clears the current dataset (if any) plus all saved variables (scalars, matrices, etc.) out of memory. Note that opening a new dataset, or using the `nulldata` command to create an empty dataset, also has this effect, so explicit use of `clear` is not usually necessary.

If the `--dataset` option is given, then only the dataset is cleared (plus any named lists of series); other saved objects such as matrices, scalars and bundles are preserved.

If the `--functions` option is given, then any user-defined functions, and any functions defined by packages that have been loaded, are cleared out of memory. The dataset and other variables are not affected.

If the `--all` option is given, clearing is comprehensive: the dataset, saved variables of all kinds, plus user-defined and packaged functions.

coeffsum

Argument:	*varlist*
Option:	`--quiet` (don't print anything)
Example:	`coeffsum xt xt_1 xr_2`
	`restrict.inp`

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.

The `--quiet` option may be useful if one just wants access to the $test and $pvalue values that are recorded on successful completion.

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

coint

Arguments:	*order depvar indepvars*
Options:	`--nc` (do not include a constant)
	`--ct` (include constant and trend)
	`--ctt` (include constant and quadratic trend)
	`--seasonals` (include seasonal dummy variables)
	`--skip-df` (no DF tests on individual variables)
	`--test-down[=`*criterion*`]` (automatic lag order)
	`--verbose` (print extra details of regressions)
	`--silent` (don't print anything)
Examples:	`coint 4 y x1 x2`
	`coint 0 y x1 x2 --ct --skip-df`

The Engle and Granger (1987) cointegration test. The default procedure is: (1) carry out Dickey–Fuller tests on the null hypothesis that each of the variables listed has a unit root; (2) estimate the cointegrating regression; and (3) run a DF test on the residuals from the cointegrating regression. If the `--skip-df` flag is given, step (1) is omitted.

If the specified lag order is positive all the Dickey–Fuller tests use that order, with this qualification: if the `--test-down` option is given, the given value is taken as the maximum and the actual lag order used in each case is obtained by testing down. See the adf command for details of this procedure.

By default, the cointegrating regression contains a constant. If you wish to suppress the constant, add the `--nc` flag. If you wish to augment the list of deterministic terms in the cointegrating regression with a linear or quadratic trend, add the `--ct` or `--ctt` flag. These option flags are mutually exclusive. You also have the option of adding seasonal dummy variables (in the case of quarterly or monthly data).

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

For the cointegration tests due to Søren Johansen, see johansen.

Menu path: /Model/Multivariate time series

continue

This command can be used only within a loop; it has the effect of skipping the subsequent statements within the current iteration of the current (innermost) loop. See also loop, break

corr

Variants: `corr [` *varlist* `]`
`corr --matrix=`*matname*
Options: `--uniform` (ensure uniform sample)
`--spearman` (Spearman's rho)
`--kendall` (Kendall's tau)
`--verbose` (print rankings)
`--plot=`*mode-or-filename* (see below)
`--triangle` (only plot lower half, see below)
`--quiet` (do not print anything)
Examples:
```
corr y x1 x2 x3
corr ylist --uniform
corr x y --spearman
corr --matrix=X --plot=display
```

By default, prints the pairwise correlation coefficients (Pearson's product-moment correlation) for the variables in *varlist*, or for all variables in the data set if *varlist* is not given. The standard behavior is to use all available observations for computing each pairwise coefficient, but if the `--uniform` option is given the sample is limited (if necessary) so that the same set of observations is used for all the coefficients. This option has an effect only if there are differing numbers of missing values for the variables used.

The (mutually exclusive) options `--spearman` and `--kendall` produce, respectively, Spearman's rank correlation rho and Kendall's rank correlation tau in place of the default Pearson coefficient. When either of these options is given, *varlist* should contain just two variables.

When a rank correlation is computed, the `--verbose` option can be used to print the original and ranked data (otherwise this option is ignored).

If *varlist* contains more than two series and gretl is not in batch mode, a "heatmap" plot of the correlation matrix is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command. When plotting is active the option `--triangle` can be used to show only the lower triangle of the matrix plot.

If the alternative form is given, using a named matrix rather than a list of series, the `--spearman` and `--kendall` options are not available—but see the npcorr function.

The $result accessor can be used to obtain the correlations as a matrix. Note that if it's this *matrix* that's of interest, not just the pairwise coefficients, then in the presence of missing values it's advisable to use the `--uniform` option. Unless a single, common sample is used it is not guaranteed that the correlation matrix will be positive semidefinite, as it ought to be by construction.

Menu path: /View/Correlation matrix

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

corrgm

Arguments:	*y* [*order*]
Options:	`--bartlett` (use Bartlett standard errors)
	`--plot=`*mode-or-filename* (see below)
	`--silent` (don't print anything)
	`--acf-only` (omit partial autocorrelations)
Examples:	`corrgm x 12`
	`corrgm GDP 12 --acf-only`

Prints and/or graphs the values of the autocorrelation function (ACF) for the series *y*, 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 t^{th} observation of the variable u and s denotes the number of lags.

Unless the `--acf-only` option is given, partial autocorrelations (PACF, calculated using the Durbin–Levinson algorithm) are also shown: these are net of the effects of intervening lags.

Asterisks are used to indicate statistical significance of the individual autocorrelations. By default this is assessed using a standard error of one over the square root of the sample size, but if the `--bartlett` option is given then Bartlett standard errors are used for the ACF. This option also governs the confidence band drawn in the ACF plot, if applicable. In addition the Ljung–Box Q statistic is shown; this tests the null that the series is "white noise" up to the given lag.

If an *order* 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.

Plotting

By default, if gretl is not in batch mode a plot of the correlogram is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to show a plot even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Accessors

Upon successful completion, the accessors $test and $pvalue can be used to retrieve the Q statistic and its P-value, evaluated at the maximum lag. Note that if you just want this test you can use use the ljungbox function instead.

Menu path: /Variable/Correlogram

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

cusum

Options:	`--squares` (perform the CUSUMSQ test)
	`--quiet` (just print the Harvey–Collier test)
	`--plot=`*mode-or-filename* (see below)

Must follow the estimation of a model via OLS. Performs the CUSUM test—or if the `--squares` option is given, the CUSUMSQ test—for parameter stability. A series of 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, or the squares of these errors, is printed. 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.

In the case of the CUSUM test, the Harvey–Collier t-statistic for testing the null hypothesis of parameter stability is also printed. See Greene's *Econometric Analysis* for details. For the CUSUMSQ test, the 95 percent confidence band is calculated using the algorithm given in Edgerton and Wells (1994).

By default, if gretl is not in batch mode a plot of the cumulated series and confidence band is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Menu path: Model window, /Tests/CUSUM(SQ)

data

Argument:	*varlist*
Options:	`--compact=`*method* (specify compaction method)
	`--quiet` (don't report results except on error)
	`--name=`*identifier* (rename imported series)
	`--odbc` (import from ODBC database)
	`--no-align` (ODBC-specific, see below)

Reads the variables in *varlist* from a database file (native gretl, RATS 4.0 or PcGive), which must have been opened previously using the open command. The `data` command can also be used to import series from DB.NOMICS or from an ODBC database; for details on those variants see gretl + DB.NOMICS or chapter 42 of the *Gretl User's Guide*, respectively.

The data frequency and sample range may be established via the setobs and smpl commands prior to using this command. Here's an example:

```
open fedstl.bin
setobs 12 2000:01
smpl ; 2019:12
data unrate cpiaucsl
```

The commands above open the database named `fedstl.bin` (which is supplied with gretl), establish a monthly dataset starting in January 2000 and ending in December of 2019, and then import the series named `unrate` (unemployment rate) and `cpiaucsl` (all-items CPI).

If `setobs` and `smpl` are not specified in this way, the data frequency and sample range are set using the first variable read from the database.

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

```
data LHUR PUNEW --compact=average
```

The five available compaction methods are "average" (takes the mean of the high frequency observations), "last" (uses the last observation), "first", "sum" and "spread". If no method is specified, the default is to use the average. The "spread" method is special: no information is lost, rather it is spread across multiple series, one per sub-period. So for example when adding a monthly series to a quarterly dataset three series are created, one for each month of the quarter; their names bear the suffixes `m01`, `m02` and `m03`.

If the series to be read are of *lower* frequency than the working dataset the values of the added data are simply repeated as required, but note that the tdisagg function can then be used to distribution or interpolation ("temporal disaggregation").

In the case of native gretl databases (only), the "glob" characters * and ? can be used in *varlist* to import series that match the given pattern. For example, the following will import all series in the database whose names begin with cpi:

```
data cpi*
```

The --name option can be used to set a name for the imported series other than the original name in the database. The parameter must be a valid gretl identifier. This option is restricted to the case where a single series is specified for importation.

The --no-align option applies only to importation of series via ODBC. By default we require that the ODBC query returns information telling gretl on which rows of the dataset to place the incoming data—or at least that the number of incoming values matches either the length of the dataset or the length of the current sample range. Setting the --no-align option relaxes this requirement: failing the conditions just mentioned, incoming values are simply placed consecutively starting at the first row of the dataset. If there are fewer such values than rows in the dataset the trailing rows are filled with NAs; if there are more such values than rows the extra values are discarded. For more on ODBC importation see chapter 42 of the *Gretl User's Guide*.

Menu path: /File/Databases

dataset

Arguments: *keyword parameters*
Option: --panel-time (see addobs below)
Examples:

```
dataset addobs 24
dataset addobs 2 --panel-time
dataset insobs 10
dataset compact 1
dataset compact 4 last
dataset expand
dataset transpose
dataset sortby x1
dataset resample 500
dataset renumber x 4
dataset pad-daily 7
dataset unpad-daily
dataset clear
```

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

addobs: Must be followed by a positive integer, call it n. Adds n 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. If the dataset takes the form of a panel, the default action is to add n cross-sectional units to the panel, but if the --panel-time flag is given the effect is to add n observations to the time series for each unit.

insobs: Must be followed by a positive integer no greater than the current number of observations. Inserts a single observation at the specified position. All subsequent data are shifted by one place and the dataset is extended by one observation. All variables apart from the constant are given missing values at the new observation. This action is not available for panel datasets.

`clear`: No parameter required. Clears out the current data, returning gretl to its initial "empty" state.

`compact`: This action is available for time series data only; it compacts all the series in the data set to a lower frequency. It requires one parameter, a positive integer representing the new frequency. In general this should be lower than the current frequency (for example, a value of 4 when the current frequency is 12 indicates compaction from monthly to quarterly). The one exception is a new frequency of 52 (weekly) when the current data are daily (frequency 5, 6 or 7). A second parameter may be given, namely one of `sum`, `first`, `last` or `spread`, to specify, respectively, compaction using the sum of the higher-frequency values, start-of-period values, end-of-period values, or spreading of the higher-frequency values across multiple series (one per sub-period). The default is to compact by averaging.

In the case of compaction from daily to weekly frequency (only), the two special options `--repday` and `--weekstart` are available. The first of these allows you to select a "representative day" of the week to serve as the weekly value. The parameter to this option must be an integer from 0 (Sunday) to 6 (Saturday). For example, giving `--repday=3` selects Wednesday's value as the weekly value. If the `--repday` option is not given, we need to know on which day the week is deemed to start in order to align the data correctly. For 5- or 6-day data this is always taken to be Monday, but with 7-day data you have a choice between `--weekstart=0` (Sunday) and `--weekstart=1` (Monday), with Monday being the default.

`expand`: This action is only available for annual or quarterly time series data: annual data can be expanded to quarterly or monthly, and quarterly data to monthly. All series in the data set are padded out to the new frequency by repeating the existing values. If the original dataset is annual the default expansion is to quarterly but `expand` can be followed by `12` to request monthly. See the tdisagg function for more sophisticated means of converting data to higher frequency.

`transpose`: No additional parameter required. 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 action may be useful if data have been read from some external source in which the rows of the data table represent variables.

`sortby`: The name of a single series or list is required. If one series is given, the observations on all variables in the dataset are re-ordered by increasing value of the specified series. If a list is given, the sort proceeds hierarchically: if the observations are tied in sort order with respect to the first key variable then the second key is used to break the tie, and so on until the tie is broken or the keys are exhausted. Note that this action is available only for undated data.

`dsortby`: Works as `sortby` except that the re-ordering is by decreasing value of the key series.

`resample`: Constructs a new dataset by random sampling, with replacement, of the rows of the current dataset. One argument is required, namely the number of rows to include. This may be less than, equal to, or greater than the number of observations in the original data. The original dataset can be retrieved via the command `smpl full`.

`renumber`: Requires the name of an existing series followed by an integer between 1 and the number of series in the dataset minus one. Moves the specified series to the specified position in the dataset, renumbering the other series accordingly. (Position 0 is occupied by the constant, which cannot be moved.)

`pad-daily`: Valid only if the current dataset contains dated daily data with an incomplete calendar. The effect is to pad the data out to a complete calendar by inserting blank rows (that is, rows containing nothing but `NA`s). This option requires an integer parameter, namely the number of days per week, which must be 5, 6 or 7, and must be greater than or equal to the current data frequency. On successful completion, the data calendar will be "complete" relative to this value. For example if days-per-week is 5 then all weekdays will be represented, whether or not any data are available for those days.

`unpad-daily`: Valid only if the current dataset contains dated daily data, in which case it performs the inverse operation of `pad-daily`. That is, any rows that contain nothing but `NA`s are removed,

while the time-series property of the dataset is preserved along with the dates of the individual observations.

Menu path: /Data

delete

Variants: `delete` *varlist*
`delete` *varname*
`delete --type=`*type-name*
`delete` *pkgname*
Options: `--db` (delete series from database)
`--force` (see below)

This command is an all-purpose destructor. It should be used with caution; no confirmation is asked.

In the first form above, *varlist* is a list of series, given by name or ID number. Note that when you delete series any series with higher ID numbers than those on the deletion list will be re-numbered. If the `--db` option is given, this command deletes the listed series not from the current dataset but from a gretl database, assuming that a database has been opened, and the user has write permission for file in question. See also the open command.

In the second form, the name of a scalar, matrix, string or bundle may be given for deletion. The `--db` option is not applicable in this case. Note that series and variables of other types should not be mixed in a given call to `delete`.

In the third form, the `--type` option must be accompanied by one of the following type-names: `matrix`, `bundle`, `string`, `list`, `scalar` or `array`. The effect is to delete all variables of the given type. In this case no argument other than the option should be given.

The fourth form can be used to unload a function package. In this case the `.gfn` suffix must be supplied, as in

```
delete somepkg.gfn
```

Note that this does not delete the package file, it just unloads the package from memory.

Deleting variables in a loop

In general it is not permitted to delete variables in the context of a loop, since this may threaten the integrity of the loop code. However, if you are confident that deleting a certain variable is safe you can override this prohibition by appending the `--force` flag to the `delete` command.

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

diff

Argument: *varlist*
Examples: `penngrow.inp`, `sw_ch12.inp`, `sw_ch14.inp`

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: /Add/First differences of selected variables

difftest

Arguments:	*series1 series2*
Options:	`--sign` (Sign test, the default)
	`--rank-sum` (Wilcoxon rank-sum test)
	`--signed-rank` (Wilcoxon signed-rank test)
	`--verbose` (print extra output)
	`--quiet` (suppress printed output)
Example:	`ooballot.inp`

Carries out a nonparametric test for a difference between two populations or groups, the specific test depending on the option selected.

With the `--sign` option, the Sign test is performed. This test is based on the fact that if two samples, x and y, are drawn randomly from the same distribution, the probability that $x_i > y_i$, for each observation i, should equal 0.5. The test statistic is w, the number of observations for which $x_i > y_i$. Under the null hypothesis this follows the Binomial distribution with parameters (n, 0.5), where n is the number of observations.

With the `--rank-sum` option, the Wilcoxon rank-sum test is performed. This test proceeds by ranking the observations from both samples jointly, from smallest to largest, then finding the sum of the ranks of the observations from one of the samples. The two samples do not have to be of the same size, and if they differ the smaller sample is used in calculating the rank-sum. Under the null hypothesis that the samples are drawn from populations with the same median, the probability distribution of the rank-sum can be computed for any given sample sizes; and for reasonably large samples a close Normal approximation exists.

With the `--signed-rank` option, the Wilcoxon signed-rank test is performed. This is designed for matched data pairs such as, for example, the values of a variable for a sample of individuals before and after some treatment. The test proceeds by finding the differences between the paired observations, $x_i - y_i$, ranking these differences by absolute value, then assigning to each pair a signed rank, the sign agreeing with the sign of the difference. One then calculates W_+, the sum of the positive signed ranks. As with the rank-sum test, this statistic has a well-defined distribution under the null that the median difference is zero, which converges to the Normal for samples of reasonable size.

For the Wilcoxon tests, if the `--verbose` option is given then the ranking is printed. (This option has no effect if the Sign test is selected.)

On successful completion the accessors $test and $pvalue are available. If one just wants to obtain these values the `--quiet` flag can be appended to the command.

discrete

Argument:	*varlist*
Option:	`--reverse` (mark variables as continuous)
Examples:	`ooballot.inp`, `oprobit.inp`

Marks each variable in *varlist* as being discrete. By default all variables are treated as continuous; marking a variable as discrete affects the way the variable is handled in frequency plots, and also allows you to select the variable for the command dummify.

If the `--reverse` flag is given, the operation is reversed; that is, the variables in *varlist* are marked as being continuous.

Menu path: /Variable/Edit attributes

dpanel

Argument:	*p* ; *depvar indepvars* [; *instruments*]
Options:	`--quiet` (don't show estimated model)
	`--vcv` (print covariance matrix)
	`--two-step` (perform 2-step GMM estimation)
	`--system` (add equations in levels)
	`--collapse` (see below)
	`--time-dummies` (add time dummy variables)
	`--dpdstyle` (emulate DPD package for Ox)
	`--asymptotic` (uncorrected asymptotic standard errors)
	`--keep-extra` (see below)
Examples:	`dpanel 2 ; y x1 x2`
	`dpanel 2 ; y x1 x2 --system`
	`dpanel {2 3} ; y x1 x2 ; x1`
	`dpanel 1 ; y x1 x2 ; x1 GMM(x2,2,3)`
	See also `bbond98.inp`

Carries out estimation of dynamic panel data models (that is, panel models including one or more lags of the dependent variable) using either the GMM-DIF or GMM-SYS method.

The parameter *p* represents the order of the autoregression for the dependent variable. In the simplest case this is a scalar value, but a pre-defined matrix may be given for this argument, to specify a set of (possibly non-contiguous) lags to be used.

The dependent variable and regressors should be given in levels form; they will be differenced automatically (since this estimator uses differencing to cancel out the individual effects).

The last (optional) field in the command is for specifying instruments. If no instruments are given, it is assumed that all the independent variables are strictly exogenous. If you specify any instruments, you should include in the list any strictly exogenous independent variables. For predetermined regressors, you can use the `GMM` function to include a specified range of lags in block-diagonal fashion. This is illustrated in the third example above. The first argument to `GMM` is the name of the variable in question, the second is the minimum lag to be used as an instrument, and the third is the maximum lag. The same syntax can be used with the `GMMlevel` function to specify GMM-type instruments for the equations in levels.

The `--collapse` option can be used to limit the proliferation of "GMM-style" instruments, which can be a problem with this estimator. Its effect is to reduce such instruments from one per lag per observation to one per lag.

By default the results of 1-step estimation are reported (with robust standard errors). You may select 2-step estimation as an option. In both cases tests for autocorrelation of orders 1 and 2 are provided, as well as Sargan and/or Hansen overidentification tests and a Wald test for the joint significance of the regressors. Note that in this differenced model first-order autocorrelation is not a threat to the validity of the model, but second-order autocorrelation violates the maintained statistical assumptions.

In the case of 2-step estimation, standard errors are by default computed using the finite-sample correction suggested by Windmeijer (2005). The standard asymptotic standard errors associated with the 2-step estimator are generally reckoned to be an unreliable guide to inference, but if for some reason you want to see them you can use the `--asymptotic` option to turn off the Windmeijer correction.

If the `--time-dummies` option is given, a set of time dummy variables is added to the specified regressors. The number of dummies is one less than the maximum number of periods used in estimation, to avoid perfect collinearity with the constant. The dummies are entered in differenced

form unless the --dpdstyle option is given, in which case they are entered in levels.

As with other estimation commands, a $model bundle is available after estimation. In the case of dpanel, the --keep-extra option can be used to save additional information in this bundle, namely the GMM weight and instrument matrices.

For further details and examples, please see chapter 24 of the *Gretl User's Guide*.

Menu path: /Model/Panel/Dynamic panel model

dummify

Argument:	*varlist*
Options:	--drop-first (omit lowest value from encoding)
	--drop-last (omit highest value from encoding)

For any suitable variables in *varlist*, creates a set of dummy variables coding for the distinct values of that variable. Suitable variables are those that have been explicitly marked as discrete, or those that take on a fairly small number of values all of which are "fairly round" (multiples of 0.25).

By default a dummy variable is added for each distinct value of the variable in question. For example if a discrete variable x has 5 distinct values, 5 dummy variables will be added to the data set, with names Dx_1, Dx_2 and so on. The first dummy variable will have value 1 for observations where x takes on its smallest value, 0 otherwise; the next dummy will have value 1 when x takes on its second-smallest value, and so on. If one of the option flags --drop-first or --drop-last is added, then either the lowest or the highest value of each variable is omitted from the encoding (which may be useful for avoiding the "dummy variable trap").

There is also a corresponding function for this command, see dummify. This makes it possible to embed the call directly in a regression specification. For example, the following line specifies a model where y is regressed on the set of dummy variables coding for x. (However, option flags are for the command variant only, not for the function variant of dummify.)

```
ols y dummify(x)
```

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

duration

Arguments:	*depvar indepvars* [; *censvar*]
Options:	--exponential (use exponential distribution)
	--loglogistic (use log-logistic distribution)
	--lognormal (use log-normal distribution)
	--medians (fitted values are medians)
	--robust (robust (QML) standard errors)
	--cluster=*clustvar* (see logit for explanation)
	--vcv (print covariance matrix)
	--verbose (print details of iterations)
	--quiet (don't print anything)
Examples:	duration y 0 x1 x2
	duration y 0 x1 x2 ; cens
	See also weibull.inp

Estimates a duration model: the dependent variable (which must be positive) represents the duration of some state of affairs, for example the length of spells of unemployment for a cross-section

of respondents. By default the Weibull distribution is used but the exponential, log-logistic and log-normal distributions are also available.

If some of the duration measurements are right-censored (e.g. an individual's spell of unemployment has not come to an end within the period of observation) then you should supply the trailing argument *censvar*, a series in which non-zero values indicate right-censored cases.

By default the fitted values obtained via the accessor `$yhat` are the conditional means of the durations, but if the `--medians` option is given then `$yhat` provides the conditional medians instead.

Please see chapter 38 of the *Gretl User's Guide* for details.

Menu path: /Model/Limited dependent variable/Duration data

elif

See if.

else

See if. Note that `else` requires a line to itself, before the following conditional command. You can append a comment, as in

```
else # OK, do something different
```

But you cannot append a command, as in

```
else x = 5 # wrong!
```

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

Options: `--complete` (Create a complete document)
`--output=`*filename* (send output to specified file)

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 `--output` option 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.

The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification.

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)
`--quiet` (don't print results)
`--verbose` (print details of iterations)
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=`. These arguments are optional if the system in question has already been estimated and occupies the place of the "last model"; in that case the estimator defaults to the previously used value.

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.

If the `--verbose` option is given and an iterative method is specified, details of the iterations are printed.

eval

Argument: *expression*
Examples:
```
eval x
eval inv(X'X)
eval sqrt($pi)
```

This command makes gretl act like a glorified calculator. The program evaluates *expression* and prints its value. The argument may be the name of a variable, or something more complicated. In any case, it should be an expression which could stand as the right-hand side of an assignment statement.

In interactive use (for instance in the gretl console) an equals sign works as shorthand for `eval`, as in

```
=sqrt(x)
```

(with or without a space following "="). But this variant is not accepted in scripting mode since it could easily mask coding errors.

In most contexts print can be used in place of `eval` to much the same effect. See also printf for the case where you wish to combine textual and numerical output.

fcast

Variants: `fcast [`*startobs endobs*`] [`*vname*`]`
`fcast [`*startobs endobs*`]` *steps-ahead* `[`*vname*`] --recursive`
Options:
`--dynamic` (create dynamic forecast)
`--static` (create static forecast)
`--out-of-sample` (generate post-sample forecast)
`--no-stats` (don't print forecast statistics)
`--stats-only` (only print forecast statistics)
`--quiet` (don't print anything)
`--recursive` (see below)
`--all-probs` (see below)
`--plot=`*filename* (see below)
Examples:
```
fcast 1997:1 2001:4 f1
fcast fit2
fcast 2004:1 2008:3 4 rfcast --recursive
```
See also `gdp_midas.inp`

Must follow an estimation command. Forecasts are generated for a certain range of observations: if *startobs* and *endobs* are given, for that range (if possible); otherwise if the `--out-of-sample` option is given, for observations following the range over which the model was estimated; otherwise over the currently defined sample range. If an out-of-sample forecast is requested but no relevant observations are available, an error is flagged. Depending on the nature of the model, standard errors may also be generated; see below. Also see below for the special effect of the `--recursive` option.

If the last model estimated is a single equation, then the optional *vname* argument has the following effect: the forecast values are not printed, but are saved to the dataset under the given name. If the last model is a system of equations, *vname* has a different effect, namely selecting a particular endogenous variable for forecasting (the default being to produce forecasts for all the endogenous variables). In the system case, or if *vname* is not given, the forecast values can be retrieved using the accessor $fcast, and the standard errors, if available, via $fcse.

Static and dynamic forecasts

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 within 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.

Recursive forecasts

The `--recursive` option is presently available only for single-equation models estimated via OLS. When this option is given the forecasts are recursive. That is, each forecast is generated from an estimate of the given model using data from a fixed starting point (namely, the start of the sample range for the original estimation) up to the forecast date minus k, where k is the number of steps ahead, which must be given in the *steps-ahead* argument. The forecasts are always dynamic if this is applicable. Note that the *steps-ahead* argument should be given only in conjunction with the `--recursive` option.

Ordered and multinomial models

When estimation is via ordered logit or probit, or multinomial logit, one may be interested in the estimated probabilities of each of the discrete outcomes rather than just a "most likely" outcome for each observation. This is supported by the `--all-probs` option: the output of `fcast` is then a matrix with one column per possible outcome. The *vname* argument can be used to name this matrix, in which case nothing is printed. If *vname* is not given the matrix can be retrieved via $fcast. The `--plot` option is incompatible with `--all-probs`.

Forecast plots

The `--plot` option calls for a plot file to be produced, containing a graphical representation of the forecast. In the system case this option is available only when the *vname* argument is used to select a single variable for forecasting. The suffix of the *filename* argument to this option controls the format of the plot: `.eps` for EPS, `.pdf` for PDF, `.png` for PNG, `.plt` for a gnuplot command file. The dummy filename `display` can be used to force display of the plot in a window. For example,

```
fcast --plot=fc.pdf
```

will generate a graphic in PDF format. Absolute pathnames are respected, otherwise files are written to the gretl working directory.

Standard errors

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, /Analysis/Forecasts

flush

This simple command (no arguments, no options) is intended for use in time-consuming scripts that may be executed via the gretl GUI (it is ignored by the command-line program), to give the user a visual indication that things are moving along and gretl is not "frozen".

Ordinarily if you launch a script in the GUI no output is shown until its execution is completed, but the effect of invoking `flush` is as follows:

- On the first invocation, gretl opens a window, displays the output so far, and appends the message "Processing...".
- On subsequent invocations the text shown in the output window is updated, and a new "processing" message is appended.

When execution of the script is completed any remaining output is automatically flushed to the text window.

Please note, there is no point in using `flush` in scripts that take less than (say) 5 seconds to execute. Also note that this command should not be used at a point in the script where there is no further output to be printed, as the "processing" message will then be misleading to the user.

The following illustrates the intended use of `flush`:

```
set echo off
scalar n = 10
loop i=1..n
    # do some time-consuming operation
    loop 100 --quiet
        a = mnormal(200,200)
        b = inv(a)
    endloop
    # print some results
    printf "Iteration %2d done\n", i
    if i < n
        flush
    endif
endloop
```

foreign

Syntax: `foreign language=`*lang*

Options: `--send-data[=`*list*`]` (pre-load data; see below)
`--quiet` (suppress output from foreign program)

This command opens a special mode in which commands to be executed by another program are accepted. You exit this mode with `end foreign`; at this point the stacked commands are executed.

At present the "foreign" programs supported in this way are GNU R (`language=R`), Python, Julia, GNU Octave (`language=Octave`), Jurgen Doornik's Ox and Stata. Language names are recognized on a case-insensitive basis.

In connection with R, Octave and Stata the `--send-data` option has the effect of making data from gretl's workspace available within the target program. By default the entire dataset is sent, but you can limit the data to be sent by giving the name of a predefined list of series. For example:

```
list Rlist = x1 x2 x3
foreign language=R --send-data=Rlist
```

See chapter 44 of the *Gretl User's Guide* for details and examples.

fractint

Arguments:	*series* [*order*]
Options:	`--gph` (do Geweke and Porter-Hudak test)
	`--all` (do both tests)
	`--quiet` (don't print results)

Tests the specified series for fractional integration ("long memory"). The null hypothesis is that the integration order of the series is zero. By default the local Whittle estimator (Robinson, 1995) is used but if the `--gph` option is given the GPH test (Geweke and Porter-Hudak, 1983) is performed instead. If the `--all` flag is given then the results of both tests are printed.

For details on this sort of test, see Phillips and Shimotsu (2004).

If the optional *order* argument is not given the order for the test(s) is set automatically as the lesser of $T/2$ and $T^{0.6}$.

The estimated fractional integration orders and their standard errors are available via the $result accessor. With the `--all` option, the Local Whittle estimate will be in the first row and the GPH estimate in the second one.

The results of the test can be retrieved using the accessors $test and $pvalue. These values are based on the Local Whittle Estimator unless the `--gph` option is given.

Menu path: /Variable/Unit root tests/Fractional integration

freq

Argument:	*var*
Options:	`--nbins=`*n* (specify number of bins)
	`--min=`*minval* (specify minimum, see below)
	`--binwidth=`*width* (specify bin width, see below)
	`--normal` (test for the normal distribution)
	`--gamma` (test for gamma distribution)
	`--silent` (don't print anything)
	`--matrix=`*name* (use column of named matrix)
	`--plot=`*mode-or-filename* (see below)
Examples:	`freq x`
	`freq x --normal`
	`freq x --nbins=5`
	`freq x --min=0 --binwidth=0.10`

With no options given, displays the frequency distribution for the series *var* (given by name or number) in tabular form, with the number of bins and their size chosen automatically, with or without an accompanying plot as explained below. Upon successful completion of the command, the frequency table can be retrieved as a matrix using the $result accessor.

If the `--matrix` option is given, *var* (which must be an integer) is instead interpreted as a 1-based index that selects a column from the named matrix. If the matrix in question is in fact a column vector, the *var* argument may be omitted.

By default the frequency distribution employs an automatically calculated number of bins if the data are continuous, or no binning if the data are discrete. To control this point you can (a) use the discrete command to set the status of *var* or (b), if the data are continuous, specify *either* the number of bins or the minimum value and the width of the bins, as shown in the last two examples above. The `--min` option sets the lower limit of the left-most bin.

If the `--normal` option is given, the Doornik–Hansen chi-square test for normality is computed. 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). Note that the parameterization of the gamma distribution used in gretl is (shape, scale).

By default, if gretl is not in batch mode a plot of the distribution is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

The `--silent` flag suppresses the usual text output. This might be used in conjunction with one or other of the distribution test options: the test statistic and its p-value are recorded, and can be retrieved using the accessors $test and $pvalue. It might also be used along with the `--plot` option if you just want a histogram and don't care to see the accompanying text.

Note that gretl does not have a function that matches this command, but it is possible to use the aggregate function to achieve the same purpose. In addition, the frequency distribution constructed by `freq` can be obtained in matrix form via the $result accessor.

Menu path: /Variable/Frequency distribution

funcerr

Argument: [*message*]

Applicable only in the context of a user-defined function (see function). Causes execution of the current function to terminate with an error condition flagged.

The optional *message* argument can take the form of a string literal or the name of a string variable; if present it is printed as part of the error message shown to the caller of the function.

See also the closely related function, errorif.

function

Argument: *fnname*

Opens a block of statements in which a function is defined. This block must be closed with `end function`. (An exception is the case when a user-defined function shall be deleted, which is achieved by the single command line `function foo delete` for a function named "foo".) See chapter 14 of the *Gretl User's Guide* for details.

garch

Arguments: *p q* ; *depvar* [*indepvars*]
Options: `--robust` (robust standard errors)
`--verbose` (print details of iterations)
`--quiet` (don't print anything)
`--vcv` (print covariance matrix)
`--nc` (do not include a constant)
`--stdresid` (standardize the residuals)
`--fcp` (use Fiorentini, Calzolari, Panattoni algorithm)
`--arma-init` (initial variance parameters from ARMA)
Examples:
```
garch 1 1 ; y
garch 1 1 ; y 0 x1 x2 --robust
```
See also `garch.inp`, `sw_ch14.inp`

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 (which may be given in numerical form or as the names of pre-existing scalar variables) represent the lag orders in the conditional variance equation:

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

The parameter p therefore represents the Generalized (or "AR") order, while q represents the regular ARCH (or "MA") order. If p is non-zero, q must also be non-zero otherwise the model is unidentified. However, you can estimate a regular ARCH model by setting q to a positive value and p to zero. The sum of p and q must be no greater than 5. Note that a constant is automatically included in the mean equation unless the `--nc` option is given.

By default native gretl code is used in estimation of GARCH models, but you also have the option of using the algorithm of Fiorentini *et al.* (1996). The former uses the BFGS maximizer while the latter uses the information matrix to maximize the likelihood, with fine-tuning via the Hessian.

Several variant estimators of the 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 via the `garch_vcv` keyword under 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.

The GARCH residuals and estimated conditional variance can be retrieved as `$uhat` and `$h` respectively. For example, to get the conditional variance:

```
series ht = $h
```

If the `--stdresid` option is given, the `$uhat` values are divided by the square root of h_t.

Menu path: /Model/Univariate time series/GARCH

genr

Arguments: *newvar = formula*

NOTE: this command has undergone numerous changes and enhancements since the following help text was written, so for comprehensive and updated info on this command you'll want to refer to chapter 10 of the *Gretl User's Guide*. On the other hand, this help does not contain anything actually erroneous, so take the following as "you have this, plus more".

In the appropriate context, `series`, `scalar`, `matrix`, `string`, `bundle` and `array` are synonyms for this command.

Creates new variables, often via transformations of existing variables. See also diff, logs, lags, ldiff, 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 chapter 10 of the *Gretl User's Guide*.

```
series c = 10
```

A genr 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 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), >, <, == (is equal to), >= (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.

The genr command supports a wide range of mathematical and statistical functions, including all the common ones plus several that are special to econometrics. In addition it offers access to numerous internal variables that are defined in the course of running regressions, doing hypothesis tests, and so on.

For a listing of functions and accessors, see Chapter 2.

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. In the case of quarterly data (periodicity 4), the program creates dq1 = 1 for first quarter and 0 in other quarters, dq2 = 1 for the second quarter and 0 in other quarters, and so on. With monthly data the dummies are named dm1, dm2, and so on; with daily data they are named dd1, dd2, and so on; and with other frequencies the names are dummy_1, dummy_2, etc.
- genr unitdum and genr timedum create sets of special dummy variables for use with panel data. The first codes for the cross-sectional units and the second for the time period of the observations.

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 Add 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 "Save" menu in the model's window.

The special variable obs serves as an index of the observations. For instance series dum = (obs==15) will generate a dummy variable that has value 1 for observation 15, 0 otherwise. You can also use this variable to pick out particular observations by date or name. For example, series d = (obs>1986:4), series d = (obs>"2008-04-01"), or series d = (obs=="CA"). If daily dates or observation labels are used in this context, they should be enclosed in double quotes. Quarterly and monthly dates (with a colon) may be used unquoted. Note that in the case of annual time series data, the year is not distinguishable syntactically from a plain integer; therefore if you

wish to compare observations against `obs` by year you must use the function `obsnum` to convert the year to a 1-based index value, as in `series d = (obs>obsnum(1986))`.

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`.

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 = dsort(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 `NA` 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.

Menu path: /Add/Define new variable

Other access: Main window pop-up menu

gmm

Options: `--two-step` (two step estimation)
`--iterate` (iterated GMM)
`--vcv` (print covariance matrix)
`--verbose` (print details of iterations)
`--quiet` (don't print anything)
`--lbfgs` (use L-BFGS-B instead of regular BFGS)

Example: `hall_cbapm.inp`

Performs Generalized Method of Moments (GMM) estimation using the BFGS (Broyden, Fletcher,

Goldfarb, Shanno) algorithm. You must specify one or more commands for updating the relevant quantities (typically GMM residuals), one or more sets of orthogonality conditions, an initial matrix of weights, and a listing of the parameters to be estimated, all enclosed between the tags `gmm` and `end gmm`. Any options should be appended to the `end gmm` line.

Please see chapter 27 of the *Gretl User's Guide* for details on this command. Here we just illustrate with a simple example.

```
gmm e = y - X*b
  orthog e ; W
  weights V
  params b
end gmm
```

In the example above we assume that `y` and `X` are data matrices, `b` is an appropriately sized vector of parameter values, `W` is a matrix of instruments, and `V` is a suitable matrix of weights. The statement

```
orthog e ; W
```

indicates that the residual vector `e` is in principle orthogonal to each of the instruments composing the columns of `W`.

Parameter names

In estimating a nonlinear model it is often convenient to name the parameters tersely. In printing the results, however, it may be desirable to use more informative labels. This can be achieved via the additional keyword `param_names` within the command block. For a model with k parameters the argument following this keyword should be a double-quoted string literal holding k space-separated names, the name of a string variable that holds k such names, or the name of an array of k strings.

Menu path: /Model/Instrumental variables/GMM

gnuplot

Arguments:	*yvars xvar* [*dumvar*]
Options:	`--with-lines`[=*varspec*] (use lines, not points)
	`--with-lp`[=*varspec*] (use lines and points)
	`--with-impulses`[=*varspec*] (use vertical lines)
	`--with-steps`[=*varspec*] (use perpendicular line segments)
	`--time-series` (plot against time)
	`--single-yaxis` (force use of just one y-axis)
	`--y2axis`=*yvar* (put specified variable on second y-axis)
	`--ylogscale`[=*base*] (use log scale for vertical axis)
	`--control` (see below)
	`--dummy` (see below)
	`--fit`=*fitspec* (see below)
	`--font`=*fontspec* (see below)
	`--band`=*bandspec* (see below)
	`--matrix`=*name* (plot columns of named matrix)
	`--output`=*filename* (send output to specified file)
	`--outbuf`=*stringname* (send output to specified string)
	`--input`=*filename* (take input from specified file)
	`--inbuf`=*stringname* (take input from specified string)
Examples:	`gnuplot y1 y2 x`
	`gnuplot x --time-series --with-lines`
	`gnuplot wages educ gender --dummy`
	`gnuplot y x --fit=quadratic`
	`gnuplot y1 y2 x --with-lines=y2`

The series in the list *yvars* are graphed against *xvar*. For a time series plot you may either give `time` as *xvar* or use the option flag `--time-series`. See also the plot and panplot commands.

By default, data-points are shown as points; this can be overridden by giving one of the options `--with-lines`, `--with-lp` (lines and points), `--with-impulses` or `--with-steps`. If more than one variable is to be plotted on the y axis, the effect of these options may be confined to a subset of the variables by using the *varspec* parameter. This should take the form of a comma-separated listing of the names or numbers of the variables to be plotted with lines or impulses respectively. For instance, the final example above shows how to plot `y1` and `y2` against `x`, such that `y2` is represented by a line but `y1` by points.

When *yvars* contains more than one variable it may be preferable to use two y axes (left and right). By default this is handled automatically, via a heuristic based on the relative scales of the variables, but two (mutually exclusive) options can be used to override the default. As you'd expect, `--single-yaxis` prevents use of a second axis, while `--y2axis`=*yvar* specifies that a selected variable (only) be plotted relative to a second axis.

If the `--dummy` option is selected, exactly three variables should be given: a single y variable, an x variable, and *dvar*, a discrete variable. The effect is to plot *yvar* against *xvar* with the points shown in different colors depending on the value of *dvar* at the given observation.

The `--control` option is similar, in that exactly three variables should be given: a single y variable and two "explanatory" variables x and z. The effect is to plot y against x controlling for z. Such plot can be useful to visualize the relationship between x and y, taking into account the effect that z can have on both. Statistically, this would be equivalent to a regression of y on x *and* z.

You can specify the scale for the y axis as logarithmic rather than linear by using the `--ylogscale`

option, together with a base parameter. For example,

```
gnuplot y x --ylogscale=2
```

plots the data such that the vertical axis is expressed as powers of 2. If the base is omitted, it defaults to 10.

Taking data from a matrix

In the primary case the arguments *yvars* and *xvar* are required, and refer to series in the current dataset (given either by name or ID number). But if a named matrix is supplied via the `--matrix` option these arguments become optional: if the specified matrix has k columns, by default the first $k - 1$ columns are treated as the *yvars* and the last column as *xvar*. But if the `--time-series` option is given all k columns are plotted against time. If you wish to plot selected columns of the matrix, you should specify *yvars* and *xvar* in the form of 1-based column numbers. For example if you want a scatterplot of column 2 of matrix `M` against column 1, you can do:

```
gnuplot 2 1 --matrix=M
```

Showing a line of best fit

The `--fit` option is applicable only for bivariate scatterplots and single time-series plots. The default behavior for a scatterplot is to show the OLS fit if the slope coefficient is significant at the 10 percent level, while the default behavior for time-series is not to show any fitted line. You can call for different behavior by using this option along with one of the following *fitspec* parameter values. Note that if the plot is a single time series the place of x is taken by time.

- `linear`: show the OLS fit regardless of its level of statistical significance.
- `none`: don't show any fitted line.
- `inverse`, `quadratic`, `cubic`, `semilog` or `linlog`: show a fitted line based on a regression of the specified type. By `semilog`, we mean a regression of log y on x; the fitted line represents the conditional expectation of y, obtained by exponentiation. By `linlog` we mean a regression of y on the log of x.
- `loess`: show the fit from a robust locally weighted regression (also is sometimes known as "lowess").

Plotting a band

The `--band` option can be used for plotting a "band" of some sort (typically representing a confidence interval) along with other data. The recommended way of specifying such a band is via a bundle, whose name is given as a parameter to the option. A `band` bundle has two required elements: under the key `center`, the name of a series for the center of the band, and under the key `width` the name of a series representing the width of the band—both being given as quoted strings. In addition four optional elements are supported, as follows.

- Under the key `factor`, a scalar giving a factor by which width should be multiplied (the default value being 1).
- Under the key `style`, a string to specify how the band is represented, which must be one of `line` (the default) `fill`, `dash`, `bars` or `step`.
- Under the key `color`, a color for the band, either as a string holding a gnuplot color name or as a hexadecimal RGB representation (given as a string or a scalar). By default the color is selected automatically.

- Under the key `title`, a title for the band, to appear in the key or legend of the plot. By default bands are untitled.

Note that you can access the list of color names recognized by gnuplot by issuing the command "`show colornames`" in gnuplot itself, or in the gretl console by doing

```
eval readfile("@gretldir/data/gnuplot/gpcolors.txt")
```

Here are two examples of usage, employing the shorthand syntax `_()` for defining a bundle. The first just satisfies the minimum requirement while the second exercises all three options. We assume that `y`, `x` and `w` are all series in the current dataset.

```
bundle b1 = _(center="x", width="w")
gnuplot y --time-series --with-lines --band=b1
bundle b2 = _(center="x", width="w", factor=1.96, style="fill")
b2.color=0xcccccc
b2.title = "95% interval"
gnuplot y --time-series --with-lines --band=b2
```

If the plot is to contain two or more such bands, the option flag should be given in the plural and its parameter must be the name of an *array* of bundles, as in (following on from the example above):

```
bundles bb = defarray(b1, b2)
gnuplot y --time-series --with-lines --bands=bb
```

When plotting matrix rather than series data, the only difference is that the `center` and `width` elements of the band bundle are replaced by a single element under the key `bandmat`; this should be the quoted name of a two-column matrix with the center in column 1 and the width in column 2.

Legacy band syntax

The syntax described above was introduced in gretl 2023c. Prior to that release only one band could be specified per plot and the syntax was somewhat different. The old approach, which is still accepted until further notice, split the band details over two options. First, the `--band` option required as parameter the names of two series, separated by a comma, giving center and width. The multiplicative `factor` for the width could be added as a third comma-separated term. Examples:

```
gnuplot ... --band=x,w
gnuplot ... --band=x,w,1.96
```

A second option, `--band-style`, accepted one or both of the style and color specifiers, in that order and again separated by a comma, as in these examples.

```
gnuplot ... --band-style=fill
gnuplot ... --band-style=dash,0xbbddff
gnuplot ... --band-style=,black
gnuplot ... --band-style=bars,blue
```

Recession bars

The "band" option can also be used to add "recession bars" to a plot. By this we mean vertical bars occupying the full y-dimension of the plot and indicating the presence (bar) or absence (no bar) of some qualitative feature in a time-series plot. Such bars are commonly used to flag periods of

recession; they could also be used to indicate periods of war, or anything that can be coded in a 0/1 dummy variable.

In this context the band bundle has a single required element: under the key `dummy`, the quoted name of a 0/1 series (or in the case of matrix data, the quoted name of a suitable column vector). The vertical bars will be "on" for observations where this series or vector has value 1 and "off" when it's 0. The `center`, `width`, `factor` and `style` keys are not relevant, but `color` can be used. Note that only one such specification can be used per plot. Here's an example:

```
open AWM17 --quiet
series dum = obs >= 1990:1 && obs <= 1994:2
bundle b = _(dummy="dum", color=0xcccccc)
gnuplot YER URX --with-lines --time-series \
  --band=b --output=display {set key top left;}
```

Controlling the output

In interactive mode the plot is displayed immediately. In batch mode the default behavior is that a gnuplot command file is written in the user's working directory, with a name on the pattern `gpttmpN.plt`, starting with N = `01`. The actual plots may be generated later using gnuplot (under MS Windows, wgnuplot). This behavior can be modified by use of the `--output=`*filename* option. This option controls the filename used, and at the same time allows you to specify a particular output format via the three-letter extension of the file name, as follows: `.eps` results in the production of an Encapsulated PostScript (EPS) file; `.pdf` produces PDF; `.png` produces PNG format, `.emf` calls for EMF (Enhanced MetaFile), `.fig` calls for an Xfig file, `.svg` for SVG (Scalable Vector Graphics) and `.html` for an HTML canvas. If the dummy filename "`display`" is given then the plot is shown on screen as in interactive mode. If a filename with any extension other than those just mentioned is given, a gnuplot command file is written.

An alternative means of directing output is the `--outbuf=`*stringname* option. This writes gnuplot commands to the named string or "buffer". Note that `--output` and `--outbuf` are mutually incompatible.

Specifying a font

The `--font` option can be used to specify a particular font for the plot. The *fontspec* parameter should take the form of the name of a font, optionally followed by a size in points separated from the name by a comma or space, all wrapped in double quotes, as in

```
--font="serif,12"
```

Note that the fonts available to gnuplot will vary by platform, and if you're writing a plot command that is intended to be portable it is best to restrict the font name to the generic `sans` or `serif`.

Adding gnuplot commands

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: /View/Graph specified vars

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

graphpg

Variants: `graphpg add`
`graphpg fontscale` *value*
`graphpg show`
`graphpg free`
`graphpg --output=`*filename*

The session "graph page" will work only if you have the LaTeX typesetting system installed, and are able to generate and view PDF or PostScript output.

In the session icon window, you can drag up to eight graphs onto the graph page icon. When you double-click on the graph page (or right-click and select "Display"), a page containing the selected graphs will be composed and opened in a suitable viewer. From there you should be able to print the page.

To clear the graph page, right-click on its icon and select "Clear".

Note that on systems other than MS Windows, you may have to adjust the setting for the program used to view PDF or PostScript files. Find that under the "Programs" tab in the gretl Preferences dialog box (under the Tools menu in the main window).

It's also possible to operate on the graph page via script, or using the console (in the GUI program). The following commands and options are supported:

To add a graph to the graph page, issue the command `graphpg add` after saving a named graph, as in

```
grf1 <- gnuplot Y X
graphpg add
```

To display the graph page: `graphpg show`.

To clear the graph page: `graphpg free`.

To adjust the scale of the font used in the graph page, use `graphpg fontscale` *scale*, where *scale* is a multiplier (with a default of 1.0). Thus to make the font size 50 percent bigger than the default you can do

```
graphpg fontscale 1.5
```

To call for printing of the graph page to file, use the flag `--output=` plus a filename; the filename should have the suffix "`.pdf`", "`.ps`" or "`.eps`". For example:

```
graphpg --output="myfile.pdf"
```

The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification.

In this context the output uses colored lines by default; to use dot/dash patterns instead of colors you can append the `--monochrome` flag.

gridplot

Argument:	*plotspecs*
Options:	`--fontsize`=*fs* (font size in points [10])
	`--width`=*w* (width of plot in pixels [800])
	`--height`=*h* (height of plot in pixels [600])
	`--title`=*quoted string* (add an overall title)
	`--rows`=*r* (see below)
	`--cols`=*c* (see below)
	`--layout`=*lmat* (see below)
	`--output`=*destination* (see below)
	`--outbuf`=*alternative destination* (see below)
Examples:	`gridplot myspecs --rows=3 --output=display`
	`gridplot myspecs --layout=lmat --output=composite.pdf`

This command takes two or more individual plot specifications and arranges them in a grid to produce a composite plot. The single required argument, *plotspecs*, takes the form of an array of strings, each specifying a plot. The companion command gpbuild offers an easy way of creating such an array.

Specifying the grid

The shape of the grid can be set by any one of the three mutually incompatible options `--rows`, `--cols` and `--layout`. If no such option is given the number of rows is set to the square root of the number of plots (the size of the input array), rounded up to the nearest integer if necessary. Then the number of columns is set to the number of plots divided by the number of rows, again rounded up if necessary. The plots are placed in the grid by row, in their array order. If the `--rows` option is given this takes the place of the automatic setting, but the number of columns is set automatically as described above. If the `--cols` option is given instead, the number of rows is set automatically.

The `--layout` option, which requires a matrix parameter, offers a more flexible alternative. The matrix specifies the grid layout thus: 0 elements call for empty cells in the grid, and integer elements 1 to n refer to the subplots in their array order. So for example,

```
matrix m = {1,0,0; 2,3,0; 4,5,6}
gridplot ... --layout=m ...
```

calls for a lower-triangular layout of six plots in a 3×3 grid. Using this option one can omit some subplots, or even repeat some.

Output options

The `--output` option can be used to specify `display` (show the plot immediately) or the name of an output file. Alternatively `--outbuf` can be used to direct output, in the form of a gnuplot commands buffer, to a named string. In the absence of these options the output is an automatically named gnuplot command file. See gnuplot for details.

gpbuild

Argument:	*plotspecs*
Example:	`gpbuild MyPlots`

This command starts a block in which any commands or function calls which produce plots are treated specially, in order to produce an array of plot-specification strings for use with gridplot: the *plotspecs* argument supplies the name for this array. Such a block is terminated by the command "`end gpbuild`".

Two restrictions

Within a `gpbuild` block only plotting commands get special treatment; all other commands are executed normally. There are just two restrictions to note.

- Plotting commands should *not* include an output specification in this context, since that would conflict with the automatic redirection of output to the *plotspecs* array. An exception to this rule is allowed for `--output=display` (which is quite common as the default in plot-related function packages); this directive is silently ignored in favour of the automatic treatment.
- Plots that invoke gnuplot's "`multiplot`" directive are not suitable for inclusion in a `gpbuild` block. That is because `gridplot` employs `multiplot` internally, and these constructs cannot be nested.

Manual alternative

It is possible to prepare an array of plot specifications for use with `gridplot` without using a `gpbuild` block, as in the following example:

```
open data4-10
strings MyPlots = array(3)
gnuplot ENROLL CATHOL --outbuf=MyPlots[1]
gnuplot ENROLL INCOME --outbuf=MyPlots[2]
gnuplot ENROLL COLLEGE --outbuf=MyPlots[3]
```

The above is essentially equivalent to

```
open data4-10
gpbuild MyPlots
   gnuplot ENROLL CATHOL
   gnuplot ENROLL INCOME
   gnuplot ENROLL COLLEGE
end gpbuild
```

heckit

Arguments:	*depvar indepvars* ; *selection equation*
Options:	`--quiet` (suppress printing of results)
	`--two-step` (perform two-step estimation)
	`--vcv` (print covariance matrix)
	`--opg` (OPG standard errors)
	`--robust` (QML standard errors)
	`--cluster`=*clustvar* (see logit for explanation)
	`--verbose` (print extra output)
Example:	`heckit y 0 x1 x2 ; ys 0 x3 x4`
	`heckit.inp`

Heckman-type selection model. In the specification, the list before the semicolon represents the outcome equation, and the second list represents the selection equation. The dependent variable in the selection equation (`ys` in the example above) must be a binary variable.

By default, the parameters are estimated by maximum likelihood. The covariance matrix of the parameters is computed using the negative inverse of the Hessian. If two-step estimation is desired, use the `--two-step` option. In this case, the covariance matrix of the parameters of the outcome equation is appropriately adjusted as per Heckman (1979).

Menu path: /Model/Limited dependent variable/Heckit

help

Variants: `help`
`help functions`
`help` *command*
`help` *function*
Option: `--func` (select functions help)

If no arguments are given, prints a list of available commands. If the single argument `functions` is given, prints a list of available functions (see genr).

`help` *command* describes *command* (e.g. `help smpl`). `help` *function* describes *function* (e.g. `help ldet`). Some functions have the same names as related commands (e.g. `diff`): in that case the default is to print help for the command, but you can get help on the function by using the `--func` option.

Menu path: /Help

hfplot

Arguments: *hflist* `[ ;` *lflist* `]`
Options: `--with-lines` (plot with lines)
`--time-series` (put time on x-axis)
`--output=`*filename* (send output to specified file)

Provides a means of plotting a high-frequency series, possibly along with one or more series observed at the base frequency of the dataset. The first argument should be a MIDAS list; the optional additional *lflist* terms, following a semicolon, should be regular ("low-frequency") series.

For details on the effect of the `--output` option, please see the gnuplot command.

hsk

Arguments: *depvar indepvars*
Options: `--no-squares` (see below)
`--vcv` (print covariance matrix)
`--quiet` (don't print anything)

This command is applicable where heteroskedasticity is present in the form of an unknown function of the regressors which can be approximated by a quadratic relationship. In that context it offers the possibility of consistent standard errors and more efficient parameter estimates as compared with OLS.

The procedure involves (a) OLS estimation of the model of interest, followed by (b) an auxiliary regression to generate an estimate of the error variance, then finally (c) weighted least squares, using as weight the reciprocal of the estimated variance.

In the auxiliary regression (b) we regress the log of the squared residuals from the first OLS on the original regressors and their squares (by default), or just on the original regressors (if the `--no-squares` option is given). The log transformation is performed to ensure that the estimated variances are all non-negative. Call the fitted values from this regression u^*. The weight series for the final WLS is then formed as $1/\exp(u^*)$.

Menu path: /Model/Other linear models/Heteroskedasticity corrected

hurst

Argument: *series*
Option: `--plot=`*mode-or-filename* (see below)

Calculates the Hurst exponent (a measure of persistence or long memory) for a time-series variable having at least 128 observations. The result (together with its standard error) can be retrieved via the $result accessor.

The Hurst exponent is discussed by Mandelbrot (1983). 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.

By default, if gretl is not in batch mode a plot of the rescaled range is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Menu path: /Variable/Hurst exponent

if

Flow control for command execution. Three sorts of construction are supported, as follows.

```
# simple form
if condition
    commands
endif

# two branches
if condition
    commands1
else
    commands2
endif

# three or more branches
if condition1
    commands1
elif condition2
    commands2
else
    commands3
endif
```

condition must be a Boolean expression, for the syntax of which see genr. More than one `elif` block may be included. In addition, `if ... endif` blocks may be nested.

include

Argument:	*filename*
Option:	`--force` (force re-reading from file)
Examples:	`include myfile.inp`
	`include sols.gfn`

Intended for use in a command script, primarily for including definitions of functions. *filename* should have the extension `inp` (a plain-text script) or `gfn` (a gretl function package). The commands in *filename* are executed then control is returned to the main script.

The `--force` option is specific to `gfn` files: its effect is to force gretl to re-read the function package from file even if it is already loaded into memory. (Plain `inp` files are always read and processed in response to this command.)

See also run.

info

Variants:	`info`
	`info --to-file=`*filename*
	`info --from-file=`*filename*

In its basic form, displays any supplementary information (metadata) stored with the current datafile. Otherwise, writes this information to file (with the `--to-file` option), or reads metadata from a specified file and attaches it to the current dataset (with `--from-file`, in which case the text should be valid UTF-8).

Menu path: /Data/Dataset info

intreg

Arguments:	*minvar maxvar indepvars*
Options:	`--quiet` (suppress printing of results)
	`--verbose` (print details of iterations)
	`--robust` (robust standard errors)
	`--opg` (see below)
	`--cluster=`*clustvar* (see logit for explanation)
Example:	`intreg lo hi const x1 x2`
	`wtp.inp`

Estimates an interval regression model. This model arises when the dependent variable is imperfectly observed for some (possibly all) observations. In other words, the data generating process is assumed to be

$$y_t^* = x_t\beta + \epsilon_t$$

but we only observe

$$m_t \leq y_t \leq M_t$$

(the interval may be left- or right-unbounded). Note that for some observations m may equal M. The variables *minvar* and *maxvar* must contain `NA`s for left- and right-unbounded observations, respectively.

The model is estimated by maximum likelihood, assuming normality of the disturbance term.

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. Alternatively, the `--opg` option can be given, in which case standard errors are based on the outer product of the gradient alone.

Menu path: /Model/Limited dependent variable/Interval regression

johansen

Arguments:	*order ylist* `[ ;` *xlist* `] [ ;` *rxlist* `]`
Options:	`--nc` (no constant)
	`--rc` (restricted constant)
	`--uc` (unrestricted constant)
	`--crt` (constant and restricted trend)
	`--ct` (constant and unrestricted trend)
	`--seasonals` (include centered seasonal dummies)
	`--asy` (record asymptotic p-values)
	`--quiet` (print just the tests)
	`--silent` (don't print anything)
	`--verbose` (print details of auxiliary regressions)
Examples:	`johansen 2 y x`
	`johansen 4 y x1 x2 --verbose`
	`johansen 3 y x1 x2 --rc`
	See also `hamilton.inp`, `denmark.inp`

Carries out the Johansen test for cointegration among the variables in *ylist* for the given lag order. For details of this test see chapter 33 of the *Gretl User's Guide* or Hamilton (1994), Chapter 20. P-values are computed via Doornik's gamma approximation (Doornik, 1998). Two sets of p-values are shown for the trace test, straight asymptotic values and values adjusted for the sample size. By default the $pvalue accessor gets the adjusted variant, but the `--asy` flag may be used to record the asymptotic values instead.

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 chapter 33 of the *Gretl User's Guide*.

The optional lists *xlist* and *rxlist* allow you to control for specified exogenous variables: these enter the system either unrestrictedly (*xlist*) or restricted to the cointegration space (*rxlist*). These lists are separated from *ylist* and from each other by semicolons.

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. H_0 denotes the null hypothesis, H_1 the alternative hypothesis, and c the number of cointegrating relations.

	Trace test		λ-max test	
Rank	H_0	H_1	H_0	H_1
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, and coint if you want the Engle–Granger cointegration test.

Menu path: /Model/Multivariate time series

join

Arguments: *filename varname*
Options: `--data`=*column-name* (see below)
`--filter`=*expression* (see below)
`--ikey`=*inner-key* (see below)
`--okey`=*outer-key* (see below)
`--aggr`=*method* (see below)
`--tkey`=*column-name,format-string* (see below)
`--verbose` (report on progress)

This command imports one or more data series from the source *filename* (which must be either a delimited text data file or a "native" gretl data file) under the name *varname*. For details please see chapter 7 of the *Gretl User's Guide*; here we just give a brief summary of the available options. See also append for simpler joining operations.

The `--data` option can be used to specify the column heading of the data in the source file, if this differs from the name by which the data should be known in gretl.

The `--filter` option can be used to specify a criterion for filtering the source data (that is, selecting a subset of observations).

The `--ikey` and `--okey` options can be used to specify a mapping between observations in the current dataset and observations in the source data (for example, individuals can be matched against the household to which they belong).

The `--aggr` option is used when the mapping between observations in the current dataset and the source is not one-to-one.

The `--tkey` option is applicable only when the current dataset has a time-series structure. It can be used to specify the name of a column containing dates to be matched to the dataset and/or the format in which dates are represented in that column.

Importing more than one series at once

The `join` command can handle the importation of several series at once. This happens when (a) the *varname* argument is a space-separated list of names rather than a single name, or (b) when it points to an array of strings: the elements of this array should be the names of the series to import.

This methods has some limitations, however: the `--data` option is not available. When importing multiple series you are obliged to accept their "outer" names. The other options apply uniformly to all the series imported via a given command.

kdplot

Argument:	*y*
Options:	`--alt` (use Epanechnikov kernel)
	`--scale`=*s* (scale factor for bandwidth)
	`--output`=*filename* (send plot to specified file)

Plots a kernel density estimate for the series *y*. By default the kernel is Gaussian but if the `--alt` flag is given the Epanechnikov kernel is used. The degree of smoothing can be adjusted via the `--scale` option, which has a default value of 1.0 (higher values of *s* produce a smoother result).

The option `--output` has the effect of sending the output to the specified file; use "display" to force output to the screen. See the gnuplot command for more detail on this option.

For a more flexible means of generating kernel density estimates, with the option of retrieving the result as a matrix, see the kdensity function.

Menu path: /Variable/Estimated density plot

kpss

Arguments:	*order varlist*
Options:	`--trend` (include a trend)
	`--seasonals` (include seasonal dummies)
	`--verbose` (print regression results)
	`--quiet` (suppress printing of results)
	`--difference` (use first difference of variable)
Examples:	`kpss 8 y`
	`kpss 4 x1 --trend`

For use of this command with panel data please see the final section in this entry.

Computes the KPSS test (Kwiatkowski *et al.*, 1992) for stationarity, for each of the specified variables (or their first difference, if the `--difference` option is selected). 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 a negative value is given this is taken as a signal to use an automatic window size of $4(T/100)^{0.25}$, where T is the sample size.

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.

The critical values shown for the test statistic are based on response surfaces estimated in the manner set out by Sephton (1995), which are more accurate for small samples than the values given in the original KPSS article. When the test statistic lies between the 10 percent and 1 percent critical values a p-value is shown; this is obtained by linear interpolation and should not be taken too literally. See the kpsscrit function for a means of obtaining these critical values programmatically.

Panel data

When the `kpss` command is used with panel data, to produce a panel unit root test, the applicable options and the results shown are somewhat different. While you may give a list of variables for testing in the regular time-series case, with panel data only one variable may be tested per command. And the `--verbose` option has a different meaning: it produces a brief account of the test for each individual time series (the default being to show only the overall result).

When possible, the overall test (null hypothesis: the series in question is stationary for all the

panel units) is calculated using the method of Choi (2001). This is not always straightforward, the difficulty being that while the Choi test is based on the p-values of the tests on the individual series, we do not currently have a means of calculating p-values for the KPSS test statistic; we must rely on a few critical values.

If the test statistic for a given series falls between the 10 percent and 1 percent critical values, we are able to interpolate a p-value. But if the test falls short of the 10 percent value, or exceeds the 1 percent value, we cannot interpolate and can at best place a bound on the global Choi test. If the individual test statistic falls short of the 10 percent value for some units but exceeds the 1 percent value for others, we cannot even compute a bound for the global test.

Menu path: /Variable/Unit root tests/KPSS test

labels

Variants: `labels [` *varlist* `]`
`labels --to-file=`*filename*
`labels --from-file=`*filename*
`labels --delete`
Example: `oprobit.inp`

In the first form, prints out the informative labels (if present) for the series in *varlist*, or for all series in the dataset if *varlist* is not specified.

With the option `--to-file`, writes to the named file the labels for all series in the dataset, one per line. If no labels are present an error is flagged; if some series have labels and others do not, a blank line is printed for series with no label. The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification.

With the option `--from-file`, reads the specified file (which should be plain text) and assigns labels to the series in the dataset, reading one label per line and taking blank lines to indicate blank labels.

The `--delete` option does what you'd expect: it removes all the series labels from the dataset.

Menu path: /Data/Variable labels

lad

Arguments: *depvar indepvars*
Options: `--vcv` (print covariance matrix)
`--no-vcv` (don't compute covariance matrix)
`--quiet` (don't print anything)

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. Since this is quite an expensive operation, the `--no-vcv` option is provided for the case where the covariance matrix is not required; when this option is given standard errors will not be available.

Note that this method can be slow when the sample is large or there are many regressors; in that case it may be preferable to use the quantreg command. Given a dependent variable `y` and a list of regressors `X`, the following commands are basically equivalent, except that the quantreg method uses the faster Frisch–Newton algorithm and provides analytical rather than bootstrapped standard errors.

```
lad y const X
quantreg 0.5 y const X
```

Menu path: /Model/Robust estimation/Least Absolute Deviation

lags

Arguments: [*order* ;] *laglist*
Option: `--bylag` (order terms by lag)
Examples: `lags x y`
`lags 12 ; x y`
`lags 4 ; x1 x2 x3 --bylag`
See also `sw_ch12.inp`, `sw_ch14.inp`

Creates new series which are lagged values of each of the series in *varlist*. By default the number of lags created 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 (which, if present, must be followed by a semicolon).

The `--bylag` option is meaningful only if *varlist* contains more than one series and the maximum lag order is greater than 1. By default the lagged terms are added to the dataset by variable: first all lags of the first series, then all lags of the second series, and so on. But if `--bylag` is given, the ordering is by lags: first lag 1 of all the listed series, then lag 2 of all the list series, and so on.

This facility is also available as a function: see lags.

Menu path: /Add/Lags of selected variables

ldiff

Argument: *varlist*

The first difference of the natural log of each series in *varlist* is obtained and the result stored in a new series 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: /Add/Log differences of selected variables

leverage

Options: `--save` (save the resulting series)
`--overwrite` (OK to overwrite existing series)
`--quiet` (don't print results)
`--plot`=*mode-or-filename* (see below)
Example: `leverage.inp`

Must 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 computed: these are Studentized residuals (residuals divided by their standard errors) multiplied by $\sqrt{h/(1-h)}$. They give a measure of the difference in fit for observation i depending on whether or not that observation is included in the sample for estimation. For more on this point see chapter 12 of Maddala's Maddala (1992) or Belsley *et al.* (1980). For more on Studentized residuals see the section headed *Accessor matrix* below.

If the `--save` flag is given with this command, the leverage, influence and DFFITS values are added to the current data set; in this context the `--quiet` flag may be used to suppress the printing of results. The default names of the saved series are, respectively, `lever`, `influ` and `dffits`. If series of these names already exist, what happens depends on whether the `--overwrite` option is given. If so, the existing series are overwritten; if not, the names will be adjusted to ensure uniqueness. In the latter case the newly generated series will always be the highest-numbered three series in the dataset.

After execution, the $test accessor returns the cross-validation criterion, which is defined as

$$\sum_{i=1}^{n}(y_i - \hat{y}_{-i})^2$$

where $\hat{y}_{-i}$ is the forecast error for the i-th observation, after it has been excluded from the sample. The criterion is, hence, the sum of the squared forecasting errors where all n observations but the i-th one are used to predict it (the so-called *leave-one-out* estimator). For a broader discussion of the cross-validation criterion, see Davidson and MacKinnon's *Econometric Theory and Methods*, pages 685–686, and the references therein.

By default, if this command is invoked interactively a plot of the leverage and influence values is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when in script mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Accessor matrix

Besides the `--save` option noted above, results from this command can be retrieved in the form of a three-column matrix via the $result accessor. The first two columns of this matrix contain leverage and influence values (as with `--save`) but the third column holds Studentized residuals rather than DFFITS values. These are "externally Studentized" or "jackknifed" residuals—that is, the standard error in the divisor for observation i uses the residual mean square with that observation omitted. Such a residual can be interpreted as a t statistic for the hypothesis that a 0/1 dummy variable coding specifically for observation i would have a true coefficient of zero. For further discussion of Studentized residuals see Chatterjee and Hadi (1986).

DFFITS values may be obtained from the `$result` matrix as follows:

```
R = $result
dffits = R[,3] .* sqrt(R[,1] ./ (1-R[,1]))
```

Or using series:

```
series h = $result[,1]  # leverage
series sr = $result[,3] # Studentized residual
series dffits = sr * sqrt(h/(1-h))
```

Menu path: Model window, /Analysis/Influential observations

levinlin

Arguments:	*order series*
Options:	`--nc` (test without a constant)
	`--ct` (with constant and trend)
	`--quiet` (suppress printing of results)
	`--verbose` (print per-unit results)
Examples:	`levinlin 0 y`
	`levinlin 2 y --ct`
	`levinlin {2,2,3,3,4,4} y`

Carries out the panel unit-root test described by Levin *et al.* (2002). The null hypothesis is that all of the individual time series exhibit a unit root, and the alternative is that none of the series has a unit root. (That is, a common AR(1) coefficient is assumed, although in other respects the statistical properties of the series are allowed to vary across individuals.)

By default the test ADF regressions include a constant; to suppress the constant use the `--nc` option, or to add a linear trend use the `--ct` option. (See the adf command for explanation of ADF regressions.)

The (non-negative) *order* for the test (governing the number of lags of the dependent variable to include in the ADF regressions) may be given in either of two forms. If a scalar value is given, this is applied to all the individuals in the panel. The alternative is to provide a matrix containing a specific lag order for each individual; this must be a vector with as many elements as there are individuals in the current sample range. Such a matrix can be specified by name, or constructed using braces as illustrated in the last example above.

When the `--verbose` option is given, the following results are printed for each unit in the panel: `delta`, the coefficient on the lagged level in each ADF regression; `s2e`, the estimated variance of the innovations; and `s2y`, the estimated long-run variance of the differenced series.

Note that panel unit-root tests can also be conducted using the adf and kpss commands.

Menu path: /Variable/Unit root tests/Levin-Lin-Chu test

logistic

Arguments:	*depvar indepvars*
Options:	`--ymax=`*value* (specify maximum of dependent variable)
	`--robust` (robust standard errors)
	`--cluster=`*clustvar* (see logit for explanation)
	`--vcv` (print covariance matrix)
	`--fixed-effects` (see below)
	`--quiet` (don't print anything)
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,

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

In the case of panel data the specification may include individual fixed effects.

The dependent variable must be strictly positive. If all its values lie between 0 and 1, the default is to use a y^* value (the asymptotic maximum of the dependent variable) of 1; if its values lie between 0 and 100, the default y^* is 100.

If you wish to set a different maximum, use the `--ymax` option. Note that 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 adjusted using the inverse of the logistic transformation:

$$y \approx E\left(\frac{y^*}{1+e^{-x}}\right)$$

where x represents either a fitted value or a residual from the OLS regression using the logistic dependent variable. The reported values are therefore comparable with the original dependent variable. The need for approximation arises because the inverse transformation is nonlinear and therefore does not conserve expectation.

The `--fixed-effects` option is applicable only if the dataset takes the form of a panel. In that case we subtract the group means from the logistic transform of the dependent variable and estimation proceeds as usual for fixed effects.

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

Menu path: /Model/Limited dependent variable/Logistic

Menu path: /Model/Panel/FE logistic

logit

Arguments:	*depvar indepvars*
Options:	`--robust` (robust standard errors)
	`--cluster`=*clustvar* (clustered standard errors)
	`--multinomial` (estimate multinomial logit)
	`--vcv` (print covariance matrix)
	`--verbose` (print details of iterations)
	`--quiet` (don't print results)
	`--p-values` (show p-values instead of slopes)
	`--estrella` (select pseudo-R-squared variant)
Examples:	`keane.inp`, `oprobit.inp`

If the dependent variable is a binary variable (all values are 0 or 1) maximum likelihood estimates of the coefficients on *indepvars* are obtained via the Newton–Raphson method. As the model is nonlinear the slopes depend on the values of the independent variables. By default the slopes with respect to each of the independent variables are calculated (at the means of those variables) and these slopes replace the usual p-values in the regression output. This behavior can be suppressed by giving the `--p-values` option. 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 chapter 10 of Davidson and MacKinnon (2004). But if the `--cluster` option is given, then "cluster-robust" standard errors are produced; see chapter 22 of the *Gretl User's Guide* for details.

By default the pseudo-R-squared statistic suggested by McFadden (1974) is shown, but in the binary case if the `--estrella` option is given, the variant recommended by Estrella (1998) is shown instead. This variant arguably mimics more closely the properties of the regular R^2 in the context of least-squares estimation.

If the dependent variable is binary, logit coefficients represent the log of the odds ratio (the ratio of the probability of $y = 1$ to that of $y = 0$). In this case the `$model` bundle available after estimation includes an the extra element named `oddsratios`, a matrix with four columns holding

the exponentiated coefficient (odds ratio) plus standard error computed via the delta method and 95 percent confidence interval, for each regressor. Note, however, that the confidence interval is calculated as the exponential of that for the original coefficient.

If the dependent variable is not binary but is discrete, then by default it is interpreted as an ordinal response, and Ordered Logit estimates are obtained. However, if the `--multinomial` option is given, the dependent variable is interpreted as an unordered response, and Multinomial Logit estimates are produced. (In either case, if the variable selected as dependent is not discrete an error is flagged.) The accessor `$allprobs` is available after estimation, to get a matrix containing the estimated probabilities of the outcomes at each observation (observations in rows, outcomes in columns).

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

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

and use this as the dependent variable in an OLS regression. See chapter 12 of Ramanathan (2002).

Menu path: /Model/Limited dependent variable/Logit

logs

Argument: *varlist*

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

Menu path: /Add/Logs of selected variables

loop

Argument: *control*
Options: `--progressive` (enable special forms of certain commands)
`--verbose` (echo commands and show confirmatory messages)
`--decr` (see below)
Examples:
```
loop 1000
loop i=1..10
loop while essdiff > .00001
loop for (r=-.99; r<=.99; r+=.01)
loop foreach i xlist
```
See also `armaloop.inp`, `keane.inp`

This command opens a special mode in which the program accepts commands to be executed repeatedly. You exit the mode of entering loop commands with `endloop`: at this point the stacked commands are executed.

The parameter *control* may take any of five forms, as shown in the examples: an integer number of times to repeat the commands within the loop; a range of integer values for an index variable; "`while`" plus a boolean condition; "`for`" plus three expressions in parentheses, separated by semicolons (which emulates the `for` statement in the C programming language); or "`foreach`" plus an index variable and a list.

The `--decr` option is specific to the "range of integer values" form of loop. By default the index is incremented by 1 at each iteration, and if the starting value is less than the ending value the loop will not run at all. When `--decr` is given the index is decremented by 1 at each iteration.

See chapter 13 of the *Gretl User's Guide* for full details and examples. The effect of the `--progressive` option (which is designed for use in Monte Carlo simulations) is explained there. Not all gretl commands may be used within a loop; the commands available in this context are also set out there.

By default, execution of commands proceeds more quietly within loops than in other contexts. If you want more feedback on what's going on in a loop, give the `--verbose` option.

mahal

Argument: *varlist*
Options: `--quiet` (don't print anything)
`--save` (add distances to the dataset)
`--vcv` (print covariance matrix)

Computes the Mahalanobis distances of each observation from the centroid, using the series in *varlist*. The Mahalanobis distance is the distance between two points in a 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: /View/Mahalanobis distances

makepkg

Argument: *filename*
Options: `--index` (write auxiliary index file)
`--translations` (write auxiliary strings file)
`--quiet` (operate quietly)

Supports creation of a gretl function package via the command line. The mode of operation of this command depends on the extension of *filename*, which must be either `.gfn` or `.zip`.

Gfn mode

Writes a gfn file. It is assumed that a package specification file, with the same basename as *filename* but with the extension `.spec`, is accessible, along with any auxiliary files that it references. It is also assumed that all the functions to be packaged have been read into memory.

Zip mode

Writes a zip package file (gfn plus other materials). If a gfn file of the same basename as *filename* is found, gretl checks for corresponding `inp` and `spec` files: if these are both found and at least one of them is newer than the gfn file then the gfn is rebuilt, otherwise the existing gfn is used. If no such file is found, gretl first attempts to build the gfn.

Gfn options

The option flags support the writing of auxiliary files, intended for use with gretl "addons". The index file is a short XML document containing basic information about the package; it has the same basename as the package and the extension `.xml`. The translations file contains strings from the package that may be suitable for translation, in C format; for package `foo` this file is named `foo-i18n.c`. These files are not produced if the command is operating in zip mode and a pre-existing gfn file is used.

For details on all of this, see the gretl Function Package Guide.

Menu path: /File/Function packages/New package

markers

Variants: `markers --to-file=`*filename*
`markers --from-file=`*filename*
`markers --to-array=`*name*
`markers --from-array=`*name*
`markers --from-series=`*name*
`markers --delete`

The options `--to-file` and `--to-array` provide means of saving the observation marker strings from the current dataset, either to a named file or a named array. If no such strings are present an error is flagged. In the file case output will be written in the current workdir unless the *filename* string contains a full path specification. The markers are written one per line. In the array case, if *name* is the identifier of an existing array of strings it will be overwritten, otherwise a new array will be created.

With the option `--from-file`, reads the specified file (which should be UTF-8 text) and assigns observation markers to the rows in the dataset, reading one marker per line. In general there should be at least as many markers in the file as observations in the dataset, but if the dataset is a panel it is also acceptable if the number of markers in the file matches the number of cross-sectional units (in which case the markers are repeated for each time period.) The `--from-array` option works similarly, reading from a named array of strings.

The option `--from-series` offers a convenient way of creating observation markers by copying from a string-valued series. An error is flagged if the specified series does not have string values.

The `--delete` option does what you'd expect: it removes the observation marker strings from the dataset.

Menu path: /Data/Observation markers

meantest

Variants:	`meantest` *x y*
	`meantest` *x* `--split-by=`*dummy*
Options:	`--unequal-vars` (assume variances are unequal)
	`--paired` (conduct a paired test)
	`--quiet` (suppress printed output)
	`--robust` (time series only)
Examples:	`meantest x y`
	`meantest x y --unequal-vars`
	`meantest x y --paired`
	`meantest x --split-by=d`

In its primary usage, calculates the t statistic for the null hypothesis that the population means are equal for the series x and y, and shows output including the p-value. Results can be retrieved using the accessors $test and $pvalue, in which case the `--quiet` option may used to omit the printout.

In its alternate form, with the `--split-by` option, the samples whose means are tested for equality are two subsets of the series x, for which the series *dummy* takes the values 0 and 1, respectively.

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; in this case the degrees of freedom for the test statistic are approximated as per Satterthwaite (1946).

Paired test

In the primary case (only) the `--paired` option can be given, to test the null hypothesis that the mean difference between the paired values of the two series arguments is zero. Otherwise, pairing is not assumed.

Robust test

Given time series data, in the primary case (only) the `--robust` option can be given to conduct a test of equality of the means of x and y which is robust in respect of serial correlation.

Menu path: /Tools/Test statistic calculator

midasreg

Arguments:	*depvar indepvars* `;` *MIDAS-terms*
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)
	`--quiet` (suppress printing of results)
	`--levenberg` (see below)
Examples:	`midasreg y 0 y(-1) ; mds(X, 1, 9, 1, theta)`
	`midasreg y 0 y(-1) ; mds(X, 1, 9, 0)`
	`midasreg y 0 y(-1) ; mdsl(XL, 2, theta)`
	See also `gdp_midas.inp`

Carries out least-squares estimation (either NLS or OLS, depending on the specification) of a MIDAS (Mixed Data Sampling) model. Such models include one or more independent variables that are observed at a higher frequency than the dependent variable; for a good brief introduction see Armesto *et al.* (2010).

The variables in *indepvars* should be of the same frequency as the dependent variable. This list should usually include `const` or 0 (intercept) and typically includes one or more lags of the depen-

dent variable. The high-frequency terms are given after a semicolon; each one takes the form of a number of comma-separated arguments within parentheses, prefixed by either `mds` or `mdsl`.

`mds`: this variant generally requires 5 arguments, as follows: the name of a MIDAS list, two integers giving the minimum and maximum high-frequency lags, an integer between 0 and 4 (or string, see below) specifying the type of parameterization to use, and the name of a vector holding initial values of the parameters. The example below calls for lags 3 to 11 of the high-frequency series represented by the list `X`, using parameterization type 1 (exponential Almon, see below) with initializer `theta`.

```
mds(X, 3, 11, 1, theta)
```

`mdsl`: generally requires 3 arguments: the name of a list of MIDAS lags, an integer (or string, see below) to specify the type of parameterization and the name of an initialization vector. In this case the minimum and maximum lags are implicit in the initial list argument. In the example below `Xlags` should be a list which already holds all the required lags; such a list can be constructed using the hflags function.

```
mdsl(XLags, 1, theta)
```

The supported types of parameterization are shown below; in the context of `mds` and `mdsl` specifications these may be given in the form of numeric codes or the double-quoted strings shown after the numbers.

0 or `"umidas"`: unrestricted MIDAS or U-MIDAS (each lag has its own coefficient)

1 or `"nealmon"`: normalized exponential Almon; requires at least one parameter, commonly uses two

2 or `"beta0"`: normalized beta with a zero last lag; requires exactly two parameters

3 or `"betan"`: normalized beta with non-zero last lag; requires exactly three parameters

4 or `"almonp"`: (non-normalized) Almon polynomial; requires at least one parameter

5 or `"beta1"`: as `beta0`, but with the first parameter fixed at 1, leaving a single free parameter.

When the parameterization is U-MIDAS, the final initializer argument is not required. In other cases you can request an automatic initialization by substituting one or other of these two forms for the name of an initial parameter vector:

- The keyword `null`: this is accepted if the parameterization has a fixed number of terms (the beta cases, with 2 or 3 parameters). It's also accepted for the exponential Almon case, implying the default of 2 parameters.
- An integer value giving the required number of parameters.

The estimation method used by this command depends on the specification of the high-frequency terms. In the case of U-MIDAS the method is OLS, otherwise it is nonlinear least squares (NLS). When the normalized exponential Almon or normalized beta parameterization is specified, the default NLS method is a combination of constrained BFGS and OLS, but the `--levenberg` option can be given to force use of the Levenberg–Marquardt algorithm.

Menu path: /Model/Univariate time series/MIDAS

mle

Arguments:	*log-likelihood function* [*derivatives*]
Options:	`--quiet` (don't show estimated model)
	`--vcv` (print covariance matrix)
	`--hessian` (base covariance matrix on the Hessian)
	`--robust`[=*hac*] (QML or HAC covariance matrix)
	`--cluster`=*clustvar* (cluster-robust covariance matrix)
	`--verbose` (print details of iterations)
	`--no-gradient-check` (see below)
	`--auxiliary` (see below)
	`--lbfgs` (use L-BFGS-B instead of regular BFGS)
Examples:	`weibull.inp`, `biprobit_via_ghk.inp`, `frontier.inp`, `keane.inp`

Performs Maximum Likelihood (ML) estimation using either the BFGS (Broyden, Fletcher, Goldfarb, Shanno) algorithm or Newton's method. The user must specify the log-likelihood function. The parameters of this function must be declared and given starting values 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.

This help text assumes use of the default BFGS maximizer. For information on using Newton's method please see chapter 26 of the *Gretl User's Guide*.

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. For example, `scalar 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 and can be either scalars, or vectors, or any combination of the two. 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.

Note that any option flags should be appended to the ending line of the MLE block. For example:

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

Covariance matrix and standard errors

If the log-likelihood function returns a series or vector giving per-observation values then estimated standard errors are by default based on the Outer Product of the Gradient (OPG), while if the `--hessian` option is given they are instead based on the negative inverse of the Hessian, which is approximated numerically. If the `--robust` option is given, a QML estimator is used (namely, a sandwich of the negative inverse of the Hessian and the OPG). If the `hac` parameter is added to this option the OPG is augmented in the manner of Newey and West (1987) to allow for serial correlation of the gradient. (This only makes sense with time-series data.) However, if the log-likelihood function just returns a scalar value, the OPG is not available (and so neither is the QML estimator), and standard errors are of necessity computed using the numerical Hessian.

In the event that you just want the primary parameter estimates you can give the `--auxiliary` option, which suppresses computation of the covariance matrix and standard errors; this will save some CPU cycles and memory usage.

Checking analytical derivatives

If you supply analytical derivatives, by default gretl runs a numerical check on their plausibility. Occasionally this may produce false positives, instances where correct derivatives appear to be wrong and estimation is refused. To counter this, or to achieve a little extra speed, you can give the option `--no-gradient-check`. Obviously, you should do this only if you are confident that the gradient you have specified is right.

Parameter names

In estimating a nonlinear model it is often convenient to name the parameters tersely. In printing the results, however, it may be desirable to use more informative labels. This can be achieved via the additional keyword `param_names` within the command block. For a model with k parameters the argument following this keyword should be a double-quoted string literal holding k space-separated names, the name of a string variable that holds k such names, or the name of an array of k strings.

For an in-depth description of `mle` please refer to chapter 26 of the *Gretl User's Guide*.

Menu path: /Model/Maximum likelihood

modeltab

Variants: `modeltab add`
`modeltab show`
`modeltab free`
`modeltab --output=`*filename*
`modeltab --options=`*bundle*

Manipulates the gretl "model table". See chapter 3 of 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.

To call for printing of the model table, use the flag `--output=` plus a filename. If the filename has the suffix "`.tex`", the output will be in TEX format; if the suffix is "`.rtf`" the output will be RTF; otherwise it will be plain text. In the case of TEX output the default is to produce a "fragment", suitable for inclusion in a document; if you want a stand-alone document instead, use the `--complete` option, for example

```
modeltab --output="myfile.tex" --complete
```

The `--options=` flag, which requires the name of a gretl bundle, can be used to control certain aspects of the formatting of the model table. The following keys are recognized:

- `colheads`: integer from 1 to 4, selects from the four supported column-head styles: Arabic numbering, Roman numbering, alphabetical, or using the names under which models have been saved. The default is 1 (Arabic numbering).
- `tstats`: boolean, replace standard errors with t-statistics or not (default 0).
- `pvalues`: boolean, include *P*-values or not (default 0).
- `asterisks`: boolean, show significance-level asterisks or not (default 1).
- `digits`: integer from 2 to 6, selects the number of significant digits shown (default 4).
- `decplaces`: integer from 2 to 6, selects the number of decimal places shown.

Note that the last two keys are mutually exclusive. They offer alternative ways of specifying the precision to which numerical values are shown: either in terms of significant digits or decimal places. The default is 4 significant digits.

An options bundle can be supplied via a stand-alone command (as in the last of the examples above) or it can be combined with the `show` action or `--output` option. For example, the following script builds a simple model table and displays it, with *P*-values shown instead of significance asterisks:

```
open data9-7
ols 1 0 2 3 4
modeltab add
ols 1 0 2 3
modeltab add
bundle myopts = _(pvalues=1, asterisks=0)
modeltab show --options=myopts
```

Menu path: Session icon window, Model table icon

modprint

Arguments: *coeffmat names* [*addstats*]
Option: `--output`=*filename* (send output to specified file)

Prints the coefficient table and optional additional statistics for a model estimated "by hand". Mainly useful for user-written functions.

The argument *coeffmat* should be a k by 2 matrix containing k coefficients and k associated standard errors. The *names* argument should supply at least k names for labeling the coefficients; it can take the form of a string literal (in double quotes) or string variable, in which case the names should be separated by commas or spaces, or it may be given as a named array of strings.

The optional argument *addstats* is a vector containing p additional statistics to be printed under the coefficient table. If this argument is given, then *names* should contain $k + p$ names, the additional p names to be associated with the extra statistics.

If *addstats* is not provided and the *coeffmat* matrix has row names attached, then the *names* argument can be omitted.

To put the output into a file, use the flag `--output=` plus a filename. If the filename has the suffix "`.tex`", the output will be in TEX format; if the suffix is "`.rtf`" the output will be RTF; otherwise it will be plain text. In the case of TEX output the default is to produce a "fragment", suitable for inclusion in a document; if you want a stand-alone document instead, use the `--complete` option.

The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification.

modtest

Argument:	[*order*]
Options:	`--normality` (normality of residual)
	`--logs` (nonlinearity, logs)
	`--squares` (nonlinearity, squares)
	`--autocorr` (serial correlation)
	`--arch` (ARCH)
	`--white` (heteroskedasticity, White's test)
	`--white-nocross` (White's test, squares only)
	`--breusch-pagan` (heteroskedasticity, Breusch–Pagan)
	`--robust` (robust variance estimate for Breusch–Pagan)
	`--panel` (heteroskedasticity, groupwise)
	`--comfac` (common factor restriction, AR1 models only)
	`--xdepend` (cross-sectional dependence, panel data only)
	`--quiet` (don't print details)
	`--silent` (don't print anything)
Example:	`credscore.inp`

Must immediately follow an estimation command. The discussion below applies to usage of the command following estimation of a single-equation model; see chapter 32 of the *Gretl User's Guide* for an account of how `modtest` operates after estimation of a VAR.

Depending on the option given, this command carries out one of the following: the Doornik–Hansen test for the normality of the error term; a Lagrange Multiplier test for nonlinearity (logs or squares); White's test (with or without cross-products) or the Breusch–Pagan test (Breusch and Pagan (1979)) for heteroskedasticity; the LMF test for serial correlation (Kiviet, 1986); a test for ARCH (Autoregressive Conditional Heteroskedasticity; see also the `arch` command); a test of the common factor restriction implied by AR(1) estimation; or a test for cross-sectional dependence in panel-data models. With the exception of the normality, common factor and cross-sectional dependence tests most of the options are only available for models estimated via OLS, but see below for details regarding two-stage least squares.

The optional `order` argument is relevant only in case the `--autocorr` or `--arch` options are selected. The default is to run these tests using a lag order equal to the periodicity of the data, but this can be adjusted by supplying a specific lag order.

The `--robust` option applies only when the Breusch–Pagan test is selected; its effect is to use the robust variance estimator proposed by Koenker (1981), making the test less sensitive to the assumption of normality.

The `--panel` option is available only when the model is estimated on panel data: in this case a test for groupwise heteroskedasticity is performed (that is, for a differing error variance across the cross-sectional units).

The `--comfac` option is available only when the model is estimated via an AR(1) method such as Hildreth–Lu. The auxiliary regression takes the form of a relatively unrestricted dynamic model, which is used to test the common factor restriction implicit in the AR(1) specification.

The `--xdepend` option is available only for models estimated on panel data. The test statistic is that developed by Pesaran (2004). The null hypothesis is that the error term is independently distributed across the cross-sectional units or individuals.

By default, the program prints the auxiliary regression on which the test statistic is based, where applicable. This may be suppressed by using the `--quiet` flag (minimal printed output) or the `--silent` flag (no printed output). The test statistic and its p-value may be retrieved using the accessors $test and $pvalue respectively.

When a model has been estimated by two-stage least squares (see tsls), the LM principle breaks down and gretl offers some equivalents: the `--autocorr` option computes Godfrey's test for autocorrelation (Godfrey, 1994) while the `--white` option yields the HET1 heteroskedasticity test (Pesaran and Taylor, 1999).

For additional diagnostic tests on models, see chow, cusum, reset and qlrtest.

Menu path: Model window, /Tests

mpi

Argument: *see below*

The `mpi` command starts a block of statements (which must be ended with `end mpi`) to be executed using MPI (Message Passing Interface) parallelization. See `gretl-mpi.pdf` for a full account of this facility.

mpols

Arguments: *depvar indepvars*
Options: `--vcv` (print covariance matrix)
`--simple-print` (do not print auxiliary statistics)
`--quiet` (suppress printing of results)

Computes OLS estimates for the specified model using multiple precision floating-point arithmetic, with the help of the Gnu Multiple Precision (GMP) library. By default 256 bits of precision are used for the calculations, but this can be increased via the environment variable `GRETL_MP_BITS`. For example, when using the bash shell one could issue the following command, before starting gretl, to set a precision of 1024 bits.

```
export GRETL_MP_BITS=1024
```

A rather arcane option is available for this command (primarily for testing purposes): if the *indepvars* list is followed by a semicolon and a further list of numbers, those numbers are taken as powers of x to be added to the regression, where x is the last variable in *indepvars*. These additional terms are computed and stored in multiple precision. In the following example `y` is regressed on `x` and the second, third and fourth powers of `x`:

```
mpols y 0 x ; 2 3 4
```

Menu path: /Model/Other linear models/High precision OLS

negbin

Arguments: *depvar indepvars* `[ ;` *offset* `]`
Options: `--model1` (use NegBin 1 model)
`--robust` (QML covariance matrix)
`--cluster=`*clustvar* (see logit for explanation)
`--opg` (see below)
`--vcv` (print covariance matrix)
`--verbose` (print details of iterations)
`--quiet` (don't print results)
Example: `camtriv.inp`

Estimates a Negative Binomial model. The dependent variable is taken to represent a count of the occurrence of events of some sort, and must have only non-negative integer values. By default the model NegBin 2 is used, in which the conditional variance of the count is given by $\mu(1 + \alpha\mu)$, where μ denotes the conditional mean. But if the `--model1` option is given the conditional variance is $\mu(1 + \alpha)$.

The optional `offset` series works in the same way as for the poisson command. The Poisson model is a restricted form of the Negative Binomial in which $\alpha = 0$ by construction.

By default, standard errors are computed using a numerical approximation to the Hessian at convergence. But if the `--opg` option is given the covariance matrix is based on the Outer Product of the Gradient (OPG), or if the `--robust` option is given QML standard errors are calculated, using a "sandwich" of the inverse of the Hessian and the OPG.

Menu path: /Model/Limited dependent variable/Count data

nls

Arguments:	*function* [*derivatives*]
Options:	`--quiet` (don't show estimated model)
	`--robust` (robust standard errors)
	`--vcv` (print covariance matrix)
	`--verbose` (print details of iterations)
	`--no-gradient-check` (see below)
Examples:	`wg_nls.inp`, `ects_nls.inp`

Performs Nonlinear Least Squares (NLS) estimation using a modified version of the Levenberg–Marquardt algorithm. You must supply a function specification. The parameters of this function must be declared and given starting values prior to estimation. Optionally, you may specify the derivatives of the regression function with respect to each of the parameters. If you do not supply derivatives you should instead give a list of the parameters to be estimated (separated by spaces or commas), preceded by the keyword `params`. In the latter case 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 any option flags, such as `--vcv` for printing the covariance matrix of the parameter estimates, should be appended to the final command, `end nls`.

```
1   open greene11_3.gdt
2   ols C 0 Y
3   scalar a = $coeff(0)
4   scalar b = $coeff(Y)
5   scalar g = 1.0
6   nls C = a + b * Y^g
7    deriv a = 1
8    deriv b = Y^g
9    deriv g = b * Y^g * 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. As an alternative to supplying analytical derivatives, you could substitute the following for lines 7 to 9:

```
params a b g
```

Line 10, end nls, completes the command and calls for estimation. Any options should be appended to this line.

If you supply analytical derivatives, by default gretl runs a numerical check on their plausibility. Occasionally this may produce false positives, instances where correct derivatives appear to be wrong and estimation is refused. To counter this, or to achieve a little extra speed, you can give the option --no-gradient-check. Obviously, you should do this only if you are confident that the gradient you have specified is right.

Parameter names

In estimating a nonlinear model it is often convenient to name the parameters tersely. In printing the results, however, it may be desirable to use more informative labels. This can be achieved via the additional keyword param_names within the command block. For a model with k parameters the argument following this keyword should be a double-quoted string literal holding k space-separated names, the name of a string variable that holds k such names, or the name of an array of k strings.

For further details on NLS estimation please see chapter 25 of the *Gretl User's Guide*.

Menu path: /Model/Nonlinear Least Squares

normtest

Argument:	*series*
Options:	--dhansen (Doornik–Hansen test, the default)
	--swilk (Shapiro–Wilk test)
	--lillie (Lilliefors test)
	--jbera (Jarque–Bera test)
	--all (do all tests)
	--quiet (suppress printed output)

Carries out a test for normality for the given *series*. The specific test is controlled by the option flags (but if no flag is given, the Doornik–Hansen test is performed). Note: the Doornik–Hansen and Shapiro–Wilk tests are recommended over the others, on account of their superior small-sample properties.

The test statistic and its p-value may be retrieved using the accessors $test and $pvalue. Please note that if the --all option is given, the result recorded is that from the Doornik–Hansen test.

Menu path: /Variable/Normality test

nulldata

Argument:	*series_length*
Option:	--preserve (preserve variables other than series)
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: functions

such as `uniform()` and `normal()` will generate artificial series 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.

By default, this command cleans out all data in gretl's current workspace: not only series but also matrices, scalars, strings, etc. If you give the `--preserve` option, however, any currently defined variables other than series are retained.

Menu path: /File/New data set

ols

Arguments:	*depvar indepvars*
Options:	`--vcv` (print covariance matrix)
	`--robust` (robust standard errors)
	`--cluster`=*clustvar* (clustered standard errors)
	`--jackknife` (see below)
	`--simple-print` (do not print auxiliary statistics)
	`--quiet` (suppress printing of results)
	`--anova` (print an ANOVA table)
	`--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 chapter 13 of the *Gretl User's Guide* for details.

Various internal variables may be retrieved following estimation. For example

```
series uh = $uhat
```

saves the residuals under the name `uh`. See the "accessors" section of the gretl function reference for details.

The specific formula ("HC" version) used for generating robust standard errors when the `--robust` option is given can be adjusted via the set command. The `--jackknife` option has the effect of selecting an `hc_version` of `3a`. The `--cluster` overrides the selection of HC version, and produces robust standard errors by grouping the observations by the distinct values of *clustvar*; see chapter 22 of the *Gretl User's Guide* for details.

Menu path: /Model/Ordinary Least Squares

Other access: Beta-hat button on toolbar

omit

Argument:	*varlist*
Options:	`--test-only` (don't replace the current model)
	`--chi-square` (give chi-square form of Wald test)
	`--quiet` (print only the basic test result)
	`--silent` (don't print anything)
	`--vcv` (print covariance matrix for reduced model)
	`--auto[=`*alpha*`]` (sequential elimination, see below)
Examples:	`omit 5 7 9`
	`omit seasonals --quiet`
	`omit --auto`
	`omit --auto=0.05`
	See also `restrict.inp`, `sw_ch12.inp`, `sw_ch14.inp`

This command must follow an estimation command. In its primary form, it calculates a Wald test for the joint significance of the variables in *varlist*, which should be a subset (though not necessarily a proper subset) of the independent variables in the model last estimated. The results of the test may be retrieved using the accessors $test and $pvalue.

Unless the restriction removes all the original regressors, by default the restricted model is estimated and it replaces the original as the "current model" for the purposes of, for example, retrieving the residuals as `$uhat` or doing further tests. This behavior may be suppressed via the `--test-only` option.

By default the *F*-form of the Wald test is recorded; the `--chi-square` option may be used to record the chi-square form instead.

If the restricted model is both estimated and printed, the `--vcv` option has the effect of printing its covariance matrix, otherwise this option is ignored.

Alternatively, if the `--auto` flag is given, sequential elimination is performed: at each step the variable with the highest p-value is omitted, until all remaining variables have a p-value no greater than some cutoff. The default cutoff is 10 percent (two-sided); this can be adjusted by appending "=" and a value between 0 and 1 (with no spaces), as in the fourth example above. If *varlist* is given this process is confined to the listed variables, otherwise all regressors aside from the constant are treated as candidates for omission. Note that the `--auto` and `--test-only` options cannot be combined.

Menu path: Model window, /Tests/Omit variables

open

Argument:	*filename*
Options:	`--quiet` (don't print list of series)
	`--preserve` (preserve variables other than series)
	`--select=`*selection* (read only the specified series, see below)
	`--frompkg=`*pkgname* (see below)
	`--all-cols` (see below)
	`--www` (use a database on the gretl server)
	`--odbc` (use an ODBC database)
	See below for additional specialized options
Examples:	`open data4-1`
	`open voter.dta`
	`open fedbog.bin --www`
	`open dbnomics`

Opens a data file or database—see chapter 4 of the *Gretl User's Guide* for an explanation of this distinction. The effect is somewhat different in the two cases. When a *data file* is opened, its content is read into gretl's workspace, replacing the current dataset (if any). To add data to the current dataset instead of replacing, see append or (for greater flexibility) join. When a *database* is opened this does not immediately load any data; rather, it sets the source for subsequent invocations of the data command, which is used to import selected series. For specifics regarding databases see the section headed "Opening a database" below.

If *filename* is not given as a full path, gretl will search some relevant paths to try to find the file, with workdir as a first choice. If no filename suffix is given (as in the first example above), gretl assumes a native datafile with suffix `.gdt`. Based on the name of the file and various heuristics, gretl will try to detect the format of the data file (native, plain text, CSV, MS Excel, Stata, SPSS, etc.).

If the `--frompkg` option is used, gretl will look for the specified data file in the subdirectory associated with the function package specified by *pkgname*.

If the *filename* argument takes the form of a URI starting with `http://` or `https://`, then gretl will attempt to download the indicated data file before opening it.

By default, opening a new data file clears the current gretl session, which includes deletion of all named variables, including matrices, scalars and strings. If you wish to keep your currently defined variables (other than series, which are necessarily cleared out), use the `--preserve` option.

Spreadsheet files

When opening a data file in a spreadsheet format (Gnumeric, Open Document or MS Excel), you may give up to three additional parameters following the filename. First, you can select a particular worksheet within the file. This is done either by giving its (1-based) number, using the syntax, e.g., `--sheet=2`, or, if you know the name of the sheet, by giving the name in double quotes, as in `--sheet="MacroData"`. The default is to read the first worksheet. You can also specify a column and/or row offset into the worksheet via, e.g.,

```
--coloffset=3 --rowoffset=2
```

which would cause gretl to ignore the first 3 columns and the first 2 rows. The default is an offset of 0 in both dimensions, that is, to start reading at the top-left cell.

Delimited text files

With plain text files, gretl generally expects to find the data columns delimited in some standard manner (generally via comma, tab, space or semicolon). By default gretl looks for observation labels

or dates in the first column if its heading is empty or is a suggestive string such as "year", "date" or "obs". You can prevent gretl from treating the first column specially by giving the `--all-cols` option.

Fixed format text

A "fixed format" text data file is one without column delimiters, but in which the data are laid out according to a known set of specifications such as "variable *k* occupies 8 columns starting at column 24". To read such files, you should append a string `--fixed-cols=`*colspec*, where *colspec* is composed of comma-separated integers. These integers are interpreted as a set of pairs. The first element of each pair denotes a starting column, measured in bytes from the beginning of the line with 1 indicating the first byte; and the second element indicates how many bytes should be read for the given field. So, for example, if you say

```
open fixed.txt --fixed-cols=1,6,20,3
```

then for variable 1 gretl will read 6 bytes starting at column 1; and for variable 2, 3 bytes starting at column 20. Lines that are blank, or that begin with `#`, are ignored, but otherwise the column-reading template is applied, and if anything other than a valid numerical value is found an error is flagged. If the data are read successfully, the variables will be named `v1`, `v2`, etc. It's up to the user to provide meaningful names and/or descriptions using the commands rename and/or setinfo.

String-valued series

By default, when you import a file that contains string-valued series, a text box will open showing you the contents of `string_table.txt`, a file which contains the mapping between strings and their numeric coding. You can suppress this behavior via the `--quiet` option.

Loading selected series

Use of `open` with a data file argument (as opposed to the database case, see below) generally implies loading all series from the specified file. However, in the case of native gretl files (`gdt` and `gdtb`) only, it is possible to specify by name a subset of series to load. This is done via the `--select` option, which requires an accompanying argument in one of three forms: the name of a single series; a list of names, separated by spaces and enclosed in double quotes; or the name of an array of strings. Examples:

```
# single series
open somefile.gdt --select=x1
# more than one series
open somefile.gdt --select="x1 x5 x27"
# alternative method
strings Sel = defarray("x1", "x5", "x27")
open somefile.gdt --select=Sel
```

Opening a database

As mentioned above, the `open` command can be used to open a database file for subsequent reading via the data command. Supported file-types are native gretl databases, RATS 4.0 and PcGive.

Besides reading a file of one of these types on the local machine, three further cases are supported. First, if the `--www` option is given, gretl will try to access a native gretl database of the given name on the gretl server—for instance the Federal Reserve interest rates database `fedbog.bin` in the third example shown above. Second, the command "`open dbnomics`" can be used to set DB.NOMICS as the source for database reads; on this see dbnomics for gretl. Third, if the `--odbc` option is given

gretl will try to access an ODBC database. This option is explained at length in chapter 42 of the *Gretl User's Guide*.

Menu path: /File/Open data

Other access: Drag a data file onto gretl's main window

orthdev

Argument: *varlist*

Applicable with panel data only. A series of forward orthogonal deviations is obtained for each variable in *varlist* and stored in a new variable with the prefix `o_`. Thus `orthdev x y` creates the new variables `o_x` and `o_y`.

The values are stored one step ahead of their true temporal location (that is, `o_x` at observation t holds the deviation that, strictly speaking, belongs at $t - 1$). This is for compatibility with first differences: one loses the first observation in each time series, not the last.

outfile

Variants: `outfile` *filename*
`outfile --buffer=`*strvar*
`outfile --tempfile=`*strvar*

Options: `--append` (append to file, first variant only)
`--quiet` (see below)
`--buffer` (see below)
`--tempfile` (see below)
`--decpoint` (see below)

The `outfile` command starts a block in which any printed output is diverted to a file or buffer (or just discarded, if you wish). Such a block is terminated by the command "`end outfile`", after which output reverts to the default stream.

Diversion to a named file

The first variant shown above sends output to a file named by the *filename* argument. By default a new file is created (or an existing one is overwritten). The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification to the contrary. If you wish to append output to an existing file instead, use the `--append` flag.

A simple example follows, where the output from a particular regression is written to a named file.

```
open data4-10
outfile regress.txt
  ols ENROLL 0 CATHOL INCOME COLLEGE
end outfile
```

Special dummy filenames

Three special values for *filename* are supported, as follows:

- `null`: printed output is suppressed until redirection is ended.
- `stdout`: output is redirected to the "standard output" stream.
- `stderr`: output is redirected to the "standard error" stream.

Diversion to a string buffer

The `--buffer` option is used to store output in a string variable. The required parameter for this option must be the name of an existing string variable, whose content will be over-written. We show below the example given above, revised to write to a string. In this case printing `model_out` will display the redirected output.

```
open data4-10
string model_out = ""
outfile --buffer=model_out
  ols ENROLL 0 CATHOL INCOME COLLEGE
end outfile
print model_out
```

Diversion to a temporary file

The `--tempfile` option is used to direct output to a temporary file, with an automatically constructed name that is guaranteed to be unique, in the user's "dot" directory. As in the redirection to buffer case, the option parameter should be the name of a string variable: in this case its content is over-written with the name of the temporary file. Please note: files written to the dot directory are cleaned up on exit from the program, so don't use this form if you want the output to be preserved after your gretl session.

We repeat the simple example from above, with a couple of extra lines to illustrate the points that *strvar* tells you where the output went, and you can retrieve it using the readfile function.

```
open data4-10
string mytemp
outfile --tempfile=mytemp
  ols ENROLL 0 CATHOL INCOME COLLEGE
end outfile
printf "Output went to %s\n", mytemp
printf "The output was:\n%s\n", readfile(mytemp)
# clean up if the file is no longer needed
remove(mytemp)
```

In some cases you may wish to exercise some control over the name of the temporary file. You can do this by supplying a string variable which contains six consecutive Xs, as in

```
string mytemp = "tmpXXXXXX.csv"
outfile --tempfile=mytemp
...
```

In this case XXXXXX will be replaced by random characters that ensure uniqueness of the filename, but the "`.csv`" suffix will be preserved. As in the simpler case above, the file is automatically written into the user's "dot" directory and the content of the string variable passed via the option flag is modified to hold the full path to the temporary file.

Quietness

The effect of the `--quiet` option is to turn off the echoing of commands and the printing of auxiliary messages while output is redirected. It is equivalent to doing

```
set echo off
set messages off
```

except that when redirection is ended the original values of the `echo` and `messages` variables are restored. This option is available in all cases.

Decimal character

The effect of the `--decpoint` option is ensure that the decimal point character (as opposed to comma) is in force while output is redirected. When the `outfile` block ends the decimal character reverts to whatever was in place before it. This option is especially useful if the text file to be created is meant as an input for some other program that requires digits to follow the English convention, as would be the case, for example, of a gnuplot or R script.

Levels of redirection

In general only one file can be opened in this way at any given time, so calls to this command cannot be nested. However, use of this command is permitted inside user-defined functions (provided the output file is also closed from inside the same function) such that output can be temporarily diverted and then given back to an original output file, in case `outfile` is currently in use by the caller. For example, the code

```
function void f (string s)
    outfile inner.txt
      print s
    end outfile
end function

outfile outer.txt --quiet
  print "Outside"
  f("Inside")
  print "Outside again"
end outfile
```

will produce a file called “outer.txt” containing the two lines

```
Outside
Outside again
```

and a file called “inner.txt” containing the line

```
Inside
```

panel

Arguments:	*depvar indepvars*
Options:	`--vcv` (print covariance matrix)
	`--fixed-effects` (estimate with group fixed effects)
	`--random-effects` (random effects or GLS model)
	`--nerlove` (use the Nerlove transformation)
	`--pooled` (estimate via pooled OLS)
	`--between` (estimate the between-groups model)
	`--robust` (robust standard errors; see below)
	`--cluster`=*cvar* (clustered standard errors; see below)
	`--time-dummies` (include time dummy variables)
	`--unit-weights` (weighted least squares)
	`--iterate` (iterative estimation)
	`--matrix-diff` (compute Hausman test via matrix difference)
	`--unbalanced`=*method* (random effects only, see below)
	`--quiet` (less verbose output)
	`--verbose` (more verbose output)
Examples:	`penngrow.inp`, `panel-robust.inp`

Estimates a panel model. By default the fixed effects estimator is used; this is implemented by subtracting the group or unit means from the original data.

If the `--random-effects` flag is given, random effects estimates are computed, by default using the method of Swamy and Arora (1972). In this case (only) the option `--matrix-diff` forces use of the matrix-difference method (as opposed to the regression method) for carrying out the Hausman test for the consistency of the random effects estimator. Also specific to the random effects estimator is the `--nerlove` flag, which selects the method of Nerlove (1971) as opposed to Swamy and Arora.

Alternatively, if the `--unit-weights` flag is given, the model is estimated via weighted least squares, with the weights based on the residual variance for the respective cross-sectional units in the sample. In this case (only) the `--iterate` flag may be added to produce iterative estimates: if the iteration converges, the resulting estimates are Maximum Likelihood.

As a further alternative, if the `--between` flag is given, the between-groups model is estimated (that is, an OLS regression using the group means).

The default means of calculating robust standard errors in panel-data models is the HAC estimator of Arellano (2003) (clustered by panel unit). Alternatives are "Panel Corrected Standard Errors" (Beck and Katz (1995)) and "Spatial Correlation Consistent" standard errors (Driscoll and Kraay (1998)). These can be selected via the command `set panel_robust` with arguments `pcse` and `scc`, respectively. Other alternatives to these three options are available via the `--cluster` option; please see chapter 22 of the *Gretl User's Guide* for details. When robust standard errors are specified the joint F test on the fixed effects is performed using the robust method of Welch (1951).

The `--unbalanced` option is available only for random effects models: it can be used to choose an ANOVA method for use with an unbalanced panel. By default gretl uses the Swamy–Arora method as for balanced panels, except that the harmonic mean of the individual time-series lengths is used in place of a common T. Under this option you can specify either `bc`, to use the method of Baltagi and Chang (1994), or `stata`, to emulate the `sa` option to the `xtreg` command in Stata.

For more details on panel estimation, please see chapter 23 of the *Gretl User's Guide*.

Menu path: /Model/Panel

panplot

Argument:	*plotvar*
Options:	`--means` (time series, group means)
	`--overlay` (plot per group, overlaid, N <= 130)
	`--sequence` (plot per group, in sequence, N <= 130)
	`--grid` (plot per group, in grid, N <= 16)
	`--stack` (plot per group, stacked, N <= 6)
	`--boxplots` (boxplot per group, in sequence, N <= 150)
	`--boxplot` (single boxplot, all groups)
	`--output=`*filename* (send output to specified file)
Examples:	`panplot x --overlay`
	`panplot x --means --output=display`

Graphing command specific to panel data: the series *plotvar* is plotted in a mode specified by one or other of the options.

Apart from the `--means` and `--boxplot` options the plot explicitly represents variation in both the time-series and cross-sectional dimensions. Such plots are limited in respect of the number of groups (also known as individuals or units) in the current sample range of the panel. For example, the `--overlay` option, which shows a time series for each group in a single plot, is available only when the number of groups, N, is 130 or less. (Otherwise the graphic becomes too dense to be informative.) If a panel is too large to permit the desired plot specification one can select a reduced range of groups or units temporarily, as in

```
smpl 1 100 --unit
panplot x --overlay
smpl full
```

The `--output=`*filename* option can be used to control the form and destination of the output; see the gnuplot command for details.

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

panspec

Options:	`--nerlove` (use Nerlove method for random effects)
	`--matrix_diff` (use matrix-difference method for Hausman test)
	`--quiet` (Suppress printed output)

This command is available only after estimating a panel-data model via OLS. It tests the simple pooled specification against the most common alternatives, fixed effects and random effects.

The fixed effects specification allows the intercept of the regression to vary across the cross-sectional units. A Wald F-test is reported for the null hypotheses that the intercepts do not differ. The random effects specification 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 pooled OLS is adequate against the random effects alternative.

Pooled OLS may be rejected against both of the alternatives. Provided the unit- or group-specific error is uncorrelated with the independent variables, the random effects estimator is more efficient than fixed effects; otherwise the random effects estimator is inconsistent and fixed effects are 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 estimator is preferable). A low p-value for this test counts against random effects and in favor of fixed effects.

The first two options for this command pertain to random effects estimation. By default the method of Swamy and Arora is used, and the Hausman test statistic is calculated using the regression method. The options enable the use of Nerlove's alternative variance estimator, and/or the matrix-difference approach to the Hausman statistic.

On successful completion the accessors $test and $pvalue retrieve 3-vectors holding test statistics and p-values for the three tests noted above: poolability (Wald), poolability (Breusch–Pagan), and Hausman. If you just want the results in this form you can give the `--quiet` option to skip printed output.

Note that after estimating the random effects specification via the `panel` command, the Hausman test is automatically carried out and the results can be retrieved via the $hausman accessor.

Menu path: Model window, /Tests/Panel specification

pca

Argument:	*varlist*
Options:	`--covariance` (use the covariance matrix)
	`--save[=`*n*`]` (save major components)
	`--save-all` (save all components)
	`--quiet` (don't print results)

Principal Components Analysis. Unless the `--quiet` option is given, prints the eigenvalues of the correlation matrix (or the covariance matrix if the `--covariance` option is given) 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 you give the `--save-all` option then all components are saved to the dataset as series, with names `PC1`, `PC2` and so on. These artificial variables are formed as the sum of (component loading) times (standardized X_i), where X_i denotes the ith variable in *varlist*.

If you give the `--save` option without a parameter value, components with eigenvalues greater than the mean (which means greater than 1.0 if the analysis is based on the correlation matrix) are saved to the dataset as described above. If you provide a value for n with this option then the most important n components are saved.

See also the princomp function.

Menu path: /View/Principal components

pergm

Arguments:	*series* [*bandwidth*]
Options:	`--bartlett` (use Bartlett lag window)
	`--log` (use log scale)
	`--radians` (show frequency in radians)
	`--degrees` (show frequency in degrees)
	`--plot=`*mode-or-filename* (see below)
	`--silent` (suppress printed output)

Computes and displays the spectrum of the specified series. By default the sample periodogram is given, but optionally a Bartlett lag window is used in estimating the spectrum (see, for example, Greene's *Econometric Analysis* for a discussion of this). The default width of the Bartlett window is twice the square root of the sample size but this can be set manually using the *bandwidth* parameter, up to a maximum of half the sample size.

If the `--log` option is given the spectrum is represented on a logarithmic scale.

The (mutually exclusive) options `--radians` and `--degrees` influence the appearance of the frequency axis when the periodogram is graphed. By default the frequency is scaled by the number of periods in the sample, but these options cause the axis to be labeled from 0 to π radians or from 0 to 180°, respectively.

By default, if gretl is not in batch mode a plot of the periodogram is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Menu path: /Variable/Periodogram

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

pkg

Arguments:	*action pkgname*
Options:	`--local` (install from local file)
	`--quiet` (see below)
	`--verbose` (see below)
	`--staging` (see below)
Examples:	`pkg install armax`
	`pkg install /path/to/myfile.gfn --local`
	`pkg query ghosts`
	`pkg run-sample ghosts`
	`pkg unload armax`

This command provides a means of installing, unloading, querying or deleting gretl function packages. The *action* argument must be one of `install`, `query`, `run-sample`, `unload`, `remove` or `index`. An extension to support data file packages is described below.

`install`: In the most basic form, with no option flag and the *pkgname* argument given as the "plain" name of a gretl function package (as in the first example above), the effect is to download the specified package from the gretl server (unless *pkgname* starts with `http://`) and install it on the local machine. In this case it is not necessary to supply a filename extension. If the `--local` option is given, however, *pkgname* should be the path to an uninstalled package file on the local machine, with the correct extension (`.gfn` or `.zip`). In this case the effect is to copy the file into place (`gfn`), or unzip it into place (`zip`), "into place" meaning where the include command will find it.

`query`: The default effect is to print basic information about the specified package (author, version, etc.). If the package includes extra resources (data files and/or additional scripts) a listing of these files is included. If the `--quiet` option is appended nothing is printed; the package information is instead stored in the form of a gretl bundle, which can be accessed via $result. If no information can be found this bundle will be empty.

`run-sample`: Provides a command-line means of running the sample script included in the specified package.

`unload`: *pkgname* should be given in plain form, without path or suffix as in the last example above. The effect is to unload the package in question from gretl's memory, if it is currently loaded, and also to remove it from the GUI menu to which it is attached, if any.

`remove`: performs the actions noted for `unload` and in addition deletes the file(s) associated with the package from disk.

`index`: is a special case in which *pkgname* must be replaced by the keyword "`addons`": the effect is to update the index of the standard packages known as addons. Such updating is performed automatically from time to time but in some cases a manual update may be useful. In this case the

`--verbose` flag produces a printout of where gretl has searched and what it has found. To be clear, here's the way to get full indexing output:

```
pkg index addons --verbose
```

Data file packages

Besides its usage with function packages, the `pkg install` command can also be used with data file packages of the `tar.gz` type, as listed at https://gretl.sourceforge.net/gretl_data.html. For example, to install the Verbeek data files one can do

```
pkg install verbeek.tar.gz
```

Note that `install` is the only action supported operation for such files.

Staging

The `--staging` option is a convenience item for developers and is available only in conjunction with the `install` action as applied to a function package. Its effect is that the package in question is downloaded from the `staging` area at sourceforge rather than the public area. Packages in staging are not yet approved for general use, so ignore this option unless you know what you're doing.

Menu path: /File/Function packages/On server

plot

Argument:	`[` *data* `]`
Options:	`--output=`*filename* (send output to specified file)
	`--outbuf=`*stringname* (send output to specified string)
Example:	`nile.inp`

The `plot` block provides an alternative to the gnuplot command which may be more convenient when you are producing an elaborate plot (with several options and/or gnuplot commands to be inserted into the plot file). In addition to the following explanation, please also refer to chapter 6 of the *Gretl User's Guide* for some further examples.

A `plot` block starts with the command-word `plot`. This is commonly followed by a *data* argument, which specifies data to be plotted: this should be the name of a list, a matrix, or a single series. If no input data are specified the block must contain at least one directive to plot a formula instead; such directives may be given via `literal` or `printf` lines (see below).

If a list or matrix is given, the last element (list) or column (matrix) is assumed to be the x-axis variable and the other(s) the y-axis variable(s), unless the `--time-series` option is given in which case all the specified data go on the y axis. The option of supplying a single series name is restricted to time-series data, in which case it is assumed that a time-series plot is wanted; otherwise an error is flagged.

The starting line may be prefixed with the "*savename* `<-`" apparatus to save a plot as an icon in the GUI program. The block ends with `end plot`.

Inside the block you have zero or more lines of these types, identified by an initial keyword:

- `option`: specify a single option.
- `options`: specify multiple options on a single line, separated by spaces.
- `literal`: a command to be passed to gnuplot literally.

- `printf`: a printf statement whose result will be passed to gnuplot literally.

Note that besides `--output` and `--outbuf`, which should be appended to the line that ends the block, all the options supported by the gnuplot command are also supported by `plot`, but should be given within the block using the syntax described above. In this context it is not necessary to supply the customary double-dash before the option specifier. For details on the effects of the various options see gnuplot.

The intended use of the `plot` block is best illustrated by example:

```
string title = "My title"
string xname = "My x-variable"
plot plotmat
    options with-lines fit=none
    literal set linetype 3 lc rgb "#0000ff"
    literal set nokey
    printf "set title '%s'", title
    printf "set xlabel '%s'", xname
end plot --output=display
```

This example assumes that `plotmat` is the name of a matrix with at least 2 columns (or a list with at least two members). Note that it is considered good practice to place the `--output` option (only) on the last line of the block; other options should be placed within the block.

Plotting without data

The following example shows a case of specifying a plot without a data source.

```
plot
    literal set title 'CRRA utility'
    literal set xlabel 'c'
    literal set ylabel 'u(c)'
    literal set xrange[1:3]
    literal set key top left
    literal crra(x,s) = (x**(1-s) - 1)/(1-s)
    printf "plot crra(x, 0) t 'sigma=0', \\"
    printf " log(x) t 'sigma=1', \\"
    printf " crra(x,3) t 'sigma=3"
end plot --output=display
```

poisson

Arguments:	*depvar indepvars* `[ ; ` *offset* `]`
Options:	`--robust` (robust standard errors)
	`--cluster=`*clustvar* (see logit for explanation)
	`--vcv` (print covariance matrix)
	`--verbose` (print details of iterations)
	`--quiet` (don't print results)
Examples:	`poisson y 0 x1 x2`
	`poisson y 0 x1 x2 ; S`
	See also `camtriv.inp`, `greene19_3.inp`

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.

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 also negbin.

Menu path: /Model/Limited dependent variable/Count data

print

Variants:	`print` *varlist*
	`print`
	`print` *object-names*
	`print` *string-literal*
Options:	`--byobs` (by observations)
	`--no-dates` (use simple observation numbers)
	`--range=`*start:stop* (see below)
	`--midas` (see below)
	`--tree` (specific to bundles; see below)
Examples:	`print x1 x2 --byobs`
	`print my_matrix`
	`print "This is a string"`
	`print my_array --range=3:6`
	`print hflist --midas`

Please note that `print` is a rather "basic" command (primarily intended for printing the values of series); see printf and eval for more advanced, and less restrictive, alternatives.

In the first variant shown above (also see the first example), *varlist* should be a list of series (either a named list or a list specified via the names or ID numbers of series, separated by spaces). In that case this command prints the values of the listed series. By default the data are printed "by variable", but if the `--byobs` flag is added they are printed by observation. When printing 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. See the final paragraph of this entry for the effect of the `--midas` option (which applies only to a named list of series).

If no argument is given (the second variant shown above) then the action is similar to the first case except that *all* series in the current dataset are printed. The supported options are as decribed above.

The third variant (with the *object-names* argument; see the second example) expects a space-separated list of names of primary gretl objects other than series (scalars, matrices, strings, bundles, arrays). The value(s) of these objects are displayed. In the case of bundles, their members are sorted by type and alphabetically.

In the fourth form (third example), *string-literal* should be a string enclosed in double-quotes (and there should be nothing else following on the command line). The string in question is printed, followed by a newline character.

The `--range` option can be used to control the amount of information printed. The *start* and *stop* (integer) values refer to observations for series and lists, rows for matrices, elements for arrays, and lines of text for strings. In all cases the minimum *start* value is 1 and the maximum *stop* value is the "row-wise size" of the object in question. Negative values for these indices are taken to indicate a count back from the end. The indices may be given in numeric form or as the names of predefined scalar variables. If *start* is omitted that is taken as an implicit 1 and if *stop* is omitted that means go all the way to the end. Note that with series and lists the indices are relative to the current sample range.

The `--tree` option is specific to the printing of a gretl bundle: the effect is that if the specified bundle contains further bundles, or arrays of bundles, their contents are listed. Otherwise only the top-level members of the bundle are listed.

The `--midas` option is specific to the printing of a list of series, and moreover it is specific to datasets that contain one or more high-frequency series, each represented by a MIDAS list. If one such list is given as argument and this option is appended, the series is printed by observation at its "native" frequency.

Menu path: /Data/Display values

printf

Arguments: *format* , *args*

Prints scalar values, series, matrices, or strings under the control of a format string (providing a subset of the `printf` function in the C programming language). Recognized numeric formats are `%e`, `%E`, `%f`, `%g`, `%G`, `%d` and `%x`, 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. Note, however, that in gretl the format `%g` is a good default choice for all numerical values; you don't need to get too complicated. The format `%s` should be used for strings.

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, (b) expressions that yield some sort of printable result, 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
scalar b = $coeff[2]
scalar 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)
```

If a matrix argument is given in association with a numeric format, the entire matrix is printed using the specified format for each element. The same applies to series, except that the range of values printed is governed by the current sample setting.

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

As in C, numerical values that form part of the format (width and or precision) may be given directly as numbers, as in `%10.4f`, or they may be given as variables. In the latter case, one puts asterisks into the format string and supplies corresponding arguments in order. For example,

```
scalar width = 12
scalar precision = 6
printf "x = %*.*f\n", width, precision, x
```

probit

Arguments:	*depvar indepvars*
Options:	`--robust` (robust standard errors)
	`--cluster`=*clustvar* (see logit for explanation)
	`--vcv` (print covariance matrix)
	`--verbose` (print details of iterations)
	`--quiet` (don't print results)
	`--p-values` (show p-values instead of slopes)
	`--estrella` (select pseudo-R-squared variant)
	`--random-effects` (estimates a random effects panel probit model)
	`--quadpoints`=*k* (number of quadrature points for RE estimation)
Examples:	`ooballot.inp`, `oprobit.inp`, `reprobit.inp`

If the dependent variable is a binary variable (all values are 0 or 1) maximum likelihood estimates of the coefficients on *indepvars* are obtained via the Newton–Raphson method. As the model is nonlinear the slopes depend on the values of the independent variables. By default the slopes with respect to each of the independent variables are calculated (at the means of those variables) and these slopes replace the usual p-values in the regression output. This behavior can be suppressed by giving the `--p-values` option. 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 chapter 10 of Davidson and MacKinnon for details.

By default the pseudo-R-squared statistic suggested by McFadden (1974) is shown, but in the binary case if the `--estrella` option is given, the variant recommended by Estrella (1998) is shown instead. This variant arguably mimics more closely the properties of the regular R^2 in the context of least-squares estimation.

If the dependent variable is not binary but is discrete, then Ordered Probit estimates are obtained. (If the variable selected as dependent is not discrete, an error is flagged.)

Probit for panel data

With the `--random-effects` option, the error term is assumed to be composed of two normally distributed components: one time-invariant term that is specific to the cross-sectional unit or "individual" (and is known as the individual effect); and one term that is specific to the particular observation.

Evaluation of the likelihood for this model involves the use of Gauss-Hermite quadrature for approximating the value of expectations of functions of normal variates. The number of quadrature points used can be chosen through the `--quadpoints` option (the default is 32). Using more points will increase the accuracy of the results, but at the cost of longer compute time; with many quadrature points and a large dataset estimation may be quite time consuming.

Besides the usual parameter estimates (and associated statistics) relating to the included regressors, certain additional information is presented on estimation of this sort of model:

- `lnsigma2`: the maximum likelihood estimate of the log of the variance of the individual effect;
- `sigma_u`: the estimated standard deviation of the individual effect; and
- `rho`: the estimated share of the individual effect in the composite error variance (also known as the intra-class correlation).

The Likelihood Ratio test of the null hypothesis that `rho` equals zero provides a means of assessing whether the random effects specification is needed. If the null is not rejected that suggests that a simple pooled probit specification is adequate.

Menu path: /Model/Limited dependent variable/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 shape scale x
pvalue B bprob 10 6
pvalue P lambda x
pvalue W shape scale x
```
See also `mrw.inp`, `restrict.inp`

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, `B` for binomial, `P` for Poisson, `exp` for Exponential, `W` for Weibull).

Depending on the distribution, the following information must be given, before the *xval*: for the *t* and chi-square distributions, the degrees of freedom; for *F*, the numerator and denominator degrees of freedom; for gamma, the shape and scale parameters; for the binomial distribution, the "success" probability and the number of trials; for the Poisson distribution, the parameter λ (which is both the mean and the variance); for the Exponential, a scale parameter; and for the Weibull, shape and scale parameters. As shown in the examples above, the numerical parameters may be given in numeric form or as the names of variables.

The parameters for the gamma distribution are sometimes given as mean and variance rather than shape and scale. The mean is the product of the shape and the scale; the variance is the product of the shape and the square of the scale. So the scale may be found as the variance divided by the mean, and the shape as the mean divided by the scale.

Menu path: /Tools/P-value finder

qlrtest

Options: `--limit-to=`*list* (limit test to subset of regressors)
`--plot=`*mode-or-filename* (see below)
`--quiet` (suppress printed output)

For a model estimated on time-series data via OLS, performs the Quandt likelihood ratio (QLR) test for a structural break at an unknown point in time, with 15 percent trimming at the beginning and end of the sample period.

For each potential break point within the central 70 percent of the observations, a Chow test is performed. See chow for details; as with the regular Chow test, this is a robust Wald test if the original model was estimated with the `--robust` option, an F-test otherwise. The QLR statistic is then the maximum of the individual test statistics.

An asymptotic p-value is obtained using the method of Hansen (1997).

Besides the standard hypothesis test accessors $test and $pvalue, $qlrbreak can be used to retrieve the index of the observation at which the test statistic is maximized.

The `--limit-to` option can be used to limit the set of interactions with the split dummy variable in the Chow tests to a subset of the original regressors. The parameter for this option must be a named list, all of whose members are among the original regressors. The list should not include the constant.

When this command is run interactively (only), a plot of the Chow test statistic is displayed by default. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to display a plot even when not in interactive mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Menu path: Model window, /Tests/QLR test

qqplot

Variants: `qqplot` *y*
`qqplot` *y x*
Options: `--z-scores` (see below)
`--raw` (see below)
`--output=`*filename* (send plot to specified file)

Given just one series argument, displays a plot of the empirical quantiles of the selected series (given by name or ID number) against the quantiles of the normal distribution. The series must include at least 20 valid observations in the current sample range. By default the empirical quantiles are plotted against quantiles of the normal distribution having the same mean and variance as the sample data, but two alternatives are available: if the `--z-scores` option is given the data are standardized, while if the `--raw` option is given the "raw" empirical quantiles are plotted against the quantiles of the standard normal distribution.

The option `--output` has the effect of sending the output to the specified file; use "display" to force output to the screen. See the gnuplot command for more detail on this option.

Given two series arguments, *y* and *x*, displays a plot of the empirical quantiles of *y* against those of *x*. The data values are not standardized.

Menu path: /Variable/Normal Q-Q plot

Menu path: /View/Graph specified vars/Q-Q plot

quantreg

Arguments:	*tau depvar indepvars*
Options:	`--robust` (robust standard errors)
	`--intervals[=`*level*`]` (compute confidence intervals)
	`--vcv` (print covariance matrix)
	`--quiet` (suppress printing of results)
Examples:	`quantreg 0.25 y 0 xlist`
	`quantreg 0.5 y 0 xlist --intervals`
	`quantreg 0.5 y 0 xlist --intervals=.95`
	`quantreg tauvec y 0 xlist --robust`
	See also `mrw_qr.inp`

Quantile regression. The first argument, *tau*, is the conditional quantile for which estimates are wanted. It may be given either as a numerical value or as the name of a pre-defined scalar variable; the value must be in the range 0.01 to 0.99. (Alternatively, a vector of values may be given for *tau*; see below for details.) The second and subsequent arguments compose a regression list on the same pattern as ols.

Without the `--intervals` option, standard errors are printed for the quantile estimates. By default, these are computed according to the asymptotic formula given by Koenker and Bassett (1978), but if the `--robust` option is given, standard errors that are robust with respect to heteroskedasticity are calculated using the method of Koenker and Zhao (1994).

When the `--intervals` option is chosen, confidence intervals are given for the parameter estimates instead of standard errors. These intervals are computed using the rank inversion method, and in general they are asymmetrical about the point estimates. The specifics of the calculation are inflected by the `--robust` option: without this, the intervals are computed on the assumption of IID errors (Koenker, 1994); with it, they use the robust estimator developed by Koenker and Machado (1999).

By default, 90 percent confidence intervals are produced. You can change this by appending a confidence level (expressed as a decimal fraction) to the intervals option, as in `--intervals=0.95`.

Vector-valued *tau*: instead of supplying a scalar, you may give the name of a pre-defined matrix. In this case estimates are computed for all the given *tau* values and the results are printed in a special format, showing the sequence of quantile estimates for each regressor in turn.

Menu path: /Model/Robust estimation/Quantile regression

quit

Exits from gretl's current modality.

- When called from a script, execution of the script is terminated. If the context is gretlcli in batch mode, gretlcli itself exits, otherwise the program reverts to interactive mode.
- When called from the GUI console, the console window is closed.
- When called from gretlcli in interactive mode the program exits.

Note that this command cannot be called within functions or loops.

In no case does the `quit` command cause the gretl GUI program to exit. That is done via the `Quit` item under the `File` menu, or `Ctrl+Q`, or by clicking the close control on the title-bar of the main gretl window.

rename

Arguments:	*series newname*
Options:	`--quiet` (suppress printed output)
	`--case` (change case of all series names, see below)
Examples:	`rename x2 income`
	`rename --case=lower`

Without the `--case` option this command changes the name of *series* (identified by name or ID number) to *newname*. The new name must be of 31 characters maximum, must start with a letter, and must be composed of only letters, digits, and the underscore character. In addition, it must not be the name of an existing object of any kind.

The `--case` option allows changing the case of *all* series names in the currently open dataset. When using this option, no *series* or *newname* should be provided. The following case types are supported:

- *lower*: Convert all series names to lowercase.
- *upper*: Convert all series names to uppercase.
- *camel*: Convert all series names to camel case, meaning that underscores are deleted and the following character (if any) is capitalized. For example, `some_thing` becomes `someThing`.
- *snake*: Convert all series names to snake case, meaning that any capital letters (other than the first in the name) are converted to lowercase, with an underscore prepended. For example, `someThing` becomes `some_thing`.

Menu path: /Variable/Edit attributes

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

reset

Options:	`--quiet` (don't print the auxiliary regression)
	`--silent` (don't print anything)
	`--squares-only` (compute the test using only the squares)
	`--cubes-only` (compute the test using only the cubes)
	`--robust` (use robust standard errors in auxiliary regression)

Must follow the estimation of a model via OLS. Carries out Ramsey's RESET test for model specification (nonlinearity) by adding the squares and/or the cubes of the fitted values to the regression and calculating the F statistic for the null hypothesis that the coefficients on the added terms are zero. For numerical reasons, the squares and cubes are rescaled using the standard deviation of the fitted values.

Both the squares and the cubes are added unless one of the options `--squares-only` or `--cubes-only` is given.

The `--silent` option may be used if one plans to make use of the $test and/or $pvalue accessors to grab the results of the test.

The `--robust` option is implicit if the regression to be tested employed robust standard errors.

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

restrict

Options: `--quiet` (don't print restricted estimates)
`--silent` (don't print anything)
`--wald` (system estimators only – see below)
`--bootstrap` (bootstrap the test if possible)
`--full` (OLS and VECMs only, see below)
Examples: `hamilton.inp`, `restrict.inp`

Imposes a set of (usually linear) restrictions on either (a) the model last estimated or (b) a system of equations previously defined and named. In all 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 always implicitly to be applied to the last model, and they are evaluated as soon as the `restrict` block is closed.

In the case of a system of equations (defined via the system command), the initial "restrict" may be followed by the name of a previously defined system of equations. If this is omitted and the last model was a system then the restrictions are applied to the last model. By default the restrictions are evaluated when the system is next estimated, using the estimate command. But if the `--wald` option is given the restriction is tested right away, via a Wald chi-square test on the covariance matrix. Note that this option will produce an error if a system has been defined but not yet estimated.

Depending on the context, the restrictions to be tested may be expressed in various ways. The simplest form is as follows: each restriction is given as an equation, with a linear combination of parameters on the left and a scalar value to the right of the equals sign (either a numerical constant or the name of a scalar variable).

In the single-equation case, parameters may be referenced in the form `b[`*i*`]`, where *i* represents the position in the list of regressors (starting at 1), or `b[`*varname*`]`, where *varname* is the name of the regressor in question. 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 and the second number indicates position in the list of regressors. For example `b[2,1]` denotes the first parameter in the second equation, and `b[3,2]` the second parameter in the third equation. The `b` terms in the equation representing a restriction may be prefixed with a numeric multiplier, for example `3.5*b[4]`.

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

```
restrict
 b[1] = 0
 b[2] - b[3] = 0
 b[4] + 2*b[5] = 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 by default evaluated via a Wald test, using the covariance matrix of the model in question. If the original model was estimated via OLS then the restricted coefficient estimates are printed; to suppress this, append the `--quiet` option flag to the

initial `restrict` command. As an alternative to the Wald test, for models estimated via OLS or WLS only, you can give the `--bootstrap` option to perform a bootstrapped test of the restriction.

In the system case, 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.

Linear restrictions: alternative syntax

There are two alternatives to the method of expressing restrictions described above. First, a set of g restrictions on a k-vector of parameters, β, may be written compactly as $R\beta - q = 0$, where R is an $g \times k$ matrix and q is a g-vector. You can specify a restriction by giving the names of pre-defined, conformable matrices to be used as R and q, as in

```
restrict
  R = Rmat
  q = qvec
end restrict
```

Second, in a variant that may be useful when `restrict` is used within a function, you can construct the set of restriction statements in the form of an array of strings. You then use the `inject` keyword with the name of the array. Here's a simple example:

```
strings RS = array(2)
RS[1] = "b[1,2] = 0"
RS[2] = "b[2,1] = 0"
restrict
  inject RS
end restrict
```

In actual usage of this method one would likely use sprintf to construct the strings, based on input to a function.

Nonlinear restrictions

If you wish to test a nonlinear restriction (this is currently available for single-equation models only) you should give the restriction as the name of a function, preceded by "`rfunc =` ", as in

```
restrict
  rfunc = myfunction
end restrict
```

The constraint function should take a single `const matrix` argument; this will be automatically filled out with the parameter vector. And it should return a vector which is zero under the null hypothesis, non-zero otherwise. The length of the vector is the number of restrictions. This function is used as a "callback" by gretl's numerical Jacobian routine, which calculates a Wald test statistic via the delta method.

Here is a simple example of a function suitable for testing one nonlinear restriction, namely that two pairs of parameter values have a common ratio.

```
function matrix restr (const matrix b)
  matrix v = b[1]/b[2] - b[4]/b[5]
  return v
end function
```

On successful completion of the `restrict` command the accessors $test and $pvalue give the test statistic and its p-value.

When testing restrictions on a single-equation model estimated via OLS, or on a VECM, the `--full` option can be used to set the restricted estimates as the "last model" for the purposes of further testing or the use of accessors such as `$coeff` and `$vcv`. Note that some special considerations apply in the case of testing restrictions on Vector Error Correction Models. Please see chapter 33 of the *Gretl User's Guide* for details.

Menu path: Model window, /Tests/Linear restrictions

rmplot

Argument: *series*
Options: `--trim` (see below)
`--quiet` (suppress printed output)
`--output=`*filename* (see below)

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. The t-statistic for the null, and the corresponding p-value, are recorded and may be retrieved using the accessors $test and $pvalue respectively.

If the `--trim` option is given, the minimum and maximum values in each sub-sample are discarded before calculating the mean and range. This makes it less likely that outliers will distort the analysis.

If the `--quiet` option is given, no graph is shown and no output is printed; only the t-statistic and p-value are recorded. Otherwise the form of the plot can be controlled via the `--output` option; this works as described in connection with the gnuplot command.

Menu path: /Variable/Range-mean graph

run

Argument: *filename*

Executes the commands in *filename* then returns 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: *series*
Options: `--difference` (use first difference of variable)
`--equal` (positive and negative values are equiprobable)

Carries out the nonparametric "runs" test for randomness of the specified *series*, where runs are defined as sequences of consecutive positive or negative values. 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:

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

If the `--difference` option is given, the variable is differenced prior to the analysis, hence the runs are interpreted as sequences of consecutive increases or decreases in the value of the variable.

If the `--equal` option is given, the null hypothesis incorporates the assumption that positive and negative values are equiprobable, otherwise the test statistic is invariant with respect to the "fairness" of the process generating the sequence, and the test focuses on independence alone.

Menu path: /Tools/Nonparametric tests

scatters

Arguments: *yvar* `;` *xvars* or *yvars ;* *xvar*
Options: `--with-lines` (create line graphs)
`--matrix=`*name* (plot columns of named matrix)
`--output=`*filename* (send output to specified file)
Examples:
```
scatters 1 ; 2 3 4 5
scatters 1 2 3 4 5 6 ; 7
scatters y1 y2 y3 ; x --with-lines
```

Generates pairwise graphs of *yvar* against all the variables in *xvars*, or of all the variables in *yvars* 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 16; any extra variable in the list will be ignored.

By default the data are shown as points, but if you give the `--with-lines` flag they will be line graphs.

For details on usage of the `--output` option, please see the gnuplot command.

If a named matrix is specified as the data source the x and y lists should be given as 1-based column numbers. Alternatively, if no lists are given, all the columns are plotted against time or an index variable.

See also tsplots for a simple means of producing multiple time-series plots, and gridplot for a more flexible way of combining plots in a grid.

Menu path: /View/Multiple graphs/X-Y scatters

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: /Add/Seasonal differences of selected variables

set

Variants: `set` *variable value*
`set --to-file=`*filename*
`set --from-file=`*filename*
`set stopwatch`
`set`

Examples: `set svd on`
`set csv_delim tab`
`set horizon 10`
`set --to-file=mysettings.inp`

The most common use of this command is the first variant shown above, where it is used to set the value of a selected program parameter. This is discussed in detail below. The other uses are: with `--to-file`, to write a script file containing all the current parameter settings; with `--from-file` to read a script file containing parameter settings and apply them to the current session; with `stopwatch` to zero the gretl "stopwatch" which can be used to measure CPU time (see the entry for the $stopwatch accessor); or, if the word `set` is given alone, to print the current settings.

Values set via this comand remain in force for the duration of the gretl session unless they are changed by a further call to `set`. The parameters that can be set in this way are enumerated below. Note that the settings of `hc_version`, `hac_lag` and `hac_kernel` are used when the `--robust` option is given to an estimation command.

The available settings are grouped under the following categories: program interaction and behavior, numerical methods, random number generation, robust estimation, filtering, time series estimation, and interaction with GNU R.

Program interaction and behavior

These settings are used for controlling various aspects of the way gretl interacts with the user.

- `workdir`: *path*. Sets the default directory for writing and reading files, whenever full paths are not specified.
- `use_cwd`: `on` or `off` (the default). Governs the setting of `workdir` at start-up: if it's `on`, the working directory is inherited from the shell, otherwise it is set to whatever was selected in the previous gretl session.
- `echo`: `off` or `on` (the default). Suppress or resume the echoing of commands in gretl's output.
- `messages`: `off` or `on` (the default). Suppress or resume the printing of non-error messages associated with various commands, for example when a new variable is generated or when the sample range is changed.
- `verbose`: `off`, `on` (the default) or `comments`. Acts as a "master switch" for `echo` and `messages` (see above), turning them both off or on simultaneously. The `comments` argument turns off echo and messages but preserves printing of comments in a script.
- `warnings`: `off` or `on` (the default). Suppress or resume the printing of warning messages when numerical problems arise, for example a computation produces non-finite values or the convergence of an optimizer is questionable.
- `csv_delim`: either `comma` (the default), `space`, `tab` or `semicolon`. Sets the column delimiter used when saving data to file in CSV format.

- csv_write_na: the string used to represent missing values when writing data to file in CSV format. Maximum 7 characters; the default is NA.
- csv_read_na: the string taken to represent missing values (NAs) when reading data in CSV format. Maximum 7 characters. The default depends on whether a data column is found to contain numerical data (mostly) or string values. For numerical data the following are taken as indicating NAs: an empty cell, or any of the strings NA, N.A., na, n.a., N/A, #N/A, NaN, .NaN, ., .., -999, and -9999. For string-valued data only a blank cell, or a cell containing an empty string, is counted as NA. These defaults can be reimposed by giving default as the value for csv_read_na. To specify that only empty cells are read as NAs, give a value of "". Note that empty cells are always read as NAs regardless of the setting of this variable.
- csv_digits: a positive integer specifying the number of significant digits to use when writing data in CSV format. By default up to 15 digits are used depending on the precision of the original data. Note that CSV output employs the C library's fprintf function with "%g" conversion, which means that trailing zeros are dropped.
- display_digits: an integer from 3 to 6, specifying the number of significant digits to use when displaying regression coefficients and standard errors (the default being 6). This setting can also be used to limit the number of digits shown by the summary command; in this case the default (and also the maximum) is 5.
- mwrite_g: on or off (the default). When writing a matrix to file as text, gretl by default uses scientific notation with 18-digit precision, hence ensuring that the stored values are a faithful representation of the numbers in memory. When writing primary data with no more than 6 digits of precision it may be preferable to use %g format for a more compact and human-readable file; you can make this switch via set mwrite_g on.
- force_decpoint: on or off (the default). Force gretl to use the decimal point character, in a locale where another character (most likely the comma) is the standard decimal separator.
- loop_maxiter: one non-negative integer value (default 100000). Sets the maximum number of iterations that a while loop is allowed before halting (see loop). Note that this setting only affects the while variant; its purpose is to guard against inadvertently infinite loops. Setting this value to 0 has the effect of disabling the limit; use with caution.
- max_verbose: off (the default), on or full. Controls the verbosity of commands and functions that use numerical optimization methods. The on choice applies only to functions (such as BFGSmax and NRmax) which work silently by default; the effect is to print basic iteration information. The full setting can be used to trigger more detailed output, including parameter values and their respective gradient for the objective function at each iteration. This choice applies both to functions of the above-mentioned sort and to commands that rely on numerical optimization such as arima, probit and mle. In the case of commands the effect is to make their --verbose option produce more detail. See also chapter 37 of the *Gretl User's Guide*.
- debug: 1, 2 or 0 (the default). This is for use with user-defined functions. Setting debug to 1 is equivalent to turning messages on within all such functions; setting this variable to 2 has the additional effect of turning on max_verbose within all functions.
- shell_ok: on or off (the default). Enable launching external programs from gretl via the system shell. This is disabled by default for security reasons, and can only be enabled via the graphical user interface (Tools/Preferences/General). However, once set to on, this setting will remain active for future sessions until explicitly disabled.
- bfgs_verbskip: one integer. This setting affects the behavior of the --verbose option to those commands that use BFGS as an optimization algorithm and is used to compact output. if bfgs_verbskip is set to, say, 3, then the --verbose switch will only print iterations 3, 6, 9 and so on.

- skip_missing: on (the default) or off. Controls gretl's behavior when contructing a matrix from data series: the default is to skip data rows that contain one or more missing values but if skip_missing is set off missing values are converted to NaNs.
- matrix_mask: the name of a series, or the keyword null. Offers greater control than skip_missing when constructing matrices from series: the data rows selected for matrices are those with non-zero (and non-missing) values in the specified series. The selected mask remains in force until it is replaced, or removed via the null keyword.
- quantile_type: must be one of Q6 (the default), Q7 or Q8. Selects the specific method used by the quantile function. For details see Hyndman and Fan (1996) or the Wikipedia entry at https://en.wikipedia.org/wiki/Quantile.
- huge: a large positive number (by default, 1.0E100). This setting controls the value returned by the accessor $huge.
- assert: off (the default), warn or stop. Controls the consequences of failure (return value of 0) from the assert function.
- datacols: an integer from 1 to 15, with default value 5. Sets the maximum number of series shown side-by-side when data are displayed by observation.
- plot_collection: on, auto or off. This setting affects the way plots are displayed during interactive use. If it's on, plots of the same pixel size are gathered in a "plot collection", that is a single output window in which you can browse through the various plots going back and forth. With the off setting, instead, a different window for each plot will be generated, as in older gretl versions. Finally, the auto setting has the effect of enabling the plot collection mode only for graphs that are generated within 1.25 seconds from one another (for example, as a result of executing plotting commands in a loop).

Numerical methods

These settings are used for controlling the numerical algorithms that gretl uses for estimation.

- optimizer: either auto (the default), BFGS or newton. Sets the optimization algorithm used for various ML estimators, in cases where both BFGS and Newton–Raphson are applicable. The default is to use Newton–Raphson where an analytical Hessian is available, otherwise BFGS.
- bhhh_maxiter: one integer, the maximum number of iterations for gretl's internal BHHH routine, which is used in the arma command for conditional ML estimation. If convergence is not achieved after bhhh_maxiter, the program returns an error. The default is set at 500.
- bhhh_toler: one floating point value, or the string default. This is used in gretl's internal BHHH routine to check if convergence has occurred. The algorithm stops iterating as soon as the increment in the log-likelihood between iterations is smaller than bhhh_toler. The default value is 1.0E−06; this value may be re-established by typing default in place of a numeric value.
- bfgs_maxiter: one integer, the maximum number of iterations for gretl's BFGS routine, which is used for mle, gmm and several specific estimators. If convergence is not achieved in the specified number of iterations, the program returns an error. The default value depends on the context, but is typically of the order of 500.
- bfgs_toler: one floating point value, or the string default. This is used in gretl's BFGS routine to check if convergence has occurred. The algorithm stops as soon as the relative improvement in the objective function between iterations is smaller than bfgs_toler. The default value is the machine precision to the power 3/4; this value may be re-established by typing default in place of a numeric value.

- bfgs_maxgrad: one floating point value. This is used in gretl's BFGS routine to check if the norm of the gradient is reasonably close to zero when the bfgs_toler criterion is met. A warning is printed if the norm of the gradient exceeds 1; an error is flagged if the norm exceeds bfgs_maxgrad. At present the default is the permissive value of 5.0.
- bfgs_richardson: on or off (the default). Use Richardson extrapolation when computing numerical derivatives in the context of BFGS maximization.
- initvals: the name of a predefined matrix. Allows manual setting of the initial parameter vector for certain estimation commands that involve numerical optimization: arma, garch, logit and probit, tobit and intreg, biprobit, duration, poisson, negbin, and also when imposing certain sorts of restriction associated with VECMs. Unlike other settings, initvals is not persistent: it resets to the default initializer after its first use. For details in connection with ARMA estimation see chapter 31 of the *Gretl User's Guide*.
- lbfgs: on or off (the default). Use the limited-memory version of BFGS (L-BFGS-B) instead of the ordinary algorithm. This may be advantageous when the function to be maximized is not globally concave.
- lbfgs_mem: an integer value in the range 3 to 20 (with a default value of 8). This determines the number of corrections used in the limited memory matrix when L-BFGS-B is employed.
- nls_toler: a floating-point value. Sets the tolerance used in judging whether or not convergence has occurred in nonlinear least squares estimation using the nls command. The default value is the machine precision to the power 3/4; this value may be re-established by typing default in place of a numeric value.
- svd: on or off (the default). Use SVD rather than Cholesky or QR decomposition in least squares calculations. This option applies to the mols function as well as various internal calculations, but not to the regular ols command.
- force_qr: on or off (the default). This applies to the ols command. By default this command computes OLS estimates using Cholesky decomposition (the fastest method), with a fallback to QR if the data seem too ill-conditioned. You can use force_qr to skip the Cholesky step; in "doubtful" cases this may ensure greater accuracy.
- fcp: on or off (the default). Use the algorithm of Fiorentini, Calzolari and Panattoni rather than native gretl code when computing GARCH estimates.
- gmm_maxiter: one integer, the maximum number of iterations for gretl's gmm command when in iterated mode (as opposed to one- or two-step). The default value is 250.
- nadarwat_trim: one integer, the trim parameter used in the nadarwat function.
- fdjac_quality: one integer (0, 1 or 2), the algorithm used by the fdjac function; the default is 0.
- gmp_bits: one integer, which should be an integral power of 2 (default and minimum value 256, maximum 8192). Controls the number of bits used to represent a floating point number when GMP (the GNU Multiple Precision Arithmetic Library) is called, primarily via the mpols command. Larger values give greater precision at the cost of longer compute time. This setting can also be controlled by the environment variable GRETL_MP_BITS.

Random number generation

- seed: an unsigned integer or the keyword auto. 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. To reset the seed to a time-based automatic value, use auto.

Robust estimation

- `bootrep`: an integer. Sets the number of replications for the restrict command with the `--bootstrap` option.
- `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.
- `arma_vcv`: `hessian` (the default) or `op` (outer product matrix). Specifies the variant to be used when computing the covariance matrix for ARIMA models.
- `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 (HCCM), which does not take autocorrelation into account. Note that VARs are treated as a special case: when the `--robust` option is given the default method is regular HCCM, but the `--robust-hac` flag can be used to force the use of a HAC estimator.
- `robust_z`: `off` (the default) or `on`. This controls the distribution used when calculating p-values based on robust standard errors in the context of least-squares estimators. By default gretl uses the Student t distribution but if `robust_z` is turned on the normal distribution is used.
- `hac_lag`: `nw1` (the default), `nw2`, `nw3` or an integer. Sets the maximum lag value or bandwidth, 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, $p = 4 \times (T/100)^{2/9}$. `nw3` calls for data-based bandwidth selection. See also `qs_bandwidth` and `hac_prewhiten` below.
- `hac_kernel`: `bartlett` (the default), `parzen`, or `qs` (Quadratic Spectral). Sets the kernel, or pattern of weights, used when calculating HAC standard errors.
- `hac_prewhiten`: `on` or `off` (the default). Use Andrews-Monahan prewhitening and re-coloring when computing HAC standard errors. This also implies use of data-based bandwidth selection.
- `hac_missvals`: `es` (the default), `am` or `off`. Sets the policy regarding calculation of HAC standard errors when the estimation sample includes incomplete observations: `es` invokes the Equal Spacing method of Datta and Du (2012); `am` selects the Amplitude Modulation method of Parzen (1963); and `off` causes gretl to refuse such estimation. See chapter 22 of the *Gretl User's Guide* for details.
- `hc_version`: 0, 1, 2, 3 or 3a. Sets the variant used when calculating Heteroskedasticity Consistent standard errors with cross-sectional data. The first four 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". Variant 3a is the MacKinnon–White "jackknife" procedure. The default setting is normally 1, but this can be changed in the GUI client, via the "HCCME" tab under "/Tools/Preferences/General". Note that a setting made via the GUI persists across gretl sessions, as opposed to use of the `set` command which just affects the current session.
- `panel_robust`: `arellano` (the default), `pcse` or `scc`. This selects the robust covariance matrix estimator for use with panel-data models. See the panel command and chapter 22 of the *Gretl User's Guide* for details.
- `qs_bandwidth`: Bandwidth for HAC estimation in the case where the Quadratic Spectral kernel is selected. (Unlike the Bartlett and Parzen kernels, the QS bandwidth need not be an integer.)

Time series

- `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.
- `vecm_norm`: `phillips` (the default), `diag`, `first` or `none`. Used in the context of VECM estimation via the vecm command for identifying the cointegration vectors. See the chapter 33 of the *Gretl User's Guide* for details.
- `boot_iters`: one integer, B. Sets the number of bootstrap iterations used when computing impulse response functions with confidence intervals. The default is 1999. It is recommended that $B + 1$ is evenly divisible by $100\alpha/2$, so for example with $\alpha = 0.1$ $B + 1$ should be a multiple of 5. The minimum acceptable value is 499.

Interaction with R

- `R_lib`: `on` (the default) or `off`. When sending instructions to be executed by R, use the R shared library by preference to the R executable, if the library is available.
- `R_functions`: `off` (the default) or `on`. Recognize functions defined in R as if they were native functions (the namespace prefix "`R.`" is required). See chapter 44 of the *Gretl User's Guide* for details on this and the previous item.

Miscellaneous

- `mpi_use_smt`: `on` or `off` (the default). This switch affects the default number of processes launched in an `mpi` block within a script. If the switch is `off` the default number of processes equals the number of physical cores on the local machine; if it's `on` the default is the maximum number of threads, which will be twice the number of physical cores if the cores support SMT (Simultaneous MultiThreading, also known as Hyper-Threading). This applies only if the user has not specified a number of processes, either directly or indirectly (by specifying a `hosts` file for use with MPI).
- `graph_theme`: a string, one of `altpoints`, `classic`, `dark2` (the current default), `ethan`, `iwanthue` or `sober`. This sets the "theme" used for graphs produced by gretl. The `classic` option reverts to the single theme that was in force prior to version 2020c of gretl.

setinfo

Argument:	*series*
Options:	`--description`=*string* (set description)
	`--graph-name`=*string* (set graph name)
	`--discrete` (mark series as discrete)
	`--continuous` (mark series as continuous)
	`--coded` (mark as an encoding)
	`--numeric` (mark as not an encoding)
	`--midas` (mark as component of high-frequency data)
Examples:	`setinfo x1 --description="Description of x1"`
	`setinfo y --graph-name="Some string"`
	`setinfo z --discrete`

If the options `--description` or `--graph-name` are invoked the argument must be a single series, otherwise it may be a list of series in which case it operates on all members of the list. This command sets up to four attributes as follows.

If the `--description` flag is given followed by a string in double quotes, that string is used to set the variable's descriptive label. This label is shown in response to the labels command, and is also shown in the main window of the GUI program.

If the `--graph-name` flag is given followed by a quoted string, that string will be used in place of the variable's name in graphs.

If one or other of the `--discrete` or `--continuous` option flags is given, the variable's numerical character is set accordingly. The default is to treat all series as continuous; setting a series as discrete affects the way the variable is handled in other commands and functions, such as for example freq or dummify .

If one or other of the `--coded` or `--numeric` option flags is given, the status of the given series is set accordingly. The default is to treat all numerical values as meaningful as such, at least in an ordinal sense; setting a series as `coded` means that the numerical values are an arbitrary encoding of qualitative characteristics.

The `--midas` option sets a flag indicating that a given series holds data of a higher frequency than the base frequency of the dataset; for example, the dataset is quarterly and the series holds values for month 1, 2 or 3 of each quarter. (MIDAS = Mixed Data Sampling.)

Menu path: /Variable/Edit attributes

Other access: Main window pop-up menu

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: /Data/Set missing value code

setobs

Variants:
`setobs` *periodicity startobs*
`setobs` *unitvar timevar* `--panel-vars`

Options:
`--cross-section` (interpret as cross section)
`--time-series` (interpret as time series)
`--special-time-series` (see below)
`--stacked-cross-section` (interpret as panel data)
`--stacked-time-series` (interpret as panel data)
`--panel-vars` (use index variables, see below)
`--panel-time` (see below)
`--panel-groups` (see below)

Examples:
```
setobs 4 1990:1 --time-series
setobs 12 1978:03
setobs 1 1 --cross-section
setobs 20 1:1 --stacked-time-series
setobs unit year --panel-vars
```

This command forces the program to interpret the current data set as having a specified structure.

In the first form of the command 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 YYYY-MM-DD (or simply as 1 for undated data).

Certain time-series periodicities have standard interpretations—for example, 12 = monthly and 4 = quarterly. If you have unusual time-series data to which the standard interpretation does not apply, you can signal this by giving the `--special-time-series` option. In that case gretl will not (for example) report your frequency-12 data as being monthly.

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.

The second form of the command (which requires the `--panel-vars` flag) may be used to impose a panel interpretation when the data set contains variables that uniquely identify the cross-sectional units and the time periods. The data set will be sorted as stacked time series, by ascending values of the units variable, *unitvar*.

Panel-specific options

The `--panel-time` and `--panel-groups` options can only be used with a dataset which has already been defined as a panel.

The purpose of `--panel-time` is to set extra information regarding the time dimension of the panel. This should be given on the pattern of the first form of `setobs` noted above. For example, the following may be used to indicate that the time dimension of a panel is quarterly, starting in the first quarter of 1990.

```
setobs 4 1990:1 --panel-time
```

The purpose of `--panel-groups` is to create a string-valued series holding names for the groups (individuals, cross-sectional units) in the panel. (This will be used where appropriate in panel graphs.) With this option you supply either one or two arguments as follows.

First case: the (single) argument is the name of a string-valued series. If the number of distinct values equals the number of groups in the panel this series is used to define the group names. If necessary, the numerical content of the series will be adjusted such that the values are all 1s for the first group, all 2s for the second, and so on. If the number of string values doesn't match the number of groups an error is flagged.

Second case: the first argument is the name of a series and the second is a string literal or variable holding a name for each group. The series will be created if it does not already exist. If the second argument is a string literal or string variable the group names should be separated by spaces; if a name includes spaces it should be wrapped in backslash-escaped double-quotes. Alternatively the second argument may be an array of strings.

For example, the following will create a series named `country` in which the names in `cstrs` are each repeated T times, T being the time-series length of the panel.

```
string cstrs = sprintf("France Germany Italy \"United Kingdom\"")
setobs country cstrs --panel-groups
```

Menu path: /Data/Dataset structure

setopt

Arguments: *command* [*action*] *options*
Examples: `setopt mle --hessian`
`setopt ols persist --quiet`
`setopt ols clear`
See also `gdp_midas.inp`

This command enables the pre-setting of options for a specified command. Ordinarily this is not required, but it may be useful for the writers of hansl functions when they wish to make certain command options conditional on the value of an argument supplied by the caller.

For example, suppose a function offers a boolean "`quiet`" switch, whose intended effect is to suppress the printing of results from a certain regression executed within the function. In that case one might write:

```
if quiet
  setopt ols --quiet
endif
ols ...
```

The `--quiet` option will then be applied to the next `ols` command if and only if the variable `quiet` has a non-zero value.

By default, options set in this way apply only to the following instance of *command*; they are not persistent. However if you give `persist` as the value for *action* the options will continue to apply to the given command until further notice. The antidote to the `persist` action is `clear`: this erases any stored setting for the specified command.

It should be noted that options set via `setopt` are compounded with any options attached to the target command directly. So for example one might append the `--hessian` option to an `mle` command unconditionally but use `setopt` to add `--quiet` conditionally.

shell

Argument: *shellcommand*
Examples: `! ls -al`
`! dir c:\users`
`launch notepad`
`launch emacs myfile.txt`

The facility described here is not activated by default. See below for details.

An exclamation mark, `!`, 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. The *shellcommand* argument is passed to `/bin/sh` on unix-type systems such as Linux and macOS or to `cmd.exe` on MS Windows. It is executed in synchronous mode—gretl waits for it to complete before proceeding. If the command outputs any text this is printed to the console or script output window.

A variant of synchronous shell access allows the user to "grab" the output of a command into a string variable. This is achieved by wrapping the command in parentheses, preceded by a dollar sign, as in

```
string s = $(ls -l $HOME)
```

The `launch` keyword, on the other hand, executes an external program asynchronously (without waiting for completion), as in the third and fourth examples above. This is designed for opening an application in interactive mode. The user's `PATH` is searched for the specified executable. On MS Windows the command is executed directly, not passed to `cmd.exe` (so environment variables are not expanded automatically).

Activation

For reasons of security the shell-access facility is not enabled by default. To activate it, check the box titled "Allow shell commands" under Tools/Preferences/General in the GUI program. This also makes shell commands available in the command-line program (and is the only way to do so).

smpl

Variants:
`smpl` *startobs endobs*
`smpl` *+i -j*
`smpl` *dumvar* `--dummy`
`smpl` *condition* `--restrict`
`smpl --no-missing [` *varlist* `]`
`smpl --no-all-missing [` *varlist* `]`
`smpl --contiguous [` *varlist* `]`
`smpl` *n* `--random`
`smpl full`
`smpl`

Options:
`--dummy` (argument is a dummy variable)
`--restrict` (apply boolean restriction)
`--replace` (replace any existing boolean restriction)
`--no-missing` (restrict to valid observations)
`--no-all-missing` (omit empty observations (see below))
`--contiguous` (see below)
`--random` (form random sub-sample)
`--permanent` (see below)
`--preserve-panel` (panel data: see below)
`--unit` (panel data: sample in cross-sectional dimension)
`--time` (panel data: sample in time-series dimension)
`--dates` (interpret observation numbers as dates)
`--quiet` (don't report sample range)

Examples:
```
smpl 3 10
smpl 1960:2 1982:4
smpl +1 -1
smpl x > 3000 --restrict
smpl y > 3000 --restrict --replace
smpl 100 --random
```

This command can be used only when a dataset is in place. When no arguments are given, it displays the current sample range, otherwise it sets the sample range. The 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. (For more on *startobs* and *endobs* see the section titled "Dates versus sequential indices" below.) In the second form, the integers *i* and *j* (which may be positive or

negative, and must 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.

The options `--no-missing` and `--no-all-missing` may be used to exclude from the sample observations for which data are missing. The first variant excludes those rows in the dataset for which at least one variable has a missing value, while the second excludes just those rows on which *all* variables have missing values. In each case the test is confined to the variables in *varlist* if this argument is given, otherwise it is applied to all series—with the qualification that in the case of `--no-all-missing` and no *varlist*, the generic variables `index` and `time` are ignored.

The `--contiguous` form of `smpl` is intended for use with time series data. The effect is to trim any observations at the start and end of the current sample range that contain missing values (either for the variables in *varlist*, or for all data series if no *varlist* is given). Then a check is performed to see if there are any missing values in the remaining range; if so, an error is flagged.

With the `--random` flag, the specified number of cases are selected from the current dataset at random (without replacement). 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. (But this option is not compatible with the `--contiguous` option.)

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`: "structural" information in the data file (regarding the time series or panel nature of the data) is likely to be lost when this command is issued. You may reimpose structure with the setobs command, but also see the `--preserve-panel` option below.

Dates versus sequential indices

The `--dates` option can be used to resolve a potential ambiguity in the interpretation of *startobs* and *endobs* in the case of annual time-series data. For example, should `2010` be taken to refer to the year 2010, or to the two-thousand-and-tenth observation? In most cases this should come out right automatically but you can force the date interpretation if needed. This option can also be used with dated daily data, to get `smpl` to interpret, for example, 20100301 as the first of March 2010 rather than a plain sequential index. Note that this ambiguity does not arise with time series frequencies other than annual and daily; dates such as 1980:3 (third quarter of 1980) and 2020:03 (March 2020) cannot be confused with plain indices.

Panel-specific options

The `--unit` and `--time` options are specific to panel data. They allow you to specify, respectively, a range of "units" or time-periods. For example:

```
# limit the sample to the first 50 units
smpl 1 50 --unit
# limit the sample to periods 2 to 20
smpl 2 20 --time
```

If the time dimension of a panel dataset has been specified via the setobs command with the `--panel-time` option, `smpl` with the `--time` option can be expressed in terms of dates rather than plain observation numbers. Here's an example:

```
# specify panel time as quarterly, starting in Q1 of 1990
setobs 4 1990:1 --panel-time
# limit the sample to 2000:1 to 2007:1
smpl 2000:1 2007:1 --time
```

In gretl, a panel dataset must always be "nominally balanced"—that is, each unit must have the same number of data rows, even if some rows contain nothing but `NA`s. Sub-sampling via the `--restrict` or `--dummy` options may destroy this structure. In that case the `--preserve-panel` flag can be added to request that a nominally balanced panel is reconstituted, via the insertion of "missing rows" if needed.

Permanent versus temporary sampling

By default, restrictions on the current sample range can be undone: you can restore the full dataset via `smpl full`. However, the `--permanent` flag can be used to substitute the restricted dataset for the original. If you give the `--permanent` option with no other arguments or options the effect is to shrink the dataset to the current sample range.

Please see chapter 5 of the *Gretl User's Guide* for further details.

Menu path: /Sample

spearman

Arguments: *series1 series2*
Option: `--verbose` (print ranked data)

Prints Spearman's rank correlation coefficient for the series *series1* and *series2*. 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. For example:

```
series altx = -x
spearman altx y
```

Menu path: /Tools/Nonparametric tests/Correlation

square

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

Generates new series which are squares of the series 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: /Add/Squares of selected variables

stdize

Argument: *varlist*
Options: `--no-df-corr` (no degrees of freedom correction)
`--center-only` (don't divide by s.d.)

By default a standardized version of each of the series in *varlist* is obtained and the result stored in a new series with the prefix `s_`. For example, `stdize x y` creates the new series `s_x` and `s_y`, each of which is centered and divided by its sample standard deviation (with a degrees of freedom correction of 1).

If the `--no-df-corr` option is given no degrees of freedom correction is applied; the standard deviation used is the maximum likelihood estimator. If `--center-only` is given the series just have their means subtracted, and in that case the output names have prefix `c_` rather than `s_`.

The functionality of this command is available in somewhat more flexible form via the stdize function.

Menu path: /Add/Standardize selected variables

store

Arguments: *filename* [*varlist*]
Options: `--omit-obs` (see below, on CSV format)
`--no-header` (see below, on CSV format)
`--gnu-octave` (use GNU Octave format)
`--gnu-R` (format friendly for read.table)
`--gzipped`[=*level*] (apply gzip compression)
`--jmulti` (use JMulti ASCII format)
`--dat` (use PcGive ASCII format)
`--decimal-comma` (use comma as decimal character)
`--database` (use gretl database format)
`--overwrite` (see below, on database format)
`--comment`=*string* (see below)
`--matrix`=*matrix-name* (see below)

Save data to *filename*. By default all currently defined series are saved but the optional *varlist* argument can be used to select a subset of series. If the dataset is sub-sampled, only the observations in the current sample range are saved.

The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification.

Note that the `store` command behaves in a special manner in the context of a "progressive loop"; see chapter 13 of the *Gretl User's Guide* for details.

Native formats

If *filename* has extension `.gdt` or `.gtdb` this implies saving the data in one of gretl's native formats. In addition, if no extension is given `.gdt` is taken to be implicit and the suffix is added automatically. The `gdt` format is XML, optionally gzip-compressed, while the `gdtb` format is binary. The former is recommended for datasets of moderate size (say, up to several hundred kilobytes of data); the binary format is much faster for very large datasets.

When data are saved in `gdt` format the `--gzipped` option may be used for data compression. The optional parameter for this flag controls the level of compression (from 0 to 9): higher levels produce a smaller file, but compression takes longer. The default level is 1; a level of 0 means that no compression is applied.

A special sort of "native" save is supported in the GUI program: if *filename* has extension `.gretl` and the *varlist* argument is omitted, then a gretl session file is written. Such files include the current dataset along with any named objects such as models, graphs and matrices.

Other formats

The format in which the data are written may be controlled to a degree by the extension or suffix of *filename*, as follows:

- `.csv`: comma-separated values (CSV).
- `.txt` or `.asc`: space-separated values.
- `.m`: GNU Octave matrix format.
- `.dta`: Stata dta format (version 113).

The format-related option flags shown above can be used to force the choice of format independently of the filename (or to get gretl to write in the formats of PcGive or JMulTi).

CSV options

The option flags `--omit-obs` and `--no-header` are specific to saving data in CSV format. By default, if the data are time series or panel, or if the dataset includes specific observation markers, the output file includes a first column identifying the observations (e.g. by date). If the `--omit-obs` flag is given this column is omitted. The `--no-header` flag suppresses the usual printing of the names of the variables at the top of the columns.

The option flag `--decimal-comma` is also confined to CSV. Its effect is to replace the decimal point with decimal comma; in addition the column separator is forced to be a semicolon rather than a comma.

Storing to a database

The option of saving in gretl database format is intended for construction of large sets of series with mixed frequencies and ranges of observations. At present this option is available only for annual, quarterly or monthly time-series data, or undated (cross-sectional) data. A gretl database takes the form of two files: one with suffix `.bin` to hold the data in binary form and a plain text file with suffix `.idx` for the metadata. In naming the output file on the command line you should either give the `.bin` suffix or no suffix.

When saving to a database that already exists, the default action is to append series to the prior content. In this context it is an error if any series to be saved has the same name as one already present. The `--overwrite` flag has the effect that, if there are variable names in common, the newly saved data replace the prior values.

The `--comment` option is available when saving data as a database or as CSV. The required parameter is a double-quoted one-line string, attached to the option flag with an equals sign. The string is inserted as a comment into the database index file or at the top of the CSV output.

Writing a matrix as a dataset

The `--matrix` option requires a parameter, the name of a (non-empty) matrix. The effect of `store` is then, in effect, to turn the matrix into a dataset "in the background" and write it to file as such. Matrix columns become series; their names are taken from column-names attached to the matrix, if any, or by default are assigned as `v1`, `v2` and so on. If the matrix has row names attached these are used as "observation markers" in the dataset.

Note that matrices can be written to file in their own right, see the mwrite function. But in some cases it may be useful to write them in dataset mode.

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

summary

Variants: `summary [ varlist ]`
`summary --matrix=matname`
Options: `--simple` (basic statistics only)
`--weight=wtvar` (weighting variable)
`--by=byvar` (see below)
Example: `frontier.inp`

In its first form, this command prints summary statistics for the variables in *varlist*, or for all the variables in the data set if *varlist* is omitted. By default, output consists of the mean, median, minimum, maximum, standard deviation (sd), coefficient of variation (= sd/mean), skewness coefficient, excess kurtosis, 5th and 95th percentiles, inter-quartile range and number of missing observations. But if the `--simple` option is given, output is restricted to the mean, median, standard deviation, minimum and maximum.

If the `--weight` option is given, in which case the parameter *wtvar* should be the name of a series supplying weights per observation, the statistics are weighted accordingly.

If the `--by` option is given, in which case the parameter *byvar* should be the name of a discrete variable, then statistics are printed for sub-samples corresponding to the distinct values taken on by *byvar*. For example, if *byvar* is a (binary) dummy variable, statistics are given for the cases `byvar=0` and `byvar=1`. Note: at present, this option is incompatible with the `--weight` option.

If the alternative form is given, using a named matrix, then summary statistics are printed for each column of the matrix. The `--by` option is not available in this case.

The table of statistics produced by `summary` can be retrieved in matrix form via the $result accessor. When the `--by` option is given, this accessor is produced only if *varlist* contains a single series.

See also the aggregate function for a more flexible means of producing "factorized" statistics.

Menu path: /View/Summary statistics

Other access: Main window pop-up menu

system

Variants: `system method=estimator`
sysname `<- system`
Examples: `"Klein Model 1" <- system`
`system method=sur`
`system method=3sls`
See also `klein.inp`, `kmenta.inp`, `greene14_2.inp`

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 assign 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.
- `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.

After estimation using the `system` or `estimate` commands the following accessors can be used to retrieve additional information:

- `$uhat`: the matrix of residuals, one column per equation.
- `$yhat`: matrix of fitted values, one column per equation.
- `$coeff`: column vector of coefficients (all the coefficients from the first equation, followed by those from the second equation, and so on).
- `$vcv`: covariance matrix of the coefficients. If there are k elements in the `$coeff` vector, this matrix is k by k.
- `$sigma`: cross-equation residual covariance matrix.
- `$sysGamma`, `$sysA` and `$sysB`: structural-form coefficient matrices (see below).

If you want to retrieve the residuals or fitted values for a specific equation as a data series, select a column from the `$uhat` or `$yhat` matrix and assign it to a series, as in

```
series uh1 = $uhat[,1]
```

The structural-form matrices correspond to the following representation of a simultaneous equations model:

$$\Gamma y_t = A y_{t-1} + B x_t + \epsilon_t$$

If there are n endogenous variables and k exogenous variables, Γ is an $n \times n$ matrix and B is $n \times k$. If the system contains no lags of the endogenous variables then the A matrix is not present. If the maximum lag of an endogenous regressor is p, the A matrix is $n \times np$.

Menu path: /Model/Simultaneous equations

tabprint

Options: `--output=`*filename* (send output to specified file)
`--format="f1|f2|f3|f4"` (Specify custom TeX format)
`--complete` (TeX-related, see below)

Must follow the estimation of a model. Prints the model in tabular form. The format is governed by the extension of the specified *filename*: "`.tex`" for LaTeX, "`.rtf`" for RTF (Microsoft's Rich Text Format), or "`.csv`" for comma-separated. The file will be written in the currently set workdir, unless *filename* contains a full path specification.

If CSV format is selected, values are comma-separated unless the decimal comma is in force, in which case the separator is the semicolon.

Options specific to LaTeX output

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

If you wish alter the appearance of the tabular output, you can specify a custom row format using the `--format` flag. The format string must be enclosed in double quotes and must be tied to the flag with an equals sign. The pattern for the format string is as follows. There are four fields, representing the coefficient, standard error, t-ratio and p-value respectively. These fields should be separated by vertical bars; they may contain a `printf`-type specification for the formatting of the numeric value in question, or may be left blank to suppress the printing of that column (subject to the constraint that you can't leave all the columns blank). Here are a few examples:

```
--format="%.4f|%.4f|%.4f|%.4f"
--format="%.4f|%.4f|%.3f|"
--format="%.5f|%.4f||%.4f"
--format="%.8g|%.8g||%.4f"
```

The first of these specifications prints the values in all columns using 4 decimal places. The second suppresses the p-value and prints the t-ratio to 3 places. The third omits the t-ratio. The last one again omits the t, and prints both coefficient and standard error to 8 significant figures.

Once you set a custom format in this way, it is remembered and used for the duration of the gretl session. To revert to the default format you can use the special variant `--format=default`.

Menu path: Model window, /LaTeX

textplot

Argument: *varlist*
Options: `--time-series` (plot by observation)
`--one-scale` (force a single scale)
`--tall` (use 40 rows)

Quick and simple ASCII graphics. Without the `--time-series` flag, *varlist* must contain at least two series, the last of which is taken as the variable for the x axis, and a scatter plot is produced. In this case the `--tall` option may be used to produce a graph in which the y axis is represented by 40 rows of characters (the default is 20 rows).

With the `--time-series`, a plot by observation is produced. In this case the option `--one-scale` may be used to force the use of a single scale; otherwise if *varlist* contains more than one series the data may be scaled. Each line represents an observation, with the data values plotted horizontally.

See also gnuplot.

tobit

Arguments: *depvar indepvars*
Options: `--llimit`=*lval* (specify left bound)
`--rlimit`=*rval* (specify right bound)
`--vcv` (print covariance matrix)
`--robust` (robust standard errors)
`--opg` (see below)
`--cluster`=*clustvar* (see logit for explanation)
`--verbose` (print details of iterations)
`--quiet` (don't print results)

Estimates a Tobit model, which may be appropriate when the dependent variable is "censored". 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.

By default it is assumed that the dependent variable is censored at zero on the left and is uncensored on the right. However you can use the options `--llimit` and `--rlimit` to specify a different pattern of censoring. Note that if you specify a right bound only, the assumption is then that the dependent variable is uncensored on the left.

The Tobit model is a special case of interval regression. Please see the intreg command for further details, including an account of the `--robust` and `--opg` options.

Menu path: /Model/Limited dependent variable/Tobit

tsls

Arguments: *depvar indepvars* ; *instruments*
Options: `--no-tests` (don't do diagnostic tests)
`--vcv` (print covariance matrix)
`--quiet` (don't print results)
`--no-df-corr` (no degrees-of-freedom correction)
`--robust` (robust standard errors)
`--cluster`=*clustvar* (clustered standard errors)
`--matrix-diff` (compute Hausman test via matrix difference)
`--liml` (use Limited Information Maximum Likelihood)
`--gmm` (use the Generalized Method of Moments)
Example:
```
tsls y1 0 y2 y3 x1 x2 ; 0 x1 x2 x3 x4 x5 x6
penngrow.inp
```

Computes Instrumental Variables (IV) estimates, by default using two-stage least squares (TSLS) but see below for further options. The dependent variable is *depvar*, *indepvars* is the list of regressors (which is presumed to include at least one endogenous variable); and *instruments* is the list of instruments (exogenous and/or predetermined variables). 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 endogenous and the `x`s are the exogenous variables. Note that exogenous regressors should appear in both lists.

For details on the effects of the `--robust` and `--cluster` options, please see the help for ols.

TSLS-specific tests

Output for two-stage least squares estimates includes the Hausman test and, if the model is overidentified, the Sargan overidentification test. For a good explanation of both tests see chapter 8 of Davidson and MacKinnon (2004).

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 really required. By default this test is implemented by the regression method, but if the `--matrix-diff` option is given the method of Papadopoulos (2023) is used. In both cases a robust variant is employed if the `--robust` option is also given.

A model of this sort is overidentified if there are more instruments than are strictly required. The Sargan overidentification test (Sargan, 1958) 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.

These statistics are available, upon successful completion of the command, under the names $hausman and $sargan (if applicable), respectively.

Weak instruments

For both TSLS and LIML estimation, an additional test result is shown provided that the model is estimated under the assumption of i.i.d. errors (that is, the `--robust` option is not selected). This is a test for weakness of the instruments. Weak instruments can lead to serious problems in IV regression: biased estimates and/or incorrect size of hypothesis tests based on the covariance matrix, with rejection rates well in excess of the nominal significance level (Stock *et al.*, 2002). The test statistic is the first-stage F-test if the model contains just one endogenous regressor, otherwise it is the smallest eigenvalue of the matrix counterpart of the first stage F. Critical values based on the Monte Carlo analysis of Stock and Yogo (2003) are shown when available.

R-squared

The R-squared value printed for models estimated via two-stage least squares is the square of the correlation between the dependent variable and the fitted values.

Alternative estimators

As alternatives to TSLS, the model may be estimated via Limited Information Maximum Likelihood (the `--liml` option) or via the Generalized Method of Moments (`--gmm` option). Note that if the model is just identified these methods should produce the same results as TSLS, but if it is overidentified the results will differ in general.

If GMM estimation is selected, the following additional options become available:

- `--two-step`: perform two-step GMM rather than the default of one-step.
- `--iterate`: Iterate GMM to convergence.
- `--weights=`*Wmat*: specify a square matrix of weights to be used when computing the GMM criterion function. The dimension of this matrix must equal the number of instruments. The default is an appropriately sized identity matrix.

Menu path: /Model/Instrumental variables

tsplots

Argument:	*varlist*
Options:	`--matrix=`*name* (plot columns of named matrix)
	`--output=`*filename* (send output to specified file)
Examples:	`tsplots 1 2 3 4`
	`tsplots 1 2 3 4 --matrix=X`

Provides an easy way of plotting multiple time series (up to a maximum of 16) on a single canvas. The *varlist* argument can be given as a list of ID numbers or names of series, or as column numbers in the case of matrix input.

See also scatters for means of producing multiple scatterplots, and gridplot for a more flexible way of combining plots in a grid.

Menu path: /View/Multiple graphs/Time series

var

Arguments:	*order ylist* `[ ;` *xlist* `]`
Options:	`--nc` (do not include a constant)
	`--trend` (include a linear trend)
	`--seasonals` (include seasonal dummy variables)
	`--robust` (robust standard errors)
	`--robust-hac` (HAC standard errors)
	`--quiet` (skip output of individual equations)
	`--silent` (don't print anything)
	`--impulse-responses` (print impulse responses)
	`--variance-decomp` (print variance decompositions)
	`--lagselect` (show criteria for lag selection)
	`--minlag=`*minimum lag* (lag selection only, see below)
Examples:	`var 4 x1 x2 x3 ; time mydum`
	`var 4 x1 x2 x3 --seasonals`
	`var 12 x1 x2 x3 --lagselect`
	See also `sw_ch14.inp`

Sets up and estimates (using OLS) a vector autoregression (VAR). The first argument specifies the lag order — or the maximum lag order in case the `--lagselect` option is given (see below). The order may be given numerically, or as the name of a pre-existing scalar variable. Then follows the setup for the first equation. Do not include lags among the elements of *ylist* — they will be added automatically. The semi-colon separates the stochastic variables, for which *order* lags will be included, from any exogenous variables in *xlist*. Note that a constant is included automatically unless you give the `--nc` flag, a trend can be added with the `--trend` flag, and seasonal dummy variables may be added using the `--seasonals` flag.

While a VAR specification usually includes all lags from 1 to a given maximum, it is possible to select a specific set of lags. To do this, substitute for the regular (scalar) *order* argument either the name of a predefined vector or a comma-separated list of lags, enclosed in braces. We show below two ways of specifying that a VAR should include lags 1, 2 and 4 (but not lag 3):

```
var {1,2,4} ylist
matrix p = {1,2,4}
var p ylist
```

A separate regression is reported for each variable in *ylist*. 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. For retrieval of a specified impulse response function in matrix form, see the irf function.

If the `--robust` option is given, standard errors are corrected for heteroskedasticity. Alternatively, the `--robust-hac` option can be given to produce standard errors that are robust with respect to both heteroskedasticity and autocorrelation (HAC). In general the latter correction should not be needed if the VAR includes sufficient lags.

Lag selection

If the `--lagselect` option is given, the usual VAR output is not presented. Instead the first argument is taken as the *maximum* lag order and the output consists of a table showing comparative figures computed for VARs of order 1 (by default) up to the specified maximum. The table includes log-likelihood and the P-value for a Likelihood Ratio (LR) test, followed by the Akaike (AIC), Schwarz (BIC) and Hannan–Quinn (HQC) information criteria. The LR test compares the specification on row i with that on row $i-1$, the null hypothesis being that all the parameters added on row i have zero values. The table of results may be retrieved in matrix form via the $test accessor.

In the lag-selection context (only) the `--minlag` option can be used to adjust the minimum lag order. Set this to 0 to allow for the possibility that the optimal lag order is zero (meaning that a VAR is not really called for at all). Conversely you could set `--minlag=4` if you believe you need at least 4 lags, thereby saving a little compute time.

Menu path: /Model/Multivariate time series

varlist

Option: `--type`=*typename* (scope of listing)

By default, prints a listing of the series in the current dataset (if any); `ls` may be used as an alias.

If the `--type` option is given, it should be followed (after an equals sign) by one of the following typenames: `series`, `scalar`, `matrix`, `list`, `string`, `bundle`, `array` or `accessor`. The effect is to print the names of all currently defined objects of the named type.

As a special case, if the typename is `accessor`, the names printed are those of the internal variables currently available as "accessors", such as $nobs and $uhat, regardless of their specific type.

vartest

Variants: `vartest` *x y*
`vartest` *x* `--split-by=`*dummy*

Options: `--quiet` (suppress printed output)
`--robust=`*method* (see below)

Examples:
```
vartest x y
meantest x --split-by=d
meantest x y --robust=median
meantest x y --robust=trimmed,5
```

In its primary usage, calculates the F statistic for the null hypothesis that the population variances are equal for the series x and y, and shows output including the p-value. Results can be retrieved

using the accessors $test and $pvalue, in which case the --quiet option may used to omit the printout.

For example, the following code:

```
open AWM18.gdt
vartest EEN EXR
eval $test
eval $pvalue
```

produces the output shown below:

```
Equality of variances test

EEN: Number of observations = 192
EXR: Number of observations = 188
Ratio of sample variances = 3.70707
Null hypothesis: The two population variances are equal
Test statistic: F(191,187) = 3.70707
p-value (two-tailed) = 1.94866e-18

3.7070716
1.9486605e-18
```

In its alternate form, with the --split-by option, the samples whose variances are tested for equality are two subsets of the series *x*, for which the series *dummy* takes the values 0 and 1, respectively.

Robust tests

The standard *F* test rests on the assumption of normality and can over-reject substantially if the data are in fact non-normal. The --robust option offers three alternatives, as follows:

- When *method* is mean, the test defined by Levene (1960). This is in effect a one-way ANOVA using the absolute deviations of the original data from their respective sample means.
- When *method* is median, a variant on Levene's test suggested by Brown and Forsythe (1974), where the initial centering uses the sample median rather than the mean. This is recommended if the data are substantially skewed.
- When *method* is trimmed, a second variant due to Brown and Forsythe, where centering is relative to a trimmed mean. This may be preferable if the data are fat-tailed but reasonably symmetrical. By default 10 percent trimming is employed (that is, the least and the greatest 10 percent of observations are omitted when calculating the mean) but this can be adjusted by appending a comma and an integer value, as in --robust=trimmed,5 which specifies 5 percent trimming.

Menu path: /Tools/Test statistic calculator

vecm

Arguments:	*order rank ylist* `[ ; ` *xlist* `]` `[ ; ` *rxlist* `]`
Options:	`--nc` (no constant)
	`--rc` (restricted constant)
	`--uc` (unrestricted constant)
	`--crt` (constant and restricted trend)
	`--ct` (constant and unrestricted trend)
	`--seasonals` (include centered seasonal dummies)
	`--quiet` (skip output of individual equations)
	`--silent` (don't print anything)
	`--impulse-responses` (print impulse responses)
	`--variance-decomp` (print variance decompositions)
Examples:	`vecm 4 1 Y1 Y2 Y3`
	`vecm 3 2 Y1 Y2 Y3 --rc`
	`vecm 3 2 Y1 Y2 Y3 ; X1 --rc`
	See also `denmark.inp`, `hamilton.inp`

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 johansen).

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 *ylist*.

ylist 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 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 chapter 33 of the *Gretl User's Guide*.

The optional lists *xlist* and *rxlist* allow you to specify sets of exogenous variables which enter the model either unrestrictedly (*xlist*) or restricted to the cointegration space (*rxlist*). These lists are separated from *ylist* and from each other by semicolons.

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.

Following estimation of a VECM some special accessors are available: `$jalpha`, `$jbeta` and `$jvbeta` retrieve, respectively, the α and β matrices and the estimated variance of β. For retrieval of a specified impulse response function in matrix form, see the irf function.

Menu path: /Model/Multivariate time series

vif

Option: `--quiet` (don't print anything)
Example: `longley.inp`

Must follow the estimation of a model which includes at least two independent variables. Calculates and displays diagnostic information pertaining to collinearity.

The Variance Inflation Factor or 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 *et al.* (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.

Following this command the $result accessor may be used to retrieve a column vector holding the VIFs. For a more sophisticated approach to diagnosing collinearity, see the bkw command.

Menu path: Model window, /Analysis/Collinearity

wls

Arguments: *wtvar depvar indepvars*
Options: `--vcv` (print covariance matrix)
`--robust` (robust standard errors)
`--quiet` (suppress printing of results)
`--allow-zeros` (see below)

Computes weighted least squares (WLS) estimates using *wtvar* as the weight, *depvar* as the dependent variable, and *indepvars* as the list of independent variables. Let w denote the positive square root of `wtvar`; then WLS is basically equivalent to an OLS regression of w * *depvar* on w * *indepvars*. The R-squared, however, is calculated in a special manner, namely as

$$R^2 = 1 - \frac{\text{ESS}}{\text{WTSS}}$$

where ESS is the error sum of squares (sum of squared residuals) from the weighted regression and WTSS denotes the "weighted total sum of squares", which equals the sum of squared residuals from a regression of the weighted dependent variable on the weighted constant alone.

As a special case, if *wtvar* is a 0/1 dummy variable, WLS estimation is equivalent to OLS on a sample that excludes all observations with value zero for *wtvar*. Otherwise including weights of zero is considered an error, but if you really want to mix zero weights with positive ones you can append the `--allow-zeros` option.

For weighted least squares estimation applied to panel data and based on the unit specific error variances please see the panel command with the `--unit-weights` option.

Menu path: /Model/Other linear models/Weighted Least Squares

xcorrgm

Arguments: *series1 series2* [*order*]
Options: `--plot`=*mode-or-filename* (see below)
`--silent` (suppress printed output)
Example: `xcorrgm x y 12`

Prints and/or graphs the cross-correlogram for *series1* and *series2*, which may be specified by name or number. The values are the sample correlation coefficients between the current value of *series1* and successive leads and lags of *series2*.

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

By default, when gretl is not in batch mode a plot of the cross-correlogram is shown. This can be adjusted via the `--plot` option. The acceptable parameters to this option are `none` (to suppress the plot); `display` (to produce a gnuplot graph even when in batch mode); or a file name. The effect of providing a file name is as described for the `--output` option of the gnuplot command.

Menu path: /View/Cross-correlogram

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

xtab

Arguments:	*ylist* `[ ;` *xlist* `]`
Options:	`--row` (display row percentages)
	`--column` (display column percentages)
	`--zeros` (display zero entries)
	`--no-totals` (suppress printing of marginal counts)
	`--matrix=`*matname* (use frequencies from named matrix)
	`--quiet` (suppress printed output)
	`--tex[=`*filename*`]` (output as LaTeX)
	`--equal` (see the LaTeX case below)
Examples:	`xtab 1 2`
	`xtab 1 ; 2 3 4`
	`xtab --matrix=A`
	`xtab 1 2 --tex="xtab.tex"`
	See also `ooballot.inp`

Given just the *ylist* argument, computes (and by default prints) a contingency table or cross-tabulation for each combination of the variables included in the list. If a second list *xlist* is given, each variable in *ylist* is cross-tabulated by row against each variable in *xlist* (by column). Variables in these lists can be referenced by name or by number. Note that all the variables must have been marked as discrete. Alternatively, if the `--matrix` option is given, the named matrix is treated as a precomputed set of frequencies, to be displayed as a cross-tabulation (see also the mxtab function). In this case the *list* argument(s) should be omitted.

By default the cell entries are given as frequency counts. The `--row` and `--column` options (which are mutually exclusive) replace the counts with the percentages for each row or column, respectively. By default, cells with a zero count are left blank but the `--zeros` option has the effect of showing zero counts explicitly, which may be useful for importing the table into another program, such as a spreadsheet.

Pearson's chi-square test for independence is shown if the expected frequency under independence is at least 1.0e-7 for all cells. A common rule of thumb for the validity of this statistic is that at least 80 percent of cells should have expected frequencies of 5 or greater; if this criterion is not met a warning is printed.

If the contingency table is 2 by 2, Fisher's Exact Test for independence is shown. Note that this test is based on the assumption that the row and column totals are fixed, which may or may not be appropriate depending on how the data were generated. The left p-value should be used when the alternative to independence is negative association (values tend to cluster in the lower left and

upper right cells), the right p-value when the alternative is positive association. The two-tailed p-value for this test is calculated by method (b) in section 2.1 of Agresti (1992): it is the sum of the probabilities of all possible tables with the given row and column totals and a probability no greater than that of the observed table.

The bivariate case

In the case of a bivariate cross-tabulation (only one list is given, and it has two members) certain results are stored. The contingency table may be retrieved in matrix form via the $result accessor. In addition, if the minimum expected value condition is met, the Pearson chi-square test and its p-value may be retrieved via the $test and $pvalue accessors. If it's these results that are of interest, the `--quiet` option can be used to suppress the usual printout.

LaTeX output

If the `--tex` option is given the cross-tabulation is printed in the form of a LaTeX `tabular` environment, either inline (from where it may be copied and pasted) or, if the *filename* parameter is appended, to the specified file. (If *filename* does not specify a full path the file is written in the currently set workdir.) No test statistic is computed. The additional option `--equal` can be used to flag, by printing in boldface, the count or percentage for cells in which the row and column variables have the same numerical value. This option is ignored unless the `--tex` option is given, and also when one or both of the cross-tabulated variables are string-valued.

1.3 Commands by topic

The following sections show the available commands grouped by topic.

Estimation

ar	Autoregressive estimation	ar1	AR(1) estimation
arch	ARCH model	arima	ARIMA model
arma	ARMA model	biprobit	Bivariate probit
dpanel	Dynamic panel models	duration	Duration models
equation	Define equation within a system	estimate	Estimate system of equations
garch	GARCH model	gmm	GMM estimation
heckit	Heckman selection model	hsk	Heteroskedasticity-corrected estimates
intreg	Interval regression model	lad	Least Absolute Deviation estimation
logistic	Logistic regression	logit	Logit regression
midasreg	MIDAS regression	mle	Maximum likelihood estimation
mpols	Multiple-precision OLS	negbin	Negative Binomial regression
nls	Nonlinear Least Squares	ols	Ordinary Least Squares
panel	Panel models	poisson	Poisson estimation
probit	Probit model	quantreg	Quantile regression
system	Systems of equations	tobit	Tobit model
tsls	Instrumental variables regression	var	Vector Autoregression
vecm	Vector Error Correction Model	wls	Weighted Least Squares

Tests

add	Add variables to model	adf	Augmented Dickey-Fuller test

bds	BDS nonlinearity test	bkw	Collinearity Diagnostics
chow	Chow test	coeffsum	Sum of coefficients
coint	Engle-Granger cointegration test	cusum	CUSUM test
difftest	Nonparametric tests for differences	johansen	Johansen cointegration test
kpss	KPSS stationarity test	leverage	Influential observations
levinlin	Levin-Lin-Chu test	meantest	Difference of means
modtest	Model tests	normtest	Normality test
omit	Omit variables	panspec	Panel specification
qlrtest	Quandt likelihood ratio test	reset	Ramsey's RESET
restrict	Testing restrictions	runs	Runs test
vartest	Difference of variances	vif	Variance Inflation Factors

Transformations

diff	First differences	discrete	Mark variables as discrete
dummify	Create sets of dummies	lags	Create lags
ldiff	Log-differences	logs	Create logs
orthdev	Orthogonal deviations	sdiff	Seasonal differencing
square	Create squares of variables	stdize	Standardize series

Statistics

anova	ANOVA	corr	Correlation coefficients
corrgm	Correlogram	fractint	Fractional integration
freq	Frequency distribution	hurst	Hurst exponent
mahal	Mahalanobis distances	pca	Principal Components Analysis
pergm	Periodogram	pvalue	Compute p-values
spearman	Spearmans's rank correlation	summary	Descriptive statistics
xcorrgm	Cross-correlogram	xtab	Cross-tabulate variables

Dataset

append	Append data	data	Import from database
dataset	Manipulate the dataset	delete	Delete variables
genr	Generate a new variable	info	Information on data set
join	Manage data sources	labels	Labels for variables
markers	Observation markers	nulldata	Creating a blank dataset
open	Open a data file	rename	Rename variables
setinfo	Edit attributes of variable	setmiss	Missing value code
setobs	Set frequency and starting observation	smpl	Set the sample range
store	Save data	varlist	Listing of variables

Graphs

boxplot	Boxplots	gnuplot	Create a gnuplot graph
graphpg	Gretl graph page	gridplot	
gpbuild		hfplot	Create a MIDAS plot

kdplot	Kernel density plot	panplot	plot a panel series
plot		qqplot	Q-Q plot
rmplot	Range-mean plot	scatters	Multiple pairwise graphs
textplot	ASCII plot	tsplots	Multiple time-series plots

Printing

eqnprint	Print model as equation	modprint	Print a user-defined model
outfile	Direct printing to file	print	Print data or strings
printf	Formatted printing	tabprint	Print model in tabular form

Prediction

fcast	Generate forecasts

Programming

break	Break from loop	catch	Catch errors
clear		continue	Skip forward in loop
elif	Flow control	else	Flow control
end	End block of commands	endif	Flow control
endloop	End a command loop	flush	
foreign	Non-native script	funcerr	Exit on error
function	Define a function	if	Flow control
include	Include function definitions	loop	Start a command loop
makepkg	Make function package	mpi	Message Passing Interface
run	Execute a script	set	Set program parameters
setopt	Set options for next command		

Utilities

eval		help	Help on commands
modeltab	The model table	pkg	
quit	Exit the program	shell	Execute shell commands

1.4 Short-form command options

As can be seen from section 1.2, the behavior of many gretl commands can be modified via the use of option flags. These take the form of two dashes followed by a string which is somewhat descriptive of the effect of the option.

Some options require a parameter, which must be joined to the option "flag" with an equals sign. Among the options that do *not* require a parameter, certain common ones have a short form—a single dash followed by a single letter—and it is considered idiomatic to use the short forms in hansl scripts. The table below shows the relevant mapping: for any command which supports the long-form option in the first column, the short form in the second column is also supported.

long form	short form
`--verbose`	`-v`
`--quiet`	`-q`
`--robust`	`-r`
`--hessian`	`-h`
`--window`	`-w`

Chapter 2

Gretl functions

2.1 Introduction

This chapter presents two listings:

- "Accessors", whose names start with $ and which serve to retrieve the values of internal variables or constants. These do not take any arguments.
- Functions proper. In almost all cases these require at least one argument, and even if no argument is required an empty "argument slot" () is mandatory.

2.2 Accessors

$ahat

Output: series

Must follow the estimation of a fixed-effects or random-effects panel data model. Returns a series containing the estimates of the individual effects.

$aic

Output: scalar

Returns the Akaike Information Criterion for the last estimated model, if available. See chapter 28 of the *Gretl User's Guide* for details of the calculation.

$allprobs

Output: matrix

Must follow estimation via ordered probit or logit, or multinomial logit. Returns an $n \times j$ matrix, where n is the number of observations used and j is the number of possible outcomes, holding the estimated probability of each outcome at each observation.

$bic

Output: scalar

Returns Schwarz's Bayesian Information Criterion for the last estimated model, if available. See chapter 28 of the *Gretl User's Guide* for details of the calculation.

$chisq

Output: scalar

Returns the overall chi-square statistic from the last estimated model, if available.

$coeff

Output: matrix or scalar
Argument: *s* (name of coefficient, optional)

With no arguments, `$coeff` returns a column vector containing the estimated coefficients for the last model. With the optional string argument it returns a scalar, namely the estimated parameter named *s*. See also $stderr, $vcv.

Example:

```
open bjg
arima 0 1 1 ; 0 1 1 ; lg
b = $coeff               # gets a vector
macoef = $coeff(theta_1) # gets a scalar
```

If the "model" in question is actually a system, the result depends on the characteristics of the system: for VARs and VECMs the value returned is a matrix with one column per equation, otherwise it is a column vector containing the coefficients from the first equation followed by those from the second equation, and so on.

$command

Output: string

Must follow the estimation of a model; returns the command word, for example `ols` or `probit`.

$compan

Output: matrix

Must follow the estimation of a VAR or a VECM; returns the companion matrix.

$datatype

Output: scalar

Returns an integer value representing the sort of dataset that is currently loaded: 0 = no data; 1 = cross-sectional (undated) data; 2 = time-series data; 3 = panel data.

$depvar

Output: string

Must follow the estimation of a single-equation model; returns the name of the dependent variable.

$df

Output: scalar

Returns the degrees of freedom of the last estimated model. If the last model was in fact a system of equations, the value returned is the degrees of freedom per equation; if this differs across the equations then the value given is the number of observations minus the mean number of coefficients per equation (rounded up to the nearest integer).

$diagpval

Output: scalar

Must follow estimation of a system of equations. Returns the *P*-value associated with the $diagtest statistic.

$diagtest

Output: scalar

Must follow estimation of a system of equations. Returns the test statistic for the null hypothesis that the cross-equation covariance matrix is diagonal. This is the Breusch–Pagan test except when the estimator is (unrestricted) iterated SUR, in which case it is a Likelihood Ratio test. See chapter 34 of the *Gretl User's Guide* for details; see also $diagpval.

$dotdir

Output: string

This accessor returns the path where gretl stores temporary files, for example when using the mwrite function with a non-zero third argument.

$dw

Output: scalar

Returns the Durbin–Watson statistic for first-order serial correlation from the model last estimated (if available).

$dwpval

Output: scalar

Returns the CDF of the Durbin–Watson distribution evaluated at the DW statistic for the model last estimated (if available), computed using the Imhof (1961) procedure. This is the p-value for a one-sided test with an alternative of positive first-order autocorrelation. If you want the p-value for a two-sided test, take $2P$ if DW < 2 or $2(1 - P)$ if DW > 2, where P is the value returned by the accessor.

Due to the limited precision of digital arithmetic, the Imhof integral can go negative when the Durbin–Watson statistic is close to its lower bound. In that case the accessor returns NA. Since any other failure mode results in an error being flagged it is probably safe to assume that an NA value means the true p-value is "very small", although we are unable to quantify it.

$ec

Output: matrix

Must follow the estimation of a VECM; returns a matrix containing the error correction terms. The number of rows equals the number of observations used and the number of columns equals the cointegration rank of the system.

$error

Output: scalar

Returns the program's internal error code, which will be non-zero in case an error has occurred but has been trapped using catch. Note that using this accessor causes the internal error code to

be reset to zero. If you want to get the error message associated with a given `$error` you need to store the value in a temporary variable, as in

```
err = $error
if (err)
    printf "Got error %d (%s)\n", err, errmsg(err)
endif
```

See also catch, errmsg.

$ess

Output: scalar

Returns the error sum of squares of the last estimated model, if available.

$evals

Output: matrix

Must follow the estimation of a VECM; returns a vector containing the eigenvalues that are used in computing the trace test for cointegration.

$fcast

Output: matrix

Must follow the fcast forecasting command; returns the forecast values as a matrix. If the model on which the forecast was based is a system of equations the returned matrix will have one column per equation, otherwise it is a column vector.

$fcse

Output: matrix

Must follow the fcast forecasting command; returns the standard errors of the forecasts, if available, as a matrix. If the model on which the forecast was based is a system of equations the returned matrix will have one column per equation, otherwise it is a column vector.

$fevd

Output: matrix

Must follow estimation of a VAR. Returns a matrix containing the forecast error variance decomposition (FEVD). This matrix has h rows where h is the forecast horizon, which can be chosen using `set horizon` or otherwise is set automatically based on the frequency of the data.

For a VAR with p variables, the matrix has p^2 columns: the first p columns contain the FEVD for the first variable in the VAR; the second p columns the FEVD for the second variable; and so on. The (decimal) fraction of the forecast error for variable i attributable to innovation in variable j is therefore found in column $(i-1)p+j$.

For a more flexible variant of this functionality, see the fevd function.

$Fstat

Output: scalar

Returns the overall F-statistic from the last estimated model, if available.

$gmmcrit

Output: scalar

Must follow a `gmm` block. Returns the value of the GMM objective function at its minimum.

$h

Output: series

Must follow a `garch` command. Returns the estimated conditional variance series.

$hausman

Output: row vector

Must follow estimation of a model via either `tsls` or `panel` with the random effects option. Returns a 1×3 vector containing the value of the Hausman test statistic, the corresponding degrees of freedom and the p-value for the test, in that order.

$hqc

Output: scalar

Returns the Hannan-Quinn Information Criterion for the last estimated model, if available. See chapter 28 of the *Gretl User's Guide* for details of the calculation.

$huge

Output: scalar

Returns a very large positive number. By default this is 1.0E100, but the value can be changed using the set command.

$jalpha

Output: matrix

Must follow the estimation of a VECM, and returns the loadings matrix. It has as many rows as variables in the VECM and as many columns as the cointegration rank.

$jbeta

Output: matrix

Must follow the estimation of a VECM, and returns the cointegration matrix. It has as many rows as variables in the VECM (plus the number of exogenous variables that are restricted to the cointegration space, if any), and as many columns as the cointegration rank.

$jvbeta

Output: square matrix

Must follow the estimation of a VECM, and returns the estimated covariance matrix for the elements of the cointegration vectors.

In the case of unrestricted estimation, this matrix has a number of rows equal to the unrestricted elements of the cointegration space after the Phillips normalization. If, however, a restricted system is estimated via the `restrict` command with the `--full` option, a singular matrix with $(n + m)r$

rows will be returned (n being the number of endogenous variables, m the number of exogenous variables that are restricted to the cointegration space, and r the cointegration rank).

Example: the code

```
open denmark.gdt
vecm 2 1 LRM LRY IBO IDE --rc --seasonals -q
s0 = $jvbeta

restrict --full
  b[1,1] = 1
  b[1,2] = -1
  b[1,3] + b[1,4] = 0
end restrict
s1 = $jvbeta

print s0
print s1
```

produces the following output.

```
s0 (4 x 4)

 0.019751     0.029816  -0.00044837   -0.12227
 0.029816     0.31005   -0.45823      -0.18526
-0.00044837  -0.45823    1.2169       -0.035437
-0.12227     -0.18526   -0.035437      0.76062

s1 (5 x 5)

 0.0000       0.0000       0.0000       0.0000       0.0000
 0.0000       0.0000       0.0000       0.0000       0.0000
 0.0000       0.0000       0.27398     -0.27398     -0.019059
 0.0000       0.0000      -0.27398      0.27398      0.019059
 0.0000       0.0000      -0.019059     0.019059     0.0014180
```

$llt

Output: series

For selected models estimated via Maximum Likelihood, returns the series of per-observation log-likelihood values. At present this is supported only for binary logit and probit, tobit and heckit.

$lnl

Output: scalar

Returns the log-likelihood for the last estimated model (where applicable).

$macheps

Output: scalar

Returns the value of "machine epsilon", which gives an upper bound on the relative error due to rounding in double-precision floating point arithmetic.

$mapfile

Output: string

If data from a GeoJSON file or ESRI shapefile have been loaded, returns the name of the file that should be opened to obtain the map polygons, otherwise returns an empty string. This is designed for use with the geoplot function.

$mnlprobs

Output: matrix

Following estimation of a multinomial logit model (only), retrieves a matrix holding the estimated probabilities of each possible outcome at each observation in the model's sample range. Each row represents an observation and each column an outcome. As of gretl 2023a this accessor is deprecated: please use $allprobs instead.

$model

Output: bundle

Must follow estimation of a single-equation model; returns a bundle containing many items of data pertaining to the model. All the regular model accessors are included: these are referenced by keys that are the same as the regular accessor names, minus the leading dollar sign. So for example the residuals appear under the key `uhat` and the error sum of squares under `ess`.

Depending on the estimator, additional information may be available; the keys for such information should hopefully be fairly self-explanatory. To see what's available you can get a copy of the bundle and print its content, as in

```
ols y 0 x
bundle b = $model
print b
```

$mpirank

Output: integer

If gretl is built with MPI support, and the program is running in MPI mode, returns the 0-based "rank" or ID number of the current process. Otherwise returns −1.

$mpisize

Output: integer

If gretl is built with MPI support, and the program is running in MPI mode, returns the number of MPI processes currently running. Otherwise returns 0.

$ncoeff

Output: integer

Returns the total number of coefficients estimated in the last model.

$nobs

Output: integer

Returns the number of observations in the currently selected sample. Related: $tmax.

In the case of panel data the value returned is the number of pooled observations (number of units times number of observations per unit). If you want the time-series length of a panel use $pd, and the number of included units can be found as `$nobs` divided by `$pd`.

$now

Output: vector

Returns a 2-vector: its first element is the number of seconds elapsed since 1970-01-01 00:00:00 +0000 (UTC, or Coordinated Universal Time), which is widely used in the computing world to represent the current time, and the second is the current date in ISO 8601 "basic" format, YYYYMMDD. The strftime function may be used to process the first element, and epochday may be used to process the second.

$nvars

Output: integer

Returns the number of series in the dataset (including the constant). Since `const` is always present in any dataset a return value of 0 indicates that no dataset is in place. Note that if this accessor is used within a function, the number of series currently accessible may well fall short of that given by `$nvars`.

$obsdate

Output: series

Applicable when the current dataset is time-series with annual, quarterly, monthly or decennial frequency, or is dated daily or weekly, or when the dataset is a panel with time-series information set appropriately (see the setobs command). The returned series holds 8-digit numbers on the pattern YYYYMMDD (ISO 8601 "basic" date format), which correspond to the day of the observation, or the first day of the observation period in case of a time-series frequency less than daily.

Such a series can be helpful when using the join command.

$obsmajor

Output: series

Returns a series holding the "major" or low-frequency component of each observation. This means the year for annual, quarterly or monthly time series; the day for hourly data; or the individual in the case of panel data. If the data are cross-sectional the series returned is just a 1-based index of the observations.

See also $obsminor, $obsmicro.

$obsmicro

Output: series

Applicable when the observations in the current dataset have a major:minor:micro structure, as in dated daily time series (year:month:day). Returns a series holding the micro or highest-frequency component of each observation (for example, the day).

See also $obsmajor, $obsminor.

$obsminor

Output: series

Applicable when the observations in the current dataset have a major:minor structure, as in quarterly time series (year:quarter), monthly time series (year:month), hourly data (day:hour) and panel data (individual:period). Returns a series holding the minor or high-frequency component of each observation (for example, the month).

In the case of dated daily data, `$obsminor` gets the month of each observation.

See also $obsmajor, $obsmicro.

$panelpd

Output: integer

Specific to panel data, returns the time-series periodicity (e.g. 4 for quarterly data). If the periodicity is not set in the active panel dataset, returns 1 in analogy to $pd for cross-sectional or undated data. If the dataset is not a panel NA is returned.

See also $pd, $datatype, setobs.

$parnames

Output: array of strings

Following estimation of a single-equation model, returns an array of strings holding the names of the model's parameters. The number of names matches the number of elements in the $coeff vector.

For models specified via a list of regressors the result will be the same as that of

```
varnames($xlist)
```

(see varnames), but `$parnames` is more general; it also works for models with no regressor list (nls, mle, gmm).

$pd

Output: integer

Returns the frequency or periodicity of the data (e.g. 4 for quarterly data). In the case of panel data the value returned is the total time-series length.

See also $panelpd.

$pi

Output: scalar

Returns the value of π in double precision.

$pkgdir

Output: string

A special facility for use by authors of function packages. Returns an empty string unless a packaged function is executing, in which case it returns the full (platform dependent) path under which the package is installed. For instance the return value might be

```
/usr/share/gretl/functions/foo
```

if that's the directory in which `foo.gfn` is located. This enables package writers to access resources such as matrix files that they have included in their package.

$pmanteau

Output: matrix

Available after estimation of a vector autoregression. Returns a row vector holding the results of the multivariate portmanteau test for autocorrelation of the residuals, as discussed on pages 21–22 in Johansen (1995). The elements are, in order, the Ljung–Box statistic, the maximum lag considered, the degrees of freedom for the (chi-square) test, and the p-value of the test.

$pvalue

Output: scalar or matrix

Returns the p-value of the test statistic that was generated by the last explicit hypothesis-testing command, if any (for example, `chow`). See chapter 10 of the *Gretl User's Guide* for details.

In most cases the return value is a scalar but sometimes it is a matrix (for example, the trace and lambda-max p-values from the Johansen cointegration test); in that case the values in the matrix are laid out in the same pattern as the printed results.

See also $test.

$qlrbreak

Output: scalar

Must follow an invocation of the qlrtest command (the QLR test for a structural break at an unknown point). The value returned is the 1-based index of the observation at which the test statistic is maximized.

$result

Output: matrix or bundle

Provides stored information following certain commands that do not have specific accessors. The commands in question include bds, bkw, corr, fractint, freq, hurst, leverage, summary, vif and xtab (in which cases the result is a matrix), plus pkg (which optionally stores a bundle result).

$rho

Output: scalar
Argument: *n* (scalar, optional)

Without arguments, returns the first-order autoregressive coefficient for the residuals of the last model. After estimating a model via the `ar` command, the syntax `$rho(n)` returns the corresponding estimate of $\rho(n)$.

$rlnl

Output: scalar

Following estimation of a restricted VECM, returns the log-likelihood for the restricted model. See also $lnl, and chapter 33 of the *Gretl User's Guide* for examples of usage.

$rsq

Output: scalar

Returns the unadjusted R^2 from the last estimated model, if available. Usually this will be the regular (centered) R^2 but if the specification contains no constant (and no set of regressors that "add up to" a constant) it will be the uncentered version. In that case the centered version can be accessed as `$model.centered_R2`.

$sample

Output: series

Must follow estimation of a single-equation model. Returns a dummy series with value 1 for observations used in estimation, 0 for observations within the currently defined sample range but not used (presumably because of missing values), and NA for observations outside of the current range.

If you wish to compute statistics based on the sample that was used for a given model, you can do, for example:

```
ols y 0 xlist
series sdum = $sample
smpl sdum --dummy
```

$sargan

Output: row vector

Must follow a `tsls` command. Returns a 1×3 vector, containing the value of the Sargan overidentification test statistic, the corresponding degrees of freedom and p-value, in that order. If the model is exactly identified, the statistic is unavailable, and trying to access it provokes an error.

$seed

Output: scalar

Returns the value with which gretl's random number generator was seeded. If you set the seed yourself there's no need to use this accessor, but it may be of interest if the seed was set automatically (based on the time that execution of the program started).

$sigma

Output: scalar or matrix

Requires that a model has been estimated. If the last model was a single equation, returns the (scalar) Standard Error of the Regression (or in other words, the standard deviation of the residuals, with an appropriate degrees of freedom correction). If the last model was a system of equations, returns the cross-equation covariance matrix of the residuals.

$stderr

Output: matrix or scalar

Argument: *s* (name of coefficient, optional)

With no arguments, `$stderr` returns a column vector containing the standard error of the coefficients for the last model. With the optional string argument it returns a scalar, namely the standard error of the parameter named *s*.

If the "model" in question is actually a system, the result depends on the characteristics of the system: for VARs and VECMs the value returned is a matrix with one column per equation, otherwise it is a column vector containing the coefficients from the first equation followed by those from the second equation, and so on.

See also $coeff, $vcv.

$stopwatch

Output: scalar

Must be preceded by `set stopwatch`, which activates the measurement of CPU time. The first use of this accessor yields the seconds of CPU time that have elapsed since the `set stopwatch` command. At each access the clock is reset, so subsequent uses of `$stopwatch` yield the seconds of CPU time since the previous access.

When a user-defined function is executing, the `set stopwatch` command and `$stopwatch` accessor are specific to that function—that is, timing within a function does not disrupt any "global" timing that may be going on in the main script.

$sysA

Output: matrix

Must follow estimation of a simultaneous equations system. Returns the matrix of coefficients on the lagged endogenous variables, if any, in the structural form of the system. See the system command.

$sysB

Output: matrix

Must follow estimation of a simultaneous equations system. Returns the matrix of coefficients on the exogenous variables in the structural form of the system. See the system command.

$sysGamma

Output: matrix

Must follow estimation of a simultaneous equations system. Returns the matrix of coefficients on the contemporaneous endogenous variables in the structural form of the system. See the system command.

$sysinfo

Output: bundle

Returns a bundle containing information on the capabilities of the gretl build and the system on which gretl is running. The members of the bundle are as follows:

- `gui_mode`: integer, equals 1 if libgretl is being called by the GUI program, otherwise 0.
- `mpi`: integer, equals 1 if the system supports MPI (Message Passing Interface), otherwise 0.
- `omp`: integer, equals 1 if gretl is built with support for Open MP, otherwise 0.
- `ncores`: integer, the number of physical processor cores available.
- `nproc`: integer, the number of processors available, which will be greater than `ncores` if hyper-threading is enabled.

- mpimax: integer, the maximum number of MPI processes that can be run in parallel. This is zero if MPI is not supported, otherwise it equals the local nproc value unless an MPI hosts file has been specified, in which case it is the sum of the number of processors or "slots" across all the machines referenced in that file.
- wordlen: integer, either 32 or 64 for 32- and 64-bit systems respectively.
- os: string representing the operating system, either linux, macos, windows or other. Note that versions of gretl prior to 2021e gave the string osx for the Mac operating system; a version-independent test for Mac is therefore instring($sysinfo.os, "os")
- hostname: the name of the host machine on which the current gretl process is running (with a fallback of localhost in case the name cannot be determined).
- mem: a 2-vector holding total physical memory and free or available memory, expressed in MB. This information may not be available on all systems but should be on Windows, macOS and Linux.
- blas: string identifying the supplier of the BLAS (Basic Linear Algebra Subprograms) library in use by gretl.
- blas_version: string identifying the version number of the blas library in use.
- blascore: (if applicable) a string identifying the CPU type for which the current blas library is optimized.
- compiler: a string identifying the compiler used when building libgretl.
- cpuid: a string identifying the vendor and model of the CPU on which libgretl is running.
- gnuplot: a string identifying the version of gnuplot available to gretl for plotting, in the form of three dot-separated numbers giving major version, minor version and patchlevel.
- foreign: a sub-bundle containing 0/1 indicators for the presence on the host system of each of the "foreign" programs supported by gretl, under the keys julia, octave, ox, python, Rbin, Rlib and stata. The two keys pertaining to R represent the R executable and shared library, respectively.

Note that individual elements in the bundle can be accessed using "dot" notation without any need to copy the whole bundle under a user-specified name. For example,

```
if $sysinfo.os == "linux"
    # do something linux-specific
endif
```

$system

Output: bundle

Must follow estimation of a system of equations via one of the commands system, var or vecm; returns a bundle containing many items of data pertaining to the system. All the relevant regular system accessors are included: these are referenced by keys that are the same as the regular accessor names, minus the leading dollar sign. So for example the residuals appear under the key uhat and the coefficients under coeff. (Exceptions are the keys A, B, and Gamma, which correspond to the regular dollar accessors sysA, sysB, and sysGamma.) The keys for additional information should hopefully be fairly self-explanatory. To see what's available you can get a copy of the bundle and print its content, as in

```
var 4 y1 y2 y2
bundle b = $system
print b
```

A bundle obtained in this way can be passed as the final, optional argument to the functions fevd and irf.

$T

Output: integer

Returns the number of observations used in estimating the last model.

$t1

Output: integer

Returns the 1-based index of the first observation in the currently selected sample.

$t2

Output: integer

Returns the 1-based index of the last observation in the currently selected sample.

$test

Output: scalar or matrix

Returns the value of the test statistic that was generated by the last explicit hypothesis-testing command, if any (e.g. chow). See chapter 10 of the *Gretl User's Guide* for details.

In most cases the return value is a scalar but sometimes it is a matrix (for example, the trace and lambda-max statistics from the Johansen cointegration test); in that case the values in the matrix are laid out in the same pattern as the printed results.

See also $pvalue.

$time

Output: series

For time-series or panel data, creates a 1-based index of the time period. In the panel case the sequence of values repeats for each cross-sectional unit.

The command "`genr time`" is an alternative, with the difference that the `genr` variant automatically creates a series called `time` while the naming of the series is up to the caller when using `$time`, as in

```
series trend = $time
```

This accessor is not available for cross-sectional data.

$tmax

Output: integer

Returns the maximum legal setting for the end of the sample range via the smpl command. In most cases this will equal the number of observations in the dataset but within a hansl function the

$tmax value may be smaller, since in general data access within functions is limited to the sample range set by the caller.

Note that $tmax does not in general equal $nobs, which gives the number of observations in the current sample range.

$trsq

Output: scalar

Returns TR^2 (sample size times R-squared) from the last model, if available.

$uhat

Output: series

Returns the residuals from the last model. This may have different meanings for different estimators. For example, after an ARMA estimation $uhat will contain the one-step-ahead forecast error; after a probit model, it will contain the generalized residuals.

If the "model" in question is actually a system (a VAR or VECM, or system of simultaneous equations), $uhat retrieves the matrix of residuals, one column per equation.

$unit

Output: series

Valid for panel datasets only. Returns a series with value 1 for all observations on the first unit or group, 2 for observations on the second unit, and so on.

$vcv

Output: matrix or scalar
Arguments: *s1* (name of coefficient, optional)
s2 (name of coefficient, optional)

With no arguments, $vcv returns a square matrix containing the estimated covariance matrix for the coefficients of the last model. If the last model was a single equation, then you may supply the names of two parameters in parentheses to retrieve the estimated covariance between the parameters named *s1* and *s2*. See also $coeff, $stderr.

This accessor is not available for VARs or VECMs; for models of that sort see $sigma and $xtxinv.

$vecGamma

Output: matrix

Must follow the estimation of a VECM; returns a matrix in which the Gamma matrices (coefficients on the lagged differences of the cointegrated variables) are stacked side by side. Each row represents an equation; for a VECM of lag order p there are $p - 1$ sub-matrices.

$version

Output: scalar

Returns an integer value that codes for the program version. The current gretl version string takes the form of a 4-digit year followed by a letter from a to j representing the sequence of releases within the year (for example, 2015d). The return value from this accessor is formed as 10 times the year plus the zero-based lexical order of the letter, so 2015d translates to 20153.

Prior to gretl 2015d, version identifiers took the form x.y.z (three integers separated by dots), and in that case the accessor value was calculated as `10000*x + 100*y + z`, so that for example 1.10.2 (the last release under the old scheme) translates as 11002. Numerical order of `$version` values is therefore preserved across the change in versioning scheme.

$vma

Output: matrix

Must follow the estimation of a VAR or a VECM; returns a matrix containing the VMA representation up to the order specified via the `set horizon` command. See chapter 32 of the *Gretl User's Guide* for details.

$windows

Output: integer

Returns 1 if gretl is running on MS Windows, otherwise 0. By conditioning on the value of this variable you can write shell calls that are portable across different operating systems.

Also see the shell command.

$xlist

Output: list

If the last model was a single equation, returns the list of regressors. If the last model was a system of equations, returns the "global" list of exogenous variables (in the same order in which they appear in $sysB). If the last model was a VAR, returns the list of exogenous regressors, if any, except for standard deterministic terms (constant, trend, seasonals).

$xtxinv

Output: matrix

Following estimation of a VAR or VECM (only), returns $X'X^{-1}$, where X is the common matrix of regressors used in each of the equations. While this accessor is available for a VECM estimated with a restriction imposed on α (the "loadings" matrix), it should be borne in mind that in that case not all coefficients of the regressors are freely varying.

$yhat

Output: series

Returns the fitted values from the last regression.

$ylist

Output: list

If the last model estimated was a VAR, VECM or simultaneous system, returns the associated list of endogenous variables. If the last model was a single equation, this accessor gives a list with a single element, the dependent variable. In the special case of the biprobit model the list contains two elements.

2.3 Built-in strings

$dotdir

Output: string

Yields the full path of the directory gretl uses for temporary files. To use it in string-substitution mode, prepend the at-sign (`@dotdir`).

$gnuplot

Output: string

Yields the path to the `gnuplot` executable. To use it in string-substitution mode, prepend the at-sign (`@gnuplot`).

$gretldir

Output: string

Yields the full path of the gretl installation directory. To use it in string-substitution mode, prepend the at-sign (`@gretldir`).

$lang

Output: string

Returns a string representing the national language in force currently, if this can be determined. The string is composed of a two-letter ISO 639-1 language code (for example, `en` for English, `jp` for Japanese, `el` for Greek) followed by an underscore plus a two-letter ISO 3166-1 country code. Thus for example Portuguese in Portugal gives `pt_PT` while Portuguese in Brazil gives `pt_BR`.

If the national language cannot be determined, the string "`unknown`" is returned.

$seats

Output: string

Yields the path to the `seats` executable, which accompanies `tramo`. To use it in string-substitution mode, prepend the at-sign (`@seats`).

$tramo

Output: string

Yields the path to the `tramo` executable. To use it in string-substitution mode, prepend the at-sign (`@tramo`).

$tramodir

Output: string

Yields the path string of the `tramo` data directory. To use it in string-substitution mode, prepend the at-sign (`@tramodir`).

$workdir

Output: string

Yields the path which gretl reads from and writes to by default. A fuller discussion is provided in the Command Reference under workdir. Note that this string can be set by the user via the set command. To use it in string-substitution mode, prepend the at-sign (`@workdir`).

$x12a

Output: string

Yields the path to the `x-12-arima` (or `x-13arima`) executable. To use it in string-substitution mode, prepend the at-sign (`@x12a`).

$x12adir

Output: string

Yields the path of the `x-12-arima` (or `x-13arima`) data directory. To use it in string-substitution mode, prepend the at-sign (`@x12adir`).

2.4 Functions proper

abs

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the absolute value of *x*.

acos

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the arc cosine of *x*, that is, the value whose cosine is *x*. The result is in radians; the input should be in the range −1 to 1.

acosh

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the inverse hyperbolic cosine of *x* (positive solution). *x* should be greater than 1; otherwise, NA is returned. See also cosh.

aggregate

Output: matrix
Arguments: *x* (series, list or matrix)
byvar (series, list or matrix)
funcname (string, optional)

Most of the following assumes that the first two arguments to this function take the form of series or lists, but see "Matrix input" below for alternative usage.

In the simplest usage *x* is set to `null`, *byvar* is a single series and the third argument is omitted or set to `null`. The return value is then a matrix with two columns holding, respectively, the distinct values of *byvar* sorted in ascending order, and the count of observations at which *byvar* takes on each of these values. For example,

```
open data4-1
eval aggregate(null, bedrms)
```

will show that the series bedrms has values 3 (with count 5) and 4 (with count 9).

More generally, if *byvar* is a list with n members then the first n columns of the returned matrix hold the combinations of the distinct values of each of the n series, and the count column holds the number of observations at which each combination is realized. (The count column can always be found at the position nelem(byvar)+1).

Specifying an aggregation function

If the third argument is given then x must not be null, and the rightmost m columns hold the values of the statistic specified by *funcname* for each of the variables in x. (So m is equal to 1 if x is a single series and equal to nelem(x) if x is a list.) The specified statistic is calculated on the sub-samples defined by the combinations in *byvar* (in ascending order); these combinations are shown in the first n column(s) of the returned matrix.

So, if both x and *byvar* are individual series, the return value is a matrix with three columns holding the distinct values of *byvar* sorted in ascending order; the count of observations at which *byvar* takes on each of these values; and the values of the statistic specified by *funcname* calculated on series x, using just those observations at which *byvar* takes on the value given in the first column.

The following values of *funcname* are supported "natively": sum, sumall, mean, sd, var, sst, skewness, kurtosis, min, max, median, nobs, gini, isconst and isdummy. Each of these functions takes a series argument and returns a scalar value, and in that sense can be said to "aggregate" the series in some way. If none of these built-in functions does what you need, you can give the name of a user-defined function as the aggregator. Like the built-ins, such a function must take a single series argument and return a scalar value.

Note that although a count of cases is provided automatically the nobs function is not redundant as an aggregator, since it gives the number of valid (non-missing) observations on x at each *byvar* combination.

Some examples

First, suppose that region represents a coding of geographical region using integer values 1 to n, and income represents household income. Then the following would produce an $n \times 3$ matrix holding the region codes, the count of observations in each region, and mean household income for each of the regions:

```
matrix m = aggregate(income, region, mean)
```

For an example using lists, let gender be a male/female dummy variable, let race be a categorical variable with three values, and consider the following:

```
list BY = gender race
list X = income age
matrix m = aggregate(X, BY, sd)
```

The aggregate call here will produce a 6×5 matrix. The first two columns hold the 6 distinct combinations of gender and race values; the middle column holds the count for each of these combinations; and the rightmost two columns contain the sample standard deviations of income and age.

If *byvar* is a list, some combinations of the *byvar* values may not be present in the data (giving a count of zero). In that case the value of the statistics for *x* are recorded as NaN (not a number). To cut out such cases you can use the selifr function to select only those rows that have a non-zero count. The column to test is one place to the right of the number of *byvar* variables, so we can do:

```
matrix m = aggregate(X, BY, sd)
scalar c = nelem(BY)
m = selifr(m, m[,c+1])
```

Matrix input

Instead of series or lists, *x* and *byvar* may be given in matrix form. However, if both arguments are provided they must match in type (you cannot give a series or list for one argument and a matrix for the other) and two matrix arguments must have the same number of rows. In this context matrix columns are treated as if they were series, so the aggregation function must follow the pattern described above, taking a series argument and returning a scalar.

argname

Output:	string
Arguments:	*s* (string)
	default (string, optional)

For *s* the name of a parameter to a user-defined function, returns the name of the corresponding argument, if the argument had a name at the caller level. If the argument was anonymous, an empty string is returned unless the optional *default* argument is provided, in which case its value is used as a fallback.

array

Output:	see below
Argument:	*n* (integer)

The basic "constructor" function for a new array variable. In using this function you must specify a type (in plural form) for the array: strings, matrices, bundles, lists or arrays. The return value is an array of the specified type with *n* elements, each of which is initialized as "empty" (e.g. zero-length string, null matrix). Examples of usage:

```
strings S = array(5)
matrices M = array(3)
```

See also defarray.

asin

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the arc sine of *x*, that is, the value whose sine is *x*. The result is in radians; the input should be in the range −1 to 1.

asinh

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the inverse hyperbolic sine of *x*. See also sinh.

asort

Output: scalar
Arguments: *a* (array)
fname (string)

Performs an in-place sort of the elements of *a*, using a comparator function specified by the caller under the control of the quicksort routine.

The argument *a* can be of any of the types supported for a gretl array, namely `strings`, `matrices`, `bundles`, `lists` or `arrays`. The *fname* argument must be the name of a function which takes two `const` arguments, whose type matches that of the elements of *a*. This function must return an integer value on the following pattern: 0 if the two arguments have the same sort order, negative if the first argument sorts before the second, or positive if the second sorts before the first. (The exact values do not matter.)

For example, suppose one wants to sort an array of bundles, each of which contains a scalar named `crit`, by increasing value of `crit`. Then the following function would be suitable for passing to `asort`:

```
function scalar my_bsort (const bundle b1, const bundle b2)
   return sgn(b1.crit - b2.crit)
end function
```

If you want to preserve the unsorted array, make a copy of it before passing it to `asort`. The return value from this function is a nominal 0 on success.

See also sort for simple sorting of an array of strings.

assert

Output: scalar
Argument: *expr* (scalar)

This function is intended for testing or debugging of hansl code. The argument should be an expression which evaluates to a scalar. The return value is 1 if *expr* evaluates to a non-zero value (boolean "true", or "success") or 0 if it evaluates to zero (boolean "false", or "failure").

By default there are no consequences of a call to `assert` failing other than the return value being zero. However, the set command can be used to make failure of an assertion more consequential. There are three levels:

```
# print a warning message but continue execution
set assert warn
# print an error message and stop script execution
set assert stop
# print a message to stderr and abort the program
set assert fatal
```

In most cases `stop` is sufficient to terminate a script but in certain special cases (such as within a function called from a command block such as mle) it may be necessary to use the `fatal` setting

to get a clear indication of the failing assertion. Note, however, that in this case the message will go to standard error output.

The default behavior can be restored via

```
set assert off
```

By way of a simple example, if at a certain point in a hansl script a scalar x ought to be non-negative, the following will flag an error if that is not the case:

```
set assert stop
assert(x >= 0)
```

atan

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the arc tangent of *x*, that is, the value whose tangent is *x*. The result is in radians.

See also tan, atan2.

atan2

Output: same type as input
Arguments: *y* (scalar, series or matrix)
x (scalar, series or matrix)

Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result. The return value is in radians, in the range $[-\pi, \pi]$.

If the two arguments differ in type, the type of the result is the "higher" of the two, where the ordering is matrix > series > scalar. For example, if *y* is a scalar and *x* an n-vector (or vice versa) the result is an n-vector. Note that matrix arguments must be vectors, and if neither argument is a scalar the two arguments must be of the same length.

See also tan, tanh.

atanh

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the inverse hyperbolic tangent of *x*. See also tanh.

atof

Output: scalar
Argument: *s* (string)

Closely related to the C library function of the same name. Returns the result of converting the string *s* (or the leading portion thereof, after discarding any initial white space) to a floating-point number. Unlike `atof` in C, however, the decimal character is always assumed (for reasons of portability) to be "`.`". Any characters that follow the portion of *s* that converts to a floating-point number under this assumption are ignored.

If none of *s* (following any discarded white space) is convertible under the stated assumption, NA is returned.

```
# examples
x = atof("1.234") # gives x = 1.234
x = atof("1,234") # gives x = 1
x = atof("1.2y")  # gives x = 1.2
x = atof("y")     # gives x = NA
x = atof(",234")  # gives x = NA
```

See also sscanf for more flexible string to numeric conversion.

bcheck

Output:	scalar
Arguments:	*target* (reference to bundle)
	input (bundle, optional)
	required-keys (array of strings, optional)

Primarily intended for use by writers of function packages. Here is the context in which `bcheck` may be useful: you have a function which accepts a bundle argument whereby the caller can make various choices. Some elements of the bundle may have default values—so the caller is not obliged to make an explicit choice—while other elements may be required. You want to determine whether the argument you get is valid. The main text below assumes that an *input* bundle is supplied by the caller of your function, but see the section headed "No input bundle" for the contrary case.

To use `bcheck` you construct a template bundle containing all the supported keys, with values that exemplify the type associated with each key, and pass this in pointer form as *target*. For the second argument, *input*, pass the bundle you get from the caller. This function then checks the following:

- Does *input* contain any keys not present in *target*? If so, `bcheck` returns a non-zero value, indicating that *input* is erroneous. (Most likely, the key in question is misspelled.)
- Does *input* contain under any given key an object whose type does not match that in *target*? If so, a non-zero value is returned.
- If some elements in *target* require input from the caller (so the value you supply is not a default value, just a placeholder to indicate the required type), you should supply a third argument to `bcheck`: an array of strings holding the keys for which input is not optional. Then the return value will be non-zero if any required elements are missing from *input*.

In addition to the above you may wish to impose lower and/or upper bounds on the value of one or more scalar members of the bundle argument. If so, add a bundle named `bounds` to your template bundle. Each member of this secondary bundle should have a *key* that identifies a member of the template bundle; its *value* should be a 2-vector holding lower and upper limits. Put `NA` in place of one of the limits if it is unbounded. So, for example, the following code will check that if `x1` is given in the caller's input it is between 1 and 5, and if `x2` is given it is non-negative:

```
template.bounds = _(x1={1,5}, x2={0,NA})
```

If no errors are detected on any of these points, values supplied in *input* are copied to *target* (defaults being replaced by valid selections on the caller's part). If errors are found a message will be printed indicating what is wrong with *input*.

To give a simple example, suppose your function's argument bundle supports a matrix `X` (required), a non-negative scalar `z` with default value 0, and a string `s` with default value "`display`". Then the following code fragment would be suitable for checking a bundle named `uservals` supplied by the caller:

```
bundle target = _(X={}, z=0, s="display")
target.bounds = _(z={0,NA})
strings req = defarray("X")
err = bcheck(&target, uservals, req)
if err
   # react appropriately
else
   # proceed, using the values in target
endif
```

No input bundle

If the *input* bundle is not supplied to bcheck, it behaves as follows. If the *required-keys* argument is not given, it returns zero (since none of the error conditions mentioned above can occur), and *target* is not modified. Otherwise it returns non-zero since it's clear that one or more specifications must be missing. This means that it's safe to pass a null *input* to bcheck.

bessel

Output:	same type as input
Arguments:	*type* (character)
	v (scalar)
	x (scalar, series or matrix)

Computes one of the Bessel function variants for order *v* and argument *x*. The return value is of the same type as *x*. The specific function is selected by the first argument, which must be J, Y, I, or K. A good discussion of the Bessel functions can be found on Wikipedia; here we give a brief account.

case J: Bessel function of the first kind. Resembles a damped sine wave. Defined for real *v* and *x*, but if *x* is negative then *v* must be an integer.

case Y: Bessel function of the second kind. Defined for real *v* and *x* but has a singularity at $x = 0$.

case I: Modified Bessel function of the first kind. An exponentially growing function. Acceptable arguments are as for case J.

case K: Modified Bessel function of the second kind. An exponentially decaying function. Diverges at $x = 0$ and is not defined for negative *x*. Symmetric around $v = 0$.

BFGSmax

Output:	scalar
Arguments:	*&b* (reference to matrix)
	f (function call)
	g (function call, optional)

Numerical maximization via the method of Broyden, Fletcher, Goldfarb and Shanno. On input the vector *b* should hold the initial values of a set of parameters, and the argument *f* should specify a call to a function that calculates the (scalar) criterion to be maximized, given the current parameter values and any other relevant data. If the object is in fact minimization, this function should return the negative of the criterion. On successful completion, BFGSmax returns the maximized value of the criterion, and *b* holds the parameter values which produce the maximum.

The optional third argument provides a means of supplying analytical derivatives (otherwise the gradient is computed numerically). The gradient function call *g* must have as its first argument a predefined matrix that is of the correct size to contain the gradient, given in pointer form. It also

must take the parameter vector as an argument (in pointer form or otherwise). Other arguments are optional.

For more details and examples see chapter 37 of the *Gretl User's Guide*. See also BFGScmax, NRmax, fdjac, simann.

BFGSmin

Output: scalar

An alias for BFGSmax; if called under this name the function acts as a minimizer.

BFGScmax

Output:	scalar
Arguments:	*&b* (reference to matrix)
	bounds (matrix)
	f (function call)
	g (function call, optional)

Constrained numerical maximization using L-BFGS-B (limited memory BFGS, see Byrd *et al.* (1995)). On input the vector *b* should hold the initial values of a set of parameters, *bounds* should hold bounds on the parameter values (see below), and *f* should specify a call to a function that calculates the (scalar) criterion to be maximized, given the current parameter values and any other relevant data. If the object is in fact minimization, this function should return the negative of the criterion. On successful completion, `BFGScmax` returns the maximized value of the criterion, subject to the constraints in *bounds*, and *b* holds the parameter values which produce the maximum.

Bounds on parameters

The *bounds* matrix must have 3 columns and as many rows as there are constrained elements in the parameter vector. The first element on a given row is the (1-based) index of the constrained parameter; the second and third are the lower and upper bounds, respectively. The values `-$huge` and `$huge` should be used to indicate that the parameter is unconstrained downward or upward, respectively. For example, the following is the way to specify that the second element of the parameter vector must be non-negative:

```
matrix bounds = {2, 0, $huge}
```

Analytical derivatives

The optional fourth argument provides a means of supplying analytical derivatives (otherwise the gradient is computed numerically). The gradient function call *g* must have as its first argument a predefined matrix that is of the correct size to contain the gradient, given in pointer form. It also must take the parameter vector as an argument (in pointer form or otherwise). Other arguments are optional.

For more details and examples see chapter 37 of the *Gretl User's Guide*. See also BFGSmax, NRmax, fdjac, simann.

BFGScmin

Output: scalar

An alias for BFGScmax; if called under this name the function acts as a minimizer.

bin2dec

Output:	matrix
Argument:	*B* (matrix)

Given a matrix *B* containing only zeros and ones, this function interprets each row as the binary representation of a 32-bit unsigned integer, and returns a column vector with the decimal representation of those integers. The argument cannot have more than 32 columns otherwise an error is flagged.

Note that the least significant bit comes in the first column. So column 1 corresponds to 2^0, column 2 to 2^1, and so on. For example, the expression

```
scalar x = bin2dec({1,0,1})
```

stores the value 5 into x.

The dec2bin function performs the inverse transformation.

bincoeff

Output:	same type as input
Arguments:	*n* (scalar, series or matrix)
	k (scalar, series or matrix)

Returns the binomial coefficient, that is the number of ways in which k items can be chosen from n items without repetition, irrespective of ordering. This is also equal to the coefficient of the $(k+1)$-th term in the polynomial expansion of the binomial power $(1+x)^n$.

For integer arguments the result is $n!/(k!(n-k)!)$ but the function also accepts noninteger arguments, and the formula above generalizes to $\frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$.

When $k > n$ or $k < 0$ no valid answer exists and an error is flagged.

If the two arguments differ in type, the type of the result is the "higher" of the two, where the ordering is matrix > series > scalar. For example, if n is a scalar and k an r-vector (or vice versa) the result is an r-vector. Note that matrix arguments must be vectors, and if neither argument is a scalar the two arguments must be of the same length.

See also gammafun and lngamma.

binperms

Output:	matrix
Arguments:	*n* (integer)
	k (integer)

Binary permutations: returns a $p \times n$ matrix, each of whose rows holds a distinct arrangement of k ones and $n-k$ zeros (in lexicographic order). The maximum supported value of n is 64, n and k must be non-negative, and k must be no greater than n; otherwise an error is flagged. In case $n = k = 0$ an empty matrix is returned.

For example, with $n = 4$ and $k = 2$, the result is

```
0   0   1   1
0   1   0   1
0   1   1   0
1   0   0   1
1   0   1   0
```

```
1   1   0   0
```

Warning: the number of permutations, p, is a steeply increasing function of n and is greatest when k is about half of n. You may want to check in advance the size of the matrix that `binperms` will attempt to allocate. The bincoeff function returns p, and the size of the resulting matrix in megabytes can be calculated as

```
MB = 8 * n * bincoeff(n, k) / 10^6
```

For $n = 30$, this gives about 34 MB when $k = 25$, 7211 MB if $k = 20$, and 20758 MB if $k = 18$.

bkfilt

Output:	series
Arguments:	*y* (series)
	f1 (integer, optional)
	f2 (integer, optional)
	k (integer, optional)

Returns the result from application of the Baxter–King bandpass filter to the series *y*. The optional parameters *f1* and *f2* represent, respectively, the lower and upper bounds of the range of frequencies to extract, while *k* is the approximation order to be used.

If these arguments are not supplied then the default values depend on the periodicity of the dataset. For yearly data the defaults for *f1*, *f2* and *k* are 2, 8 and 3, respectively; for quarterly data, 6, 32 and 12; for monthly data, 18, 96 and 36. These values are chosen to match the most common choice among practitioners, that is to use this filter for extracting the "business cycle" frequency component; this, in turn, is commonly defined as being between 18 months and 8 years. The filter, per default choice, spans 3 years of data.

If *f2* is greater than or equal to the number of available observations, then the "low-pass" version of the filter will be run and the resulting series should be taken as an estimate of the trend component, rather than the cycle. See also bwfilt, hpfilt.

bkw

Output:	matrix
Arguments:	*V* (matrix)
	parnames (array of strings, optional)
	verbose (boolean, optional)

Computes BKW collinearity diagnostics (see Belsley *et al.* (1980)) given a covariance matrix of parameter estimates, *V*. The optional second argument, which can be an array of strings or a string containing comma-separated names, is used to label the columns showing the variance proportions; the number of names should match the dimension of *V*. After estimation of a model in gretl, suitable arguments can be obtained via the $vcv and $parnames accessors.

By default this function operates silently, just returning the BKW table as a matrix, but if a non-zero value is given for the third argument the table is printed along with some analysis.

There is also a command form of this facility, bkw, which automatically references the last model and requires no arguments.

boxcox

Output:	same type as input
Arguments:	*y* (series or matrix)
	d (scalar)

Returns the Box–Cox transformation with parameter *d* for the positive series *y* (or the columns of matrix *y*).

$$y_t^{(d)} = \begin{cases} \frac{y_t^d - 1}{d} & \text{if} \quad d \neq 0 \\ \log(y_t) & \text{if} \quad d = 0 \end{cases}$$

bread

Output:	bundle
Arguments:	*fname* (string)
	import (boolean, optional)

Reads a bundle from the file specified by the *fname* argument. By default the bundle is assumed to be represented in XML, and to be gzip-compressed if *fname* has extension `.gz`. But if the extension is `.json` or `.geojson` the content is assumed to be JSON.

In the XML case the file must contain a `gretl-bundle` element, which is used to store zero or more `bundled-item` elements. For example,

```
<?xml version="1.0" encoding="UTF-8"?>
<gretl-bundle name="temp">
<bundled-item key="s" type="string">moo</bundled-item>
<bundled-item key="x" type="scalar">3</bundled-item>
</gretl-bundle>
```

As you might expect, files suitable for reading via `bread` are generated by the companion function bwrite.

If the file name does not contain a full path specification, it will be looked for in several "likely" locations, beginning with the currently set workdir. However, if a non-zero value is given for the optional *import* argument, the input file is taken to be in the user's "dot" directory. In that case *fname* should be a plain file name, without any path component.

Should an error occur (such as the file being badly formatted or inaccessible), an error is returned via the $error accessor.

See also mread, bwrite.

brename

Output:	scalar
Arguments:	*B* (bundle)
	oldkey (string)
	newkey (string)

If the bundle *B* contains a member under the key *oldkey*, its key is changed to *newkey*, otherwise an error is flagged. Returns 0 on successful renaming.

Changing the key of a bundle member is not a common task but it can arise in the context of functions that work with bundles, and `brename` is an efficient tool for the job. Example:

```
# set up a bundle holding a big matrix
bundle b
b.X = mnormal(1000, 1000)
if 0
    # change the key manually
    Xcopy = b.X
    delete b.X
    b.Y = Xcopy
    delete Xcopy
else
    # versus: change it efficiently
    brename(b, "X", "Y")
endif
```

The first method requires that the big matrix be copied twice, out of the bundle then back into it under a different key; the efficient method changes the key directly.

bwfilt

Output:	series
Arguments:	*y* (series)
	n (integer)
	omega (scalar)

Returns the result from application of a low-pass Butterworth filter with order *n* and frequency cutoff *omega* to the series *y*. The cutoff is expressed in degrees and must be greater than 0 and less than 180. Smaller cutoff values restrict the pass-band to lower frequencies and hence produce a smoother trend. Higher values of *n* produce a sharper cutoff, at the cost of possible numerical instability.

Inspecting the periodogram of the target series is a useful preliminary when you wish to apply this function. See chapter 30 of the *Gretl User's Guide* for details. See also bkfilt, hpfilt.

bwrite

Output:	integer
Arguments:	*B* (bundle)
	fname (string)
	export (boolean, optional)

Writes the bundle *B* to file, serialized in XML or, if *fname* has extension `.json` or `.geojson`, as JSON. See bread for a description of the format when XML is used. If *fname* already exists, it will be overwritten. The nominal return value is 0 on successful completion; if writing fails an error is flagged.

The output file will be written in the currently set workdir, unless *fname* contains a full path specification. However, if a non-zero value is given for the *export* argument, the file will be written into the user's "dot" directory. In that case a plain file name, without any path component, should be given for the second argument.

In the case of XML output (only), the option of gzip compression is available; this is applied if *fname* has the extension `.gz`.

See also bread, mwrite.

carg

Output:	matrix
Argument:	*C* (complex matrix)

Returns an $m \times n$ real matrix holding the complex "argument" of each element of the $m \times n$ complex matrix *C*. The argument of the complex number $z = x + yi$ can also be computed as `atan2(y, x)`.

See also abs, cmod, atan2.

cdemean

Output:	matrix
Arguments:	*X* (matrix)
	standardize (boolean, optional)
	skip_na (boolean, optional)

Centers the columns of matrix *X* around their means. If the optional second argument has a non-zero value then in addition the centered values are divided by the column standard deviations (which are calculated using $n - 1$ as divisor, where n is the number of rows of *X*).

If a non-zero value is supplied for *skip_na* missing values are ignored, otherwise if a column of *X* contains any missing values the corresponding column in the output is all missing.

Note that stdize provides more flexible functionality.

cdf

Output:	same type as input
Arguments:	*d* (string)
	\dots {} (see below)
	x (scalar, series or matrix)
Examples:	`p1 = cdf(N, -2.5)`
	`p2 = cdf(X, 3, 5.67)`
	`p3 = cdf(D, 0.25, -1, 1)`

Cumulative distribution function calculator. Returns $P(X \leq x)$, where the distribution of *X* is determined by the string *d*. Between the arguments *d* and *x*, zero or more additional scalar arguments are required to specify the parameters of the distribution, as follows (but note that the normal distribution has its own convenience function, cnorm).

Distribution	*d*	*Arg* 2	*Arg* 3	*Arg* 4
Standard normal	`z`, `n` or `N`	–	–	–
Bivariate normal	`D`	ρ	–	–
Logistic	`lgt`	–	–	–
Student's t (central)	`t`	df	–	–
Chi square	`c`, `x` or `X`	df	–	–
Snedecor's F	`f` or `F`	df (num.)	df (den.)	–
Gamma	`g` or `G`	shape	scale	–
Binomial	`b` or `B`	probability	trials	–
Poisson	`p` or `P`	mean	–	–
Exponential	`exp`	scale	–	–
Weibull	`w` or `W`	shape	scale	–
Laplace	`l` or `L`	mean	scale	–
Generalized Error	`E`	shape	–	–
Non-central χ^2	`ncX`	df	non-centrality	–
Non-central F	`ncF`	df (num.)	df (den.)	non-centrality
Non-central t	`nct`	df	non-centrality	–

Note that most cases have aliases to help memorizing the codes. The bivariate normal case is special: the syntax is `x = cdf(D, rho, z1, z2)` where `rho` is the correlation between the variables `z1` and `z2`.

The parametrization gretl uses for the Gamma random variate implies that its density function can be written as

$$f(x;k,\theta) = \frac{x^{k-1}}{\theta^k}\frac{e^{-x/\theta}}{\Gamma(k)}$$

where $k > 0$ is the shape parameter and $\theta > 0$ is the scale parameter.

See also pdf, critical, invcdf, pvalue.

cdiv

Output: matrix
Arguments: *X* (matrix)
Y (matrix)

This is a legacy function, predating gretl's native support for complex matrices.

Complex division. The two arguments must have the same number of rows, n, and either one or two columns. The first column contains the real part and the second (if present) the imaginary part. The return value is an $n \times 2$ matrix or, if the result has no imaginary part, an n-vector. See also cmult.

cdummify

Output: list
Argument: *L* (list)

This function returns a list in which each series in *L* that has the "coded" attribute is replaced by a set of dummy variables representing each of its coded values, with the least value omitted. If *L* contains no coded series the return value will be identical to *L*.

The generated dummy variables, if any, are named on the pattern D*varname_vi* where *vi* is the i^{th} represented value of the coded variable. In case any values are negative, "m" is inserted before the

(absolute) value of *vi*.

For example, suppose *L* contains a coded series named C1 with values −9, −7, 0, 1 and 2. Then the generated dummies will be DC1_m7 (coding for C1 = −7), DC1_0 (coding for C1 = 0), and so on.

See also dummify, getinfo.

ceil

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Ceiling function: returns the smallest integer greater than or equal to *x*. See also floor, int.

cholesky

Output:	square matrix
Argument:	*A* (positive definite matrix)

Performs a Cholesky decomposition of *A*. If *A* is real it must be symmetric and positive definite; if so, the result is a lower-triangular matrix *L* which satisfies $A = LL'$. If *A* is complex it must be Hermitian and positive definite, and the result is a lower-triangular complex matrix such that $A = LL^{\mathrm{H}}$. Otherwise, the function will return an error.

For the real case, see also psdroot and Lsolve.

chowlin

Output:	matrix
Arguments:	*Y* (matrix)
	xfac (integer)
	X (matrix, optional)

We no longer recommend use of this function; please use tdisagg instead.

Expands the input data, *Y*, to a higher frequency, using the method of Chow and Lin (1971). It is assumed that the columns of *Y* represent data series; the returned matrix has as many columns as *Y* and *xfac* times as many rows. It is also assumed that each low-frequency value should be treated as the average of *xfac* high-frequency values.

The *xfac* value should be 3 for quarterly to monthly, 4 for annual to quarterly or 12 for annual to monthly. The optional third argument may be used to provide a matrix of covariates at the higher (target) frequency.

The regressors used by default are a constant and trend. If *X* is provided, its columns are used as additional regressors; it is an error if the number of rows in *X* does not equal *xfac* times the number of rows in *Y*.

cmod

Output:	matrix
Argument:	*C* (complex matrix)

Returns an $m \times n$ real matrix holding the complex modulus of each element of the $m \times n$ complex matrix *C*. The modulus of the complex number $z = x + yi$ equals the square root of $x^2 + y^2$.

See also abs, carg.

cmult

Output:	matrix
Arguments:	*X* (matrix)
	Y (matrix)

This is a legacy function, predating gretl's native support for complex matrices.

Complex multiplication. The two arguments must have the same number of rows, n, and either one or two columns. The first column contains the real part and the second (if present) the imaginary part. The return value is an $n \times 2$ matrix, or, if the result has no imaginary part, an n-vector. See also cdiv.

cnorm

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the cumulative distribution function for a standard normal. See also dnorm, qnorm.

cnumber

Output:	scalar
Argument:	*X* (matrix)

Returns the condition number of the $n \times k$ matrix X, as defined in Belsley *et al.* (1980). If the columns of X are mutually orthogonal the condition number of X is unity. Conversely, a large value of the condition number is an indicator of multicollinearity; "large" is often taken to mean 50 or greater (sometimes 30 or greater).

The steps in the calculation are: (1) form a matrix Z whose columns are the columns of X divided by their respective Euclidean norms; (2) form $Z'Z$ and obtain its eigenvalues; and (3) compute the square root of the ratio of the largest to the smallest eigenvalue.

See also rcond.

cnameget

Output:	string or array of strings
Arguments:	*M* (matrix)
	col (integer, optional)

If the *col* argument is given, retrieves the name for column *col* of matrix *M*. If *M* has no column names attached the value returned is an empty string; if *col* is out of bounds for the given matrix an error is flagged.

If no second argument is given, retrieves an array of strings holding the column names from *M*, or an empty array if *M* does not have column names attached.

Example:

```
matrix A = { 11, 23, 13 ; 54, 15, 46 }
cnameset(A, "Col_A Col_B Col_C")
string name = cnameget(A, 3)
print name
```

See also cnameset.

cnameset

Output:	scalar
Arguments:	*M* (matrix)
	S (array of strings or list)

Attaches names to the columns of the $T \times k$ matrix *M*. If *S* is a named list, the names are taken from the names of the listed series; the list must have k members. If *S* is an array of strings, it should contain k elements. A single string is also acceptable as the second argument; in that case it should contain k space-separated substrings. As a special case, passing an empty string as the second argument has the effect of removing any existing column names.

The nominal return value is 0 on successful completion; in case of failure an error is flagged. See also rnameset.

Example:

```
matrix M = {1, 2; 2, 1; 4, 1}
strings S = array(2)
S[1] = "Col1"
S[2] = "Col2"
cnameset(M, S)
print M
```

cols

Output:	integer
Argument:	*X* (matrix)

Returns the number of columns of *X*. See also mshape, rows, unvech, vec, vech.

commute

Output:	matrix
Arguments:	*A* (matrix)
	m (integer)
	n (integer, optional)
	post (integer, optional)
	add_id (integer, optional)

Returns the matrix *A* premultiplied by the commutation matrix $K_{m,n}$ (using an algorithm that is more efficient than explicit multiplication). Each column of *A* is assumed to come from a vec operation on a $m \times n$ matrix. In particular,

```
commute(vec(B), rows(B), cols(B))
```

gives vec(B'). In order to compute the commutation matrix proper, just apply the function to an appropriately sized identity matrix. For example:

```
K_32 = commute(I(6), 3, 2)
```

The optional argument *n* defaults to *m*. If the optional argument *post* is non-zero, then post-multiplication is performed instead of pre-multiplication; the optional Boolean switch *add_id* will premultiply *A* by $I + K_{m,n}$ instead of $K_{m,n}$.

complex

Output:	complex matrix
Arguments:	*A* (scalar or matrix)
	B (scalar or matrix, optional)

Returns a complex matrix, where A is taken to supply the real part and B the imaginary part. If A is $m \times n$ and B is a scalar the result is $m \times n$ with a constant imaginary part—and similarly in the converse case but with a constant real part. If both arguments are matrices they must be of the same dimensions. If the second argument is omitted the imaginary part defaults to zero. See also cswitch.

conj

Output:	complex matrix
Argument:	*C* (complex matrix)

Returns an $m \times n$ complex matrix holding the complex conjugate of each element of the $m \times n$ complex matrix C. The conjugate of the complex number $z = x + yi$ equals $x - yi$.

See also carg, abs.

contains

Output:	same type as input
Arguments:	*x* (scalar, series or matrix)
	S (matrix)

Provides a means of determining whether the numerical object x contains any of the elements of S, a matrix which plays the role of a set.

The return value is an object of the same size as x containing 1s in positions where a value of x matches any element of S and zeros elsewhere. For example, the code

```
matrix A = mshape(seq(1,9), 3, 3)
matrix C = contains(A, {1, 5, 9})
```

gives

```
A (3 x 3)

1   4   7
2   5   8
3   6   9

C (3 x 3)

1   0   0
0   1   0
0   0   1
```

This function may be particularly useful when x is a series that contains a fine-grained encoding for a qualitative characteristic, and you wish to reduce this to a smaller number of categories. You can pack into S a set of values to be consolidated, and obtain a dummy variable with value 1 for observations matching this set, 0 otherwise.

Since S serves as a set, for greatest efficiency it should be a vector with no repeated values, however an arbitrary matrix is accepted.

conv2d

Output:	matrix
Arguments:	*A* (matrix)
	B (matrix)

Computes the 2-dimensional convolution of the matrices A and B. If A is $r \times c$ and B is $m \times n$ then the returned matrix will have $r + m - 1$ rows and $c + n - 1$ columns.

The 2-D convolution of A and B is defined as

$$C_{i,j} = \sum_{k=1}^{r} \sum_{l=1}^{c} A_{k,l} B_{i-k+1,j-l+1},$$

where the summations include just those values of k and l for which the subscripts of B are within bounds.

See also fft, filter.

cquad

Output:	matrix
Argument:	*Z* (matrix)

Given an $m \times n$ complex matrix Z, returns an $m \times n$ real matrix holding the quadrance of the elements of Z. The quadrance of the complex number $z = a + bi$ is $a^2 + b^2$. It therefore equals the squared modulus of z and also equals z multiplied by its complex conjugate, but the direct calculation carried out by `cquad` is considerably faster than either of the alternative approaches.

corr

Output:	scalar
Arguments:	*y1* (series or vector)
	y2 (series or vector)

Computes the correlation coefficient between *y1* and *y2*. The arguments should be either two series, or two vectors of the same length. See also cov, mcov, mcorr, npcorr.

corresp

Output:	scalar
Arguments:	*a* (series or vector)
	b (series or vector)

On the basis of a cross-tabulation of *a* and *b*, returns an integer code indicating the sort of correspondence between the two variables, as follows.

- Code = 2: there's a 1-to-1 relationship.
- Code = 1: there's a 1-to-n relationship (*a* "nests" *b*, can be interpreted as a function of *b* in the mathematical sense).
- Code = −1: there's an n-to-1 relationship (*b* "nests" *a*, can be interpreted as a function of *a*).
- Code = 0: there's no relationship.

Note that these codes are based solely on the sample values of the two arguments. In case *b* is the square of *a*, for example, the result will differ depending on whether *a* contains some pairs of values that differ only by sign (code = −1), or not (code = 2).

One possible use case is to check whether two discrete series encode the same information. For example, the following:

```
open grunfeld.gdt
c = corresp($unit, firm)
```

gives `c` = 2, indicating that the series `firm` is in fact a unique identifier for the cross-sectional units in this panel dataset.

See also mxtab.

corrgm

Output:	matrix
Arguments:	*x* (series, matrix or list)
	p (integer)
	y (series or vector, optional)

If only the first two arguments are given, computes the correlogram for *x* for lags 1 to *p*. Let *k* represent the number of elements in *x* (1 if *x* is a series, the number of columns if *x* is a matrix, or the number of list-members if *x* is a list). The return value is a matrix with *p* rows and $2k$ columns, the first *k* columns holding the respective autocorrelations and the remainder the respective partial autocorrelations.

If a third argument is given, this function computes the cross-correlogram for each of the *k* elements in *x* and *y*, from lead *p* to lag *p*. The returned matrix has $2p + 1$ rows and *k* columns. If *x* is series or list and *y* is a vector, the vector must have just as many rows as there are observations in the current sample range.

cos

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the cosine of *x*. See also sin, tan, atan.

cosh

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the hyperbolic cosine of *x*.

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

See also acosh, sinh, tanh.

cov

Output:	scalar
Arguments:	*y1* (series or vector)
	y2 (series or vector)

Returns the covariance between *y1* and *y2*. The arguments should be either two series, or two vectors of the same length. See also corr, mcov, mcorr.

critical

Output:	same type as input
Arguments:	*c* (character)
	\dots {} (see below)
	p (scalar, series or matrix)
Examples:	`c1 = critical(t, 20, 0.025)`
	`c2 = critical(F, 4, 48, 0.05)`

Critical value calculator. Returns x such that $P(X > x) = p$, where the distribution X is determined by the character *c*. Between the arguments *c* and *p*, zero or more additional scalar arguments are required to specify the parameters of the distribution, as follows.

Distribution	*c*	*Arg* 2	*Arg* 3
Standard normal	`z`, `n` or `N`	–	–
Student's t (central)	`t`	degrees of freedom	–
Chi square	`c`, `x` or `X`	degrees of freedom	–
Snedecor's F	`f` or `F`	df (num.)	df (den.)
Binomial	`b` or `B`	p	n
Poisson	`p` or `P`	λ	–
Laplace	`l` or `L`	mean	scale
Standardized GED	`E`	shape	–

See also cdf, invcdf, pvalue.

cswitch

Output:	matrix
Arguments:	*A* (matrix)
	mode (scalar)

Reinterprets a real matrix as holding complex values or vice versa. The precise action depends on *mode* (which must have value 1, 2, 3 or 4) as follows:

mode 1: *A* must be a real matrix with an even number of columns. Returns a complex matrix with half as many columns, the odd-numbered columns of *A* supplying the real parts and the even-numbered columns the imaginary parts.

mode 2: Performs the inverse operation of mode 1. *A* must be a complex matrix and the return value is a real matrix with twice as many columns as *A*.

mode 3: *A* must be a real matrix with an even number of rows. Returns a complex matrix with half as many rows, the odd-numbered rows of *A* supplying the real parts and the even-numbered rows the imaginary parts.

mode 4: Performs the inverse operation of mode 3. *A* must be a complex matrix and the return value is a real matrix with twice as many rows as *A*.

See also complex.

ctrans

Output:	complex matrix
Argument:	*C* (complex matrix)

Returns an $n \times m$ complex matrix holding the conjugate transpose of the $m \times n$ complex matrix C. The ' (prime) operator also performs conjugate transposition for complex matrices. The transp function can be used on complex matrices but it performs "straight" transposition (not conjugated).

cum

Output:	same type as input
Argument:	x (series or matrix)

Cumulates x. When x is a series, produces a series $y_t = \sum_{s=m}^{t} x_s$; the starting point of the summation, m, is the first non-missing observation of the currently selected sample. If any missing values are encountered in x, subsequent values of y will be set to missing. When x is a matrix, its elements are cumulated by columns.

In the case of panel data cumulation is in the time dimension, starting anew for each panel unit.

If you want cumulation to ignore missing values (that is, to treat them as if they were zeros), you can apply misszero to the argument, as in

```
series cx = cum(misszero(x))
```

See also diff.

curl

Output:	integer
Argument:	*&b* (reference to bundle)

Provides a somewhat flexible means of obtaining a text buffer containing data from an internet server, using libcurl. On input the bundle *b* must contain a string named URL which gives the full address of the resource on the target host. Other optional elements are as follows.

- "header": a string specifying an HTTP header to be sent to the host.
- "postdata": a string holding data to be sent to the host.

The header and postdata fields are intended for use with an HTTP POST request; if postdata is present the POST method is implicit, otherwise the GET method is implicit. (But note that for straightforward GET requests readfile offers a simpler interface.)

One other optional bundle element is recognized: if a scalar named include is present and has a non-zero value, this is taken as a request to include the header received from the host with the output body.

On completion of the request, the text received from the server is added to the bundle under the key "output".

If an error occurs in formulating the request (for example there's no URL on input) the function fails, otherwise it returns 0 if the request succeeds or non-zero if it fails, in which case the error message from the curl library is added to the bundle under the key "errmsg". Note, however, that "success" in this sense does not necessarily mean you got the data you wanted; all it means is that some response was received from the server. You must check the content of the output buffer (which may in fact be a message such as "Page not found").

Here is an example of use: downloading some data from the US Bureau of Labor Statistics site, which requires sending a JSON query. Note the use of sprintf to embed double-quotes in the POST data.

```
bundle req
req.URL = "http://api.bls.gov/publicAPI/v1/timeseries/data/"
req.include = 1
req.header = "Content-Type: application/json"
string s = sprintf("{\"seriesid\":[\"LEU0254555900\"]}")
req.postdata = s
err = curl(&req)
if err == 0
    s = req.output
    string line
    loop while getline(s, &line)
        printf "%s\n", line
    endloop
endif
```

See also the functions jsonget and xmlget for means of processing JSON and XML data received, respectively.

dayspan

Output:	integer
Arguments:	*ed1* (integer)
	ed2 (integer)
	weeklen (integer)

Returns the number of (relevant) days between the epoch days *ed1* and *ed2*, inclusive. The *weeklen*, which must equal 5, 6 or 7, gives the number of days in the week that should be counted (a value of 6 omits Sundays, and a value of 5 omits both Saturdays and Sundays).

To obtain epoch days from the more familiar form of dates, see epochday. Related: see smplspan.

dec2bin

Output:	matrix
Argument:	*x* (matrix)

This function returns the binary representation of the numbers contained in the column vector *x*, by storing each binary digit into a column of the returned matrix, which always has 32 columns. Each element of *x* must be an integer between 0 and $2^{32}-1$. Otherwise, an error is flagged.

Note that the least significant bit comes in the first column. So column 1 corresponds to 2^0, column 2 to 2^1, and so on. For example, the expression

```
matrix B = dec2bin(5)
```

produces a row vector full of zeros except for positions 1 and 3.

The bin2dec function performs the inverse transformation.

defarray

Output:	see below
Argument:	. . . (see below)

Enables the definition of an array variable *in extenso*, by providing one or more elements. In using this function you must specify a type (in plural form) for the array: `strings`, `matrices`, `bundles`

or `lists`. Each of the arguments must evaluate to an object of the specified type. On successful completion, the return value is an array of n elements, where n is the number of arguments.

```
strings S = defarray("foo", "bar", "baz")
matrices M = defarray(I(3), X'X, A*B, P[1:])
```

See also array.

defbundle

Output: bundle
Argument: . . . (see below)

Enables the initialization of a bundle variable *in extenso*, by providing zero or more pairs of the form *key*, *member*. If we count the arguments from 1, every odd-numbered argument must evaluate to a string (key) and every even-numbered argument must evaluate to an object of a type that can be included in a bundle.

A couple of simple examples:

```
bundle b1 = defbundle("s", "Sample string", "m", I(3))
bundle b2 = defbundle("yn", normal(), "x", 5)
```

The first example creates a bundle with members a string and a matrix; the second, a bundle with a series member and a scalar member. Note that you cannot specify a type for each argument when using this function, so you must accept the "natural" type of the argument in question. If you wanted to add a series with constant value 5 to a bundle named `b1` it would be necessary to do something like the following (after declaring `b1`):

```
series b1.s5 = 5
```

If no arguments are given to this function it is equivalent to creating an empty bundle (or to emptying an existing bundle of its content), as could also be done via

```
bundle b = null
```

Variant syntax

Two alternative forms of syntax are available for defining bundles. In each case the keyword `defbundle` is replaced by a single underscore. In the first variant the comma-separated arguments take the form `key=value`, where the key is taken to be a literal string and does not require quotation. Here is an example:

```
bundle b = _(x=5, strval="some string", m=I(3))
```

This form is particularly convenient for constructing an anonymous bundle on the fly as a function argument, as in

```
b = regls(ys, LX, _(lfrac=0.35, stdize=0))
```

where the `regls` function takes an optional bundle argument holding various parameters.

The second variant is designed for the case where you wish to pack several pre-existing named objects into a bundle: you just give their names, unquoted:

```
bundle b = _(x, y, z)
```

Here the object `x` is copied into the bundle under the key "x", and similarly for `y` and `z`.

These alternative forms involve less typing than the full `defbundle()` version and are likely to be more convenient in many cases, but note that they are less flexible. Only the full version can handle keys given as string variables rather than literal strings.

deflist

Output: list
Argument: . . . (see below)

Defines a list (of named series), given one or more suitable arguments. Each argument must be a named series (given by name or integer ID number), an existing named list, or an expression which evaluates to a list (including a vector which can be interpreted as a set of series ID numbers).

One point to note: this function simply concatenates its arguments to produce the list that it returns. If the intent is that the return value does not contain duplicates (does not reference any given series more than once), it is up to the caller to ensure that requirement is satisfied.

deseas

Output: series
Arguments: *x* (series)
opts (bundle, optional)

The primary purpose of this function is to produce a deseasonalized version of the (quarterly or monthly) input series *x*, using X-13ARIMA-SEATS; it is available only if X-13ARIMA-SEATS is installed. If the second, optional argument is omitted, seasonal adjustment is carried out with all X-13ARIMA options at their default values (fully automatic procedure). When *opts* is supplied, it may contain any of the following option specifications.

- `verbose`: what to print? 0 = nothing (the default); 1 = confirmation of the options selected; 2 = confirmation of options plus the output from X-13ARIMA.
- `seats`: 1 to use the SEATS algorithm in place of the default X11 algorithm for seasonal adjustment, or 0.
- `airline`: 1 to use the "airline" ARIMA model specification (0,1,1)(0,1,1) in place of the default automatic model selection, or 0.
- `arima`: can be used to impose a chosen ARIMA specification, in the form of a 6-vector holding small non-negative integers. These are given the (p,d,q,P,D,Q) interpretation, in traditional time-series notation: the first three terms represent the non-seasonal AR, Integration and MA orders, and the last three the seasonal counterparts. If both `airline` and `arima` are given, `arima` takes precedence.
- `outliers`: enable detection and correction for outliers (choices 1 through 7), or 0 (the default) to omit this feature. The three available outlier types with their numerical codes are: 1 = additive outlier (ao), 2 = level shift (ls), 4 = temporary change (tc). To combine options you add the codes, for example 1 + 2 + 4 = 7 to activate all three. Note that the choice 3 = 1 + 2 (ao and ls) is the default within X-13ARIMA-SEATS, and is selected via the outlier tickbox in gretl's dialog window for seasonal adjustment via X13.

- `critical`: a positive scalar, the critical value for defining outliers, the default being automatic, dependent on the sample size. Relevant only when `outliers` is specified.
- `logtrans`: should the input series be put in log form? 0 = no, 1 = yes, 2 = automatically selected (the default). Note that it is not recommended to pass the input series in log form; if you want the log to be used, pass the "raw" level but specify `logtrans=1`.
- `trading_days`: should trading-day effects be included? 0 = no, 1 = yes, 2 = automatic (the default).
- `working_days`: a simpler version of `trading_days` with a single distinction between weekdays and weekends rather than individual day effects. 0 = no (the default), 1 = yes, 2 = automatic. Use only one of `trading_days` and `working_days`.
- `easter`: 1 to allow for an easter effect, as a supplement to either `trading_days` or `working_days`, or 0 (the default).
- `output`: a string to select the type of the output series, `"sa"` for deseasonalized (the default), `"trend"` for the estimated trend, or `"irreg"` for the irregular component.
- `save_spc`: boolean flag, default 0; see below.

Augmented results

In some cases one may wish to obtain all three of the results available from X-13ARIMA via a single call to `deseas`. This is supported as follows. Pass the *opts* bundle in pointer form, and give the string `"all"` under the `output` key. The direct return value is then the seasonally adjusted series, but on successful completion *opts* will contain a matrix named `results` with three columns: seasonally adjusted, trend and irregular. Here's an illustration (where the direct return value is discarded).

```
bundle b = _(output="all")
deseas(y, &b)
series y_dseas = b.results[,1]
series y_trend = b.results[,2]
series y_irreg = b.results[,3]
```

Saving the X-13ARIMA specification

The `save_spc` flag can be used to save the content of the X-13ARIMA input file written by gretl. The options bundle should be passed in pointer form and the specification (as a string) can be found under the key `x13a_spc`. The following code illustrates saving this to file under the name `myspec.spc` in the user's working directory. (Note that the `.spc` extension is required by X-13ARIMA.)

```
bundle b = _(save_spc=1)
deseas(y, &b)
outfile myspec.spc
   print b.x13a_spc
end outfile
```

det

Output: scalar
Argument: *A* (square matrix)

Returns the determinant of *A*, computed via the LU factorization. If what you actually want is the log determinant you should call ldet instead. See also rcond, cnumber.

diag

Output:	matrix
Argument:	X (matrix)

Returns the principal diagonal of X in a column vector. Note: if X is an $m \times n$ matrix, the number of elements of the output vector is $\min(m, n)$. See also tr.

diagcat

Output:	matrix
Arguments:	A (matrix)
	B (matrix)

Returns the direct sum of A and B, that is a matrix holding A in its north-west corner and B in its south-east corner. If both A and B are square, the resulting matrix is block-diagonal.

diff

Output:	same type as input
Argument:	y (series, matrix or list)

Computes first differences. If y is a series, or a list of series, starting values are set to NA. If y is a matrix, differencing is done by columns and starting values are set to 0.

When a list is returned, the individual variables are automatically named according to the template d_ *varname* where *varname* is the name of the original series. The name is truncated if necessary, and may be adjusted in case of non-uniqueness in the set of names thus constructed.

See also cum, ldiff, sdiff.

digamma

Output:	same type as input
Argument:	x (scalar, series or matrix)

Returns the digamma (or Psi) function of x, that is $\frac{\mathrm{d}}{\mathrm{d}x} \log\Gamma(x)$.

See also lngamma, trigamma.

distance

Output:	matrix
Arguments:	X (matrix)
	metric (string, optional)
	Y (matrix, optional)

Computes distances between points on a metric that can be euclidean (the default), manhattan, hamming, chebyshev, cosine or mahalanobis. The string identifying the metric can be given as an unambiguous truncation. The additional metrics correlation, standardized Euclidean are supported via simple transformations of the inputs; see below.

Each row of the $m \times n$ matrix X is treated as a point in an n-dimensional space; in an econometric context this is likely to represent a single observation comprising the values of n variables.

Standard cases

This section applies to all metrics except the Mahalanobis distance, for which the syntax is slightly different (see below).

If Y is not given, the return value is a column vector of length $m(m-1)/2$ comprising the non-redundant subset of all pairwise distances between the m points (rows of X). Given such a vector named d, the full symmetric matrix of inter-point distances (with zeros on the principal diagonal) can be constructed via

```
D = unvech(d, 0)
```

since d is akin to the vech of D, with diagonal elements omitted. The optional second argument to unvech says that the diagonal should be filled with zeros.

If Y is given, it must be a $p \times n$ matrix, each row of which is again treated as a point in n-space. In this case the return value is an $m \times p$ matrix whose i, j element holds the distance between row i of X and row j of Y.

To obtain the distances from a given reference point (for example, the centroid) to each of n data-points, give Y as a single row.

Definitions of the supported metrics

- euclidean: the square root of the sum of squared deviations in each of the dimensions.
- manhattan: the sum of the absolute deviations in each of the dimensions.
- hamming: the proportion of the dimensions in which the deviation is non-zero (so bounded by 0 and 1).
- chebyshev: the greatest absolute deviation in any dimension.
- cosine: 1 minus the cosine of the angle between the "points", considered as vectors.

Mahalanobis distance

Mahalanobis distances are defined as the Euclidean distances between the points in question (rows of X) and a given centroid, scaled by the inverse of a covariance matrix. In the simplest case the centroid is constituted by the sample means of the variables (columns of X) and the covariance matrix is their sample covariance.

These can be obtained by supplying as second argument the string "mahalanobis" or any unambiguous abbreviation, as in

```
dmahal = distance(X, "mahal")
```

In this case the third argument Y is not supported, and the return value is a column vector of length m with the Mahalanobis distances from the centroid of X (that is, its sample mean). In practice, the output matrix in this case is the same you get by executing the mahal command on a list of series corresponding to the columns of X.

To obtain Mahalanobis distances using a different centroid, mu, and/or inverse covariance matrix, ICV, the following syntax can be used:

```
dmahal = distance(X*cholesky(ICV), "euc", mu)
```

Other metrics

Standardized Euclidean distances and correlation distances can be obtained as follows:

```
# standardized euclidean
dseu = distance(stdize(X), "eu")
# correlation (based on cosine)
dcor = distance(stdize(X', -1)', "cos")
```

dnorm

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the density of the standard normal distribution at *x*. To get the density for a non-standard normal distribution at x, pass the z-score of x to the `dnorm` function and multiply the result by the Jacobian of the z transformation, namely 1 over σ, as illustrated below:

```
mu = 100
sigma = 5
x = 109
fx = (1/sigma) * dnorm((x-mu)/sigma)
```

See also cnorm, qnorm.

dropcoll

Output:	list
Arguments:	*X* (list)
	epsilon (scalar, optional)

Returns a list with the same elements as *X*, but for the collinear series. Therefore, if all the series in *X* are linearly independent, the output list is just a copy of *X*.

The algorithm uses the QR decomposition (Householder transformation), so it is subject to finite precision error. In order to gauge the sensitivity of the algorithm, a second optional parameter *epsilon* may be specified to make the collinearity test more or less strict, as desired. The default value for *epsilon* is 1.0e-8. Setting *epsilon* to a larger value increases the probability of a series to be dropped.

Example:

```
nulldata 20
set seed 9876
series foo = normal()
series bar = normal()
series foobar = foo + bar
list X = foo bar foobar
list Y = dropcoll(X)
list print X
list print Y
# set epsilon to a ridiculously small value
list Y = dropcoll(X, 1.0e-30)
list print Y
```

produces

```
? list print X
foo bar foobar
? list print Y
foo bar
? list Y = dropcoll(X, 1.0e-30)
Replaced list Y
? list print Y
foo bar foobar
```

dsort

Output:	same type as input
Argument:	*x* (series, vector or strings array)

Sorts *x* in descending order, skipping observations with missing values when *x* is a series. See also sort, values.

dummify

Output:	list
Arguments:	*x* (series)
	omitval (scalar, optional)

The argument *x* should be a discrete series. This function creates a set of dummy variables coding for the distinct values in the series. By default the smallest value is taken as the omitted category and is not explicitly represented.

The optional second argument represents the value of *x* which should be treated as the omitted category. The effect when a single argument is given is equivalent to `dummify(x, min(x))`. To produce a full set of dummies, with no omitted category, use `dummify(x, NA)`.

The generated variables are automatically named according to the template `D`*varname_i* where *varname* is the name of the original series and *i* is a 1-based index. The original portion of the name is truncated if necessary, and may be adjusted in case of non-uniqueness in the set of names thus constructed.

easterday

Output:	same type as input
Argument:	*y* (scalar, series or matrix)

Given the year in argument *y*, returns the date of Easter on the Gregorian calendar as $month + day/100$. For example, in 2014 the date of Easter was April 20, which is represented under this convention as 4.2. (Note that April 2 would be returned as 4.02.) The following code shows how month and day can be extracted from the return value.

```
scalar e = easterday(2014)
scalar m = floor(e)
scalar d = round(100*(e-m))
```

ecdf

Output:	matrix
Argument:	*y* (series or vector)

Calculates the empirical CDF of y. This is returned in a matrix with two columns: the first holds the sorted unique values of y and the second holds the cumulative relative frequency,

$$F(y) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \leq y)$$

where n is total number of observations and $I()$ denotes the indicator function.

eigen

Output:	matrix
Arguments:	*A* (square matrix)
	&V (reference to matrix, or `null`)
	&W (reference to matrix, or `null`)

Computes the eigenvalues, and optionally the right and/or left eigenvectors, of the $n \times n$ matrix A, which may be real or complex. The eigenvalues are returned in a complex column vector. To obtain the norm of the eigenvalues, you can use the abs function, which accepts complex arguments.

If you wish to retrieve the right eigenvectors (as an $n \times n$ complex matrix), supply the name of an existing matrix, preceded by & to indicate the "address" of the matrix in question, as the second argument. Otherwise this argument can be omitted.

To retrieve the left eigenvectors (again, as a complex matrix), supply a matrix-address as the third argument. Note that if you want the left eigenvectors but not the right ones, you should use the keyword `null` as a placeholder for the second argument.

See also eigensym, eigsolve, svd.

eigengen

Output:	matrix
Arguments:	*A* (square matrix)
	&U (reference to matrix, or `null`)

This is a legacy function, predating gretl's native support for complex matrices. It should not be used in newly written hansl scripts. Use eigen *instead.*

Computes the eigenvalues, and optionally the right eigenvectors, of the $n \times n$ matrix A. If all the eigenvalues are real an $n \times 1$ matrix is returned; otherwise the result is an $n \times 2$ matrix, the first column holding the real components and the second column the imaginary components. The eigenvalues are not guaranteed to be sorted in any particular order.

The second argument must be either the name of an existing matrix preceded by & (to indicate the "address" of the matrix in question), in which case an auxiliary result is written to that matrix, or the keyword `null`, in which case the auxiliary result is not produced.

If a non-null second argument is given, the specified matrix will be over-written with the auxiliary result. (It is not required that the existing matrix be of the right dimensions to receive the result.) The output is organized as follows:

- If the i-th eigenvalue is real, the i-th column of U will contain the corresponding eigenvector;
- If the i-th eigenvalue is complex, the i-th column of U will contain the real part of the corresponding eigenvector and the next column the imaginary part. The eigenvector for the conjugate eigenvalue is the conjugate of the eigenvector.

In other words, the eigenvectors are stored in the same order as the eigenvalues, but the real eigenvectors occupy one column, whereas complex eigenvectors take two (the real part comes first); the total number of columns is still n, because the conjugate eigenvector is skipped.

See also eigensym, eigsolve, qrdecomp, svd.

eigensym

Output:	matrix
Arguments:	*A* (symmetric matrix)
	&U (reference to matrix, or `null`)

Works mostly as eigen except that the argument *A* must be symmetric (in which case less calculation is required), and the eigenvalues are returned in ascending order. If you want to get the eigenvalues in descending order (and have the eigenvectors reordered correspondingly) you can do the following:

```
matrix U
e = eigensym(A, &U)
Tmp = msortby((-e' | U)',1)'
e = -Tmp[1,]'
U = Tmp[2:,]
# now largest to smallest eigenvalues
print e U
```

Note: if you're interested in the eigen-decomposition of a matrix of the form $X'X$ it's preferable to compute the argument via the prime operator `X'X` rather than using the more general syntax `X'*X`. The former expression uses a specialized algorithm which offers greater computational efficiency as well as ensuring that the result is exactly symmetric.

eigsolve

Output:	matrix
Arguments:	*A* (symmetric matrix)
	B (symmetric matrix)
	&U (reference to matrix, or `null`)

Solves the generalized eigenvalue problem $|A - \lambda B| = 0$, where both *A* and *B* are symmetric and *B* is positive definite. The eigenvalues are returned directly, arranged in ascending order. If the optional third argument is given it should be the name of an existing matrix preceded by &; in that case the generalized eigenvectors are written to the named matrix.

epochday

Output:	scalar or series
Arguments:	*year* (scalar or series)
	month (scalar or series)
	day (scalar or series)

Returns the number of the day in the current epoch specified by year, month and day. The epoch day equals 1 for the first of January in the year 1 AD on the proleptic Gregorian calendar; it stood at 733786 on 2010-01-01. If any of the arguments are given as series the value returned is a series, otherwise it is a scalar.

By default the *year*, *month* and *day* values are assumed to be given relative to the Gregorian calendar, but if the year is a negative value the interpretation switches to the Julian calendar.

An alternative call is also supported: if a single argument is given, it is taken to be a date (or series of dates) in ISO 8601 "basic" numeric format, `YYYYMMDD`. So the following two calls produce the same result, namely 700115.

```
eval epochday(1917, 11, 7)
eval epochday(19171107)
```

For the inverse function, see isodate and also (for the Julian calendar) juldate. For another means of converting dates to epoch days see strpday.

errmsg

Output:	string
Argument:	*errno* (integer)

Retrieves the gretl error message associated with *errno*. See also $error.

errorif

Output:	scalar
Arguments:	*condition* (boolean)
	msg (string)

Applicable only in the context of a user-defined function, or within an mpi block. If *condition* evaluates as non-zero, it causes execution of the current function to terminate with an error condition flagged; the *msg* argument is then printed as part of the message shown to the caller of the function in question.

The return value from this function (1) is purely nominal.

exists

Output:	integer
Argument:	*name* (string)

Returns non-zero if *name*, which should be valid as a gretl identifier, names a currently defined object, be it a scalar, a series, a matrix, list, string, bundle or array; otherwise returns 0.

Intended usage is for the case where a user-defined function has an optional parameter with a `null` default. The function writer can use `exists()`, passing the parameter name, to check whether or not the caller supplied an argument. But please note, lists are an exception in this respect: if a list parameter has a `null` default and the caller doesn't supply an argument, the function gets an empty list rather than no list; therefore `exists` will always return non-zero. To check for emptiness of a list argument, use nelem.

For related checks, see typeof and inbundle.

exp

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns e^x. Note that in case of matrix input the function acts element by element. For the matrix exponential function, see mexp.

fcstats

Output:	matrix
Arguments:	*y* (series or vector)
	f (series, list or matrix)
	U2 (boolean, optional)

Produces a matrix holding several statistics which serve to evaluate f as a forecast of the observed data y.

If f is a series or vector the output is a column vector; if f is a list with k members or a $T \times k$ matrix the output has k columns, each of which holds statistics for the corresponding element (series or column) of the input as a forecast of y.

In all cases the "vertical" dimension of the input (for a series or list the length of the current sample range, for a matrix the number of rows) must match across the two arguments.

The rows of the returned matrix are as follows:

```
1  Mean Error (ME)
2  Root Mean Squared Error (RMSE)
3  Mean Absolute Error (MAE)
4  Mean Percentage Error (MPE)
5  Mean Absolute Percentage Error (MAPE)
6  Theil's U (U1 or U2)
7  Bias proportion, UM
8  Regression proportion, UR
9  Disturbance proportion, UD
```

The variant of Theil's U shown by default depends on the nature of the data: if they are known to be time series then U2 is shown, otherwise U1 is produced. But this choice can be forced via the optional trailing argument: give a non-zero value to force U2, or zero to force U1.

For details on the calculation of these statistics, and the interpretation of the U values, please see chapter 35 of the *Gretl User's Guide*.

fdjac

Output:	matrix
Arguments:	*b* (column vector)
	fcall (function call)
	h (scalar, optional)

Calculates a numerical approximation to the Jacobian associated with the n-vector b and the transformation function specified by the argument *fcall*. The function call should take b as its first argument (either straight or in pointer form), followed by any additional arguments that may be needed, and it should return an $m \times 1$ matrix. On successful completion `fdjac` returns an $m \times n$ matrix holding the Jacobian.

The optional third argument can be used to set the step size h used in the approximation mechanism (see below); if this argument is omitted the step size is determined automatically.

Here is an example of usage:

```
matrix J = fdjac(theta, myfunc(&theta, X))
```

The function can use three different methods: simple forward-difference, bilateral difference or 4-nodes Richardson extrapolation. Respectively:

$$J_0 = \frac{f(x+h) - f(x)}{h}$$

$$J_1 = \frac{f(x+h) - f(x-h)}{2h}$$

$$J_2 = \frac{8(f(x+h) - f(x-h)) - (f(x+2h) - f(x-2h))}{12h}$$

The three alternatives above provide, generally, a trade-off between accuracy and speed. You can choose among methods via the set command: specify a value of 0, 1 or 2 for the `fdjac_quality` variable. The default is 0.

For more details and examples chapter 37 of the *Gretl User's Guide*.

See also BFGSmax, numhess, set.

feval

Output:	see below
Arguments:	*funcname* (string)
	. . . (see below)

Primarily useful for writers of functions. The first argument should be the name of a function; the remaining arguments will be passed to the specified function. This permits treating the function identified by *funcname* as itself a variable. The return value is whatever the named function returns given the specified arguments.

The example below illustrates some possible uses.

```
function scalar utility (scalar c, scalar sigma)
    return (c^(1-sigma)-1)/(1-sigma)
end function

strings S = defarray("log", "utility")

# call a 1-argument built-in function
x = feval(S[1], 2.5)
# call a user-defined function
x = feval(S[2], 5, 0.5)
# a 2-argument built-in function
func = "zeros"
m = feval(func, 5-2, sqrt(4))
print m
# a 3-argument built-in
x = feval("monthlen", 12, 1980, 5)
```

There's a weak analogy between `feval` and genseries: both functions render variable a syntactic element that is usually fixed at the time a script is composed.

See also fevalb.

fevalb

Output:	see below
Arguments:	*funcname* (string)
	b (bundle)

This is a variant of feval which meets a case that may be encountered by function writers, where the number and types of the arguments to be passed to the named function are not known in advance. Instead of the arguments being passed individually, they are passed as members of the bundle argument *b*.

Since the order of the members in a gretl bundle is indeterminate, some mechanism is required to ensure that they are passed to the function in question in the right order. This is automatically ensured if the lexicographic order of the keys in the bundle gives the argument order. For examples, the keys could be `arg1`, `arg2` and so on (or `arg01`, `arg02` and so on in the unlikely event that the function takes more than nine arguments). Alternatively, the bundle may contain an array of strings under the reserved key `arglist`. This array must hold exactly the keys in *b*, except for `arglist` itself, in the desired order.

The examples below illustrate both approaches, as applied to the monthlen function.

```
# using lexicographic order
bundle b = _(arg1=12, arg2=1980, arg3=5)
n = feval("monthlen", b)

# using arglist
bundle b = _(month=12, year=1980, wkdays=5)
b.arglist = defarray("month", "year", "wkdays")
n = feval("monthlen", b)
```

See also feval.

fevd

Output:	matrix
Arguments:	*target* (integer)
	shock (integer)
	sys (bundle, optional)

This function provides a more flexible alternative to the accessor $fevd for obtaining a forecast error variance decomposition (FEVD) matrix following estimation of a VAR or VECM. Without the final optional argument, it is available only when the last model estimated was a VAR or VECM. Alternatively, information on such a system can be stored in a bundle via the $system accessor and subsequently passed to `fevd`.

The *target* and *shock* arguments take the form of 1-based indices of the endogenous variables in the system, with 0 taken to mean "all". The following code fragment illustrates usage. In the first example the matrix `fe1` holds the shares of the FEVD for `y1` due to each of `y1`, `y2` and `y3` (the rows therefore summing to 1). In the second, `fe2` holds the contribution of `y2` to the forecast error variance of all three variables (so the rows do not sum to 1). In the third case the return value is a column vector showing the "own share" of the FEVD for `y1`.

```
var 4 y1 y2 y3
bundle vb = $system
matrix fe1 = fevd(1, 0, vb)
matrix fe2 = fevd(0, 2, vb)
matrix fe3 = fevd(1, 1, vb)
```

The number of periods (rows) over which the decomposition is traced is determined automatically based on the frequency of the data, but this can be overridden via the `horizon` argument to the set command, as in `set horizon 10`.

See also irf.

fft

Output:	matrix
Argument:	*X* (matrix)

Discrete Fourier transform. The input matrix X may be real or complex. The output is a complex matrix of the same dimensions as X.

Should it be necessary to compute the Fourier transform on several vectors with the same number of elements, it is more efficient to group them into a matrix rather than invoking `fft` for each vector separately. See also ffti.

ffti

Output:	matrix
Argument:	X (matrix)

Inverse discrete Fourier transform. It is assumed that X contains n complex column vectors. A matrix with n columns is returned.

Should it be necessary to compute the inverse Fourier transform on several vectors with the same number of elements, it is more efficient to group them into a matrix rather than invoking `ffti` for each vector separately. See also fft.

filter

Output:	see below
Arguments:	x (series or matrix)
	a (scalar or vector, optional)
	b (scalar or vector, optional)
	$y0$ (scalar, optional)
	$x0$ (scalar or vector, optional)

Computes an ARMA-like filtering of the argument x. The transformation can be written as

$$y_t = \sum_{i=0}^{q} a_i x_{t-i} + \sum_{i=1}^{p} b_i y_{t-i}$$

If argument x is a series, the result will be itself a series. Otherwise, if x is a matrix with T rows and k columns, the result will be a matrix of the same size, in which the filtering is performed column by column.

The two arguments a and b are optional. They may be scalars, vectors or the keyword `null`.

If a is a scalar, this is used as a_0 and implies $q = 0$; if it is a vector of $q + 1$ elements, they contain the coefficients from a_0 to a_q. If a is `null` or omitted, this is equivalent to setting $a_0 = 1$ and $q = 0$.

If b is a scalar, this is used as b_1 and implies $p = 1$; if it is a vector of p elements, they contain the coefficients from b_1 to b_p. If b is `null` or omitted, this is equivalent to setting $B(L) = 1$.

The optional scalar argument $y0$ is taken to represent all values of y prior to the beginning of sample (used only when $p > 0$). If omitted, it is understood to be 0. Similarly, the optional argument $x0$ may be used to specify one or more pre-sample values of x, information that is relevant only when $q > 0$. Otherwise pre-sample values of x are assumed to be zero.

See also bkfilt, bwfilt, fracdiff, hpfilt, movavg, varsimul.

Example:

```
nulldata 5
y = filter(index, 0.5, -0.9, 1)
print index y --byobs
x = seq(1,5)' ~ (1 | zeros(4,1))
w = filter(x, 0.5, -0.9, 1)
```

```
print x w
```

produces

```
index            y

1            1     -0.40000
2            2      1.36000
3            3      0.27600
4            4      1.75160
5            5      0.92356

x (5 x 2)

1   1
2   0
3   0
4   0
5   0

w (5 x 2)

-0.40000     -0.40000
 1.3600       0.36000
 0.27600     -0.32400
 1.7516       0.29160
 0.92356     -0.26244
```

firstobs

Output:	integer
Arguments:	*y* (series)
	insample (boolean, optional)

Returns the 1-based index of the first non-missing observation for the series *y*. By default the whole data range is examined, so if subsampling is in effect the value returned may be smaller than the accessor $t1. But if a non-zero value is given for *insample* only the current sample range is considered. See also lastobs.

fixname

Output:	string
Arguments:	*rawname* (string)
	underscore (boolean, optional)

Primarily intended for use in connection with the join command. Returns the result of converting *rawname* to a valid gretl identifier, which must start with a letter, contain nothing but (ASCII) letters, digits and the underscore character, and must not exceed 31 characters. The rules used in conversion are:

1. Skip any leading non-letters.

2. Until the 31-character limit is reached or the input is exhausted: transcribe "legal" characters; skip "illegal" characters apart from spaces; and replace one or more consecutive spaces with an underscore, unless the previous character transcribed is an underscore in which case space is skipped.

If you are confident that the input is not too long (and hence subject to truncation), you may wish to have sequences of one or more illegal characters replaced with an underscore rather than just being deleted; this may produce a more readable identifier. To get this effect, supply a nonzero value for the optional second argument. But this is not advisable in the context of the join command, since the automatically "fixed" name will not use underscores in this way.

flatten

Output:	see below
Arguments:	*A* (array of matrices or strings)
	alt (integer or string, optional)

"Flattens" either an array of matrices into a single matrix or an array of strings into a single string.

Matrices

In the matrix case, the way the matrices in *A* are joined together depends on the the *alt* argument, which should have value 0 (horizontal), 1 (vertical) or 2 ("vec-wise"). The best way to explain the difference between the three alternatives is by example: the code

```
X = {1,3,5; 2,4,6}
A = defarray(X, X+6)
U = flatten(A,0) # = A[1] ~ A[2]
V = flatten(A,1) # = A[1] | A[2]
W = flatten(A,2) # = vec(A[1]) ~ vec(A[2])
```

produces the following three matrices:

```
U (2 x 6)

 1    3    5    7    9   11
 2    4    6    8   10   12

V (4 x 3)

 1    3    5
 2    4    6
 7    9   11
 8   10   12

W (6 x 2)

 1    7
 2    8
 3    9
 4   10
 5   11
 6   12
```

An error is flagged if the matrices in the array are not conformable for the operation. See msplitby for the inverse operation.

Strings

In the string case the result holds the strings in *A*, arranged one per line by default. If a non-zero numerical value is given for *alt* the strings are separated by spaces rather than newlines, but an

alternative usage of *alt* is supported: you may give a specific string to use as the separator. The inverse function for the string case is strsplit.

floor

Output:	same type as input
Argument:	*y* (scalar, series or matrix)

Returns the greatest integer less than or equal to *x*. Note: int and `floor` differ in their effect for negative arguments: `int(-3.5)` gives −3, while `floor(-3.5)` gives −4.

fracdiff

Output:	series
Arguments:	*y* (series)
	d (scalar)

$$\Delta^d y_t = y_t - \sum_{i=1}^{\infty} \psi_i y_{t-i}$$

where

$$\psi_i = \frac{\Gamma(i-d)}{\Gamma(-d)\Gamma(i+1)}$$

Note that in theory fractional differentiation is an infinitely long filter. In practice, presample values of y_t are assumed to be zero.

A negative value of *d* can be given, in which case fractional integration is performed.

fzero

Output:	scalar
Arguments:	*fcall* (function call)
	init (scalar or vector, optional)
	toler (scalar, optional)

Attempts to find a single root of a continuous (typically nonlinear) function f—that is, a value of the scalar variable x such that $f(x) = 0$. The *fcall* argument should provide a call to the function in question; *fcall* may include an arbitrary number of arguments but the first one must be the scalar playing the role of x. On successful completion the value of the root is returned.

The method used is that of Ridders (1979). This requires an initial bracket $\{x_0, x_1\}$ such that both x values lie in the domain of the function and the respective function values are of opposite sign. Best results are likely to be obtained if the user can supply, via the second argument, a 2-vector holding suitable end-points for the bracket. Failing that, one can supply a single scalar value and `fzero` will try to find a counterpart. If the second argument is omitted, x_0 is initialized to a small positive value and we search for a suitable x_1.

The optional *toler* argument can be used to adjust the maximum acceptable absolute difference of $f(x)$ from zero, the default being 1.0e−14.

By default this function operates silently, but the progress of the iterative method can be exposed by executing the command "`set max_verbose on`" before calling `fzero`.

Some simple examples follow.

```
# Approximate pi by finding a zero for sin() in the
# bracket 2.8 to 3.2
x = fzero(sin(x), {2.8, 3.2})
```

```
printf "\nx = %.12f vs pi = %.12f\n\n", x, $pi

# Approximate the 'Omega constant' starting from x = 0.5
function scalar f(scalar x)
    return log(x) + x
end function
x = fzero(f(x), 0.5)
printf "x = %.12f f(x) = %.15f\n", x, f(x)
```

gammafun

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the gamma function of *x*.

See also bincoeff and lngamma.

genseries

Output:	scalar
Arguments:	*varname* (string)
	rhs (series)

Provides the script writer with a convenient means of generating series whose names are not known in advance, and/or creating a series and appending it to a list in a single operation.

The first argument gives the name of the series to create (or modify); this can be a string literal, a string variable, or an expression that evaluates to a string. The second argument, *rhs* ("right-hand side"), defines the source series: this can be the name of an existing series or an expression that evaluates to a series, as would appear to the right of the equals sign when defining a series in the usual way.

The return value from this function is the ID number of the series in the dataset, a value suitable for inclusion in a list (or -1 on failure).

For example, suppose you want to add n random normal series to the dataset and put them all into a named list. The following will do the job:

```
nulldata 10
list Normals = null
scalar n = 3
loop i = 1 .. n
    Normals += genseries(sprintf("norm%d", i), normal())
endloop
```

On completion `Normals` will contain the series `norm1`, `norm2` and `norm3` .

Those who find `genseries` useful may also like to explore feval.

geoplot

Output:	none
Arguments:	*mapfile* (string)
	payload (series, optional)
	options (bundle, optional)

Calls for production of a map, when suitable geographical data are present. In most cases the *mapfile* argument should be given as $mapfile, an accessor that retrieves the name of the relevant GeoJSON file or ESRI shapefile. The optional *payload* argument is used to give the name of a series with which to colorize the regions of the map. And the final bundle argument enables you to set numerous options.

See the geoplot documentation, `geoplot.pdf`, for full details and examples. This explains all the settings configurable via the *options* argument.

getenv

Output: string
Argument: *s* (string)

If an environment variable by the name of *s* is defined, returns the string value of that variable, otherwise returns an empty string. See also ngetenv.

getinfo

Output: bundle
Argument: *y* (series)

Returns information on the specified series, which may be given by name or ID number. The returned bundle contains all the attributes which can be set via the setinfo command. It also contains additional information relevant for series that have been created as transformations of primary data (lags, logs, etc.): this includes the gretl command word for the transformation under the key "transform" and the name of the associated primary series under "parent". For lagged series, the specific lag number can be found under the key "lag".

Here is an example of usage:

```
open data9-7
lags QNC
bundle b = getinfo(QNC_2)
print b
```

On executing the above we see:

```
has_string_table = 0
lag = 2
parent = QNC
name = QNC_2
graph_name =
coded = 0
discrete = 0
transform = lags
description = = QNC(t - 2)
```

To test whether series 5 in a dataset is a lagged term one can do this sort of thing:

```
if getinfo(5).lag != 0
   printf "series 5 is a lag of %s\n", getinfo(5).parent
endif
```

Note that the dot notation to access bundle members can be used even when the bundle is "anonymous" (not saved under its own name).

getkeys

Output:	array of strings
Argument:	*b* (bundle)

Returns an array of strings holding the keys identifying the contents of *b*. If the bundle is empty an empty array is returned.

getline

Output:	scalar
Arguments:	*source* (string)
	&target (reference to string)

This function is used to read successive lines from *source*, which should be a named string variable. On each call a line from the source is written to *target* (which must also be a named string variable, given in pointer form), with the newline character stripped off. The valued returned is 1 if there was anything to be read (including blank lines), 0 if the source has been exhausted.

Here is an example in which the content of a text file is broken into lines:

```
string s = readfile("data.txt")
string line
scalar i = 1
loop while getline(s, &line)
    printf "line %d = '%s'\n", i++, line
endloop
```

In this example we can be sure that the source is exhausted when the loop terminates. If the source might not be exhausted you should follow your regular call(s) to `getline` with a "clean up" call, in which *target* is replaced by `null` (or omitted altogether) as in

```
getline(s, &line) # get a single line
getline(s, null) # clean up
```

Note that although the reading position advances at each call to `getline`, *source* is not modified by this function, only *target*.

ghk

Output:	matrix
Arguments:	*C* (matrix)
	A (matrix)
	B (matrix)
	U (matrix)
	&dP (reference to matrix, or `null`)

Computes the GHK (Geweke, Hajivassiliou, Keane) approximation to the multivariate normal distribution function; see for example Geweke (1991). The value returned is an $n \times 1$ vector of probabilities.

The argument C ($m \times m$) should give the Cholesky factor (lower triangular) of the covariance matrix of m normal variates. The arguments A and B should both be $n \times m$, giving respectively the lower and upper bounds applying to the variates at each of n observations. Where variates are unbounded, this should be indicated using the built-in constant $huge or its negative.

The matrix U should be $m \times r$, with r the number of pseudo-random draws from the uniform distribution; suitable functions for creating U are muniform and halton.

We illustrate below with a relatively simple case where the multivariate probabilities can be calculated analytically. The series P and Q should be numerically very similar to one another, P being the "true" probability and Q its GHK approximation:

```
nulldata 20
series inf1 = -2*uniform()
series sup1 = 2*uniform()
series inf2 = -2*uniform()
series sup2 = 2*uniform()

scalar rho = 0.25
matrix V = {1, rho; rho, 1}

series P = cdf(D, rho, inf1, inf2) - cdf(D, rho, sup1, inf2) \
- cdf(D, rho, inf1, sup2) + cdf(D, rho, sup1, sup2)

C = cholesky(V)
U = halton(2, 100)

series Q = ghk(C, {inf1, inf2}, {sup1, sup2}, U)
```

The optional *dP* argument can be used to retrieve the $n \times k$ matrix of analytical derivatives of the probabilities, where k equals $2m + m(m + 1)/2$. The first m columns hold the derivatives with respect to the lower bounds, the next m those with respect to the upper bounds, and the remainder the derivatives with respect to the unique elements of the C matrix in "vech" order.

gini

Output:	scalar
Argument:	*y* (series or vector)

Returns Gini's inequality index for the (non-negative) series or vector *y*. A Gini value of zero indicates perfect equality. The maximum Gini value for a series with n members is $(n - 1)/n$, occurring when only one member has a positive value; a Gini of 1.0 is therefore the limit approached by a large series with maximal inequality.

ginv

Output:	matrix
Arguments:	*A* (matrix)
	tol (scalar, optional)

Returns A^+, the Moore–Penrose or generalized inverse of the $r \times c$ matrix A, computed via the singular value decomposition.

The result of this operation depends on the number of singular values of A that are found to be numerically 0. The *tol* optional parameter can be used for tweaking this aspect. Singular values are considered to be 0 if they are less than mÖtolÖs, where m is the greater of r and c and s is the largest singular value. If the second argument is omitted *tol* is set to machine epsilon (see $macheps). In some cases, you may want to set *tol* to a larger value (eg 1.0e-9) in order to avoid overestimating the rank of A, which may lead to numerically unstable results.

This matrix has the properties

$$\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \end{aligned}$$

Moreover, the products A^+A and AA^+ are symmetric by construction.

See also inv, svd.

GSSmax

Output:	scalar
Arguments:	*&b* (reference to matrix)
	f (function call)
	toler (scalar, optional)

One-dimensional maximization via the Golden Section Search method. The matrix *b* should be a 3-vector. On input the first element is ignored while the second and third elements set the lower and upper bounds on the search. The *fncall* argument should specify a call to a function that returns the value of the maximand; element 1 of *b*, which will hold the current value of the adjustable parameter when the function is called, should be given as its first argument; any other required arguments may then follow. The function in question should be unimodal (should have no local maxima other than the global maximum) over the stipulated range, or GSS is not sure to find the maximum.

On successful completion `GSSmax` returns the optimum value of the maximand, while *b* holds the optimal parameter value along with the limits of its bracket.

The optional third argument may be used to set the tolerance for convergence, that is, the maximum acceptable width of the final bracket for the parameter. If this argument is not given a value of 0.0001 is used.

If the object is in fact minimization, either the function call should return the negative of the criterion or alternatively `GSSmax` may be called under the alias `GSSmin`.

Here is a simple example of usage:

```
function scalar trigfunc (scalar theta)
    return 4 * sin(theta) * (1 + cos(theta))
end function

matrix m = {0, 0, $pi/2}
eval GSSmax(&m, trigfunc(m[1]))
printf "\n%10.7f", m
```

GSSmin

Output:	scalar

An alias for GSSmax; if called under this name the function acts as a minimizer.

halton

Output:	matrix
Arguments:	*m* (integer)
	r (integer)
	offset (integer, optional)

Returns an $m \times r$ matrix containing m Halton sequences of length r. The sequences are constructed using the first m primes. By default the first 10 elements of each sequence are discarded, but this figure can be adjusted via the optional *offset* argument, which should be a non-negative integer. See Halton and Smith (1964).

hdprod

Output:	matrix
Arguments:	X (matrix)
	Y (matrix, optional)

Horizontal direct product. The two arguments must have the same number of rows, r. The return value is a matrix with r rows, in which the i-th row is the Kronecker product of the corresponding rows of X and Y. If Y is omitted, the "shorthand" syntax applies (see below).

If X is an $r \times k$ matrix, Y is an $r \times m$ matrix and Z is the result matrix of the horizontal direct product of X times Y, then Z will have r rows and $k \cdot m$ columns; moreover,

$$Z_{in} = X_{ij} Y_{il}$$

where $n = (j-1)m + l$.

This operation is called "horizontal direct product" in conformity to its implementation in the GAUSS programming language. Its equivalent in standard matrix algebra would be called the row-wise Khatri-Rao product, or "face-splitting" product in the signal processing literature.

Example: the code

```
A = {1,2,3; 4,5,6}
B = {0,1; -1,1}
C = hdprod(A, B)
```

produces the following matrix:

```
 0    1    0    2    0    3
-4    4   -5    5   -6    6
```

Shorthand syntax

If X and Y are the same matrix, then each row of the result is the vectorization of a symmetric matrix. In these cases, the second argument may be omitted; however, the returned matrix will only contain the non-redundant columns, and will therefore have $k(k+1)/2$ columns. For example,

```
A = {1,2,3; 4,5,6}
C = hdprod(A)
```

produces

```
1    2    3    4    6    9
16   20   24   25   30   36
```

Note that the i-th row of C is $\mathrm{vech}(a_i a_i')$, where a_i is the i-th row of A.

When using the shorthand syntax with complex matrices, the implicit second argument will be the *conjugate* of the first one, so as to make each row of the result the symmetric vectorization of a Hermitian matrix.

hfdiff

Output: list
Arguments: *hfvars* (list)
multiplier (scalar)

Given a MIDAS list, produces a list of the same length holding high-frequency first differences. The second argument is optional and defaults to unity: it can be used to multiply the differences by some constant.

hfldiff

Output: list
Arguments: *hfvars* (list)
multiplier (scalar)

Given a MIDAS list, produces a list of the same length holding high-frequency log-differences. The second argument is optional and defaults to unity: it can be used to multiply the differences by some constant, for example one might give a value of 100 to produce (approximate) percentage changes.

hflags

Output: list
Arguments: *minlag* (integer)
maxlag (integer)
hfvars (list)

Given a MIDAS list, *hfvars*, produces a list holding high-frequency lags *minlag* to *maxlag*. Use positive values for actual lags, negative for leads. For example, if *minlag* is −3 and *maxlag* is 5 then the returned list will hold 9 series: 3 leads, the contemporary value, and 5 lags.

Note that high-frequency lag 0 corresponds to the first high frequency period within a low frequency period, for example the first month of a quarter or the first day of a month.

hflist

Output: list
Arguments: *x* (vector)
m (integer)
prefix (string)

Produces from the vector *x* a MIDAS list of *m* series, where *m* is the ratio of the frequency of observation for the variable in *x* to the base frequency of the current dataset. The value of *m* must be at least 3 and the length of *x* must be *m* times the length of the current sample range.

The names of the series in the returned list are constructed from the given *prefix* (which must be an ASCII string of 24 characters or less, and valid as a gretl identifier), plus one or more digits representing the sub-period of the observation. An error is flagged if any of these names duplicate names of existing objects.

hpfilt

Output:	series
Arguments:	*y* (series)
	lambda (scalar, optional)
	one-sided (boolean, optional)

Returns the cycle component from application of the Hodrick–Prescott filter to series *y*. If the smoothing parameter, *lambda*, is not supplied then a data-based default is used, namely 100 times the square of the periodicity (100 for annual data, 1600 for quarterly data, and so on).

By default the filter is the usual two-sided version, but if the optional third argument is given with a non-zero value a one-sided variant (with no look-ahead) is computed in the manner of Stock and Watson (1999).

The most common use of the HP filter is detrending, but if it's the trend you are interested in that is easily obtained by subtraction, as in

```
series hptrend = y - hpfilt(y)
```

See also bkfilt, bwfilt.

hyp2f1

Output:	scalar or matrix
Arguments:	*a* (scalar)
	b (scalar)
	c (scalar)
	x (scalar or matrix)

Returns the Gaussian hypergeometric function ${}_2F_1(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{z^n}{n!}$ for real argument *x*. Valid values for *x* are in the closed interval $[-1, 1]$. This function is very general, and reduces to a simpler function in some special cases, such as for example $-\frac{\log(1-x)}{x}$ for $a = b = 1$ and $c = 2$. See https://en.wikipedia.org/wiki/Hypergeometric_function for more details.

If *x* is a scalar, the return value will be scalar; otherwise, it will be a matrix the same size as *x*. For example,

```
a = hyp2f1(1, 1, 2, {-1, 0, 0.5})
print a
```

produces the following output:

```
a (1 x 3)

0.69315       1.0000       1.3863
```

I

Output:	matrix
Arguments:	*n* (integer)
	m (integer, optional)

If *m* is omitted, returns an identity matrix of order *n*. Otherwise returns an $n \times m$ matrix with ones on the main diagonal and zeros elsewhere.

Im

Output: matrix
Argument: *C* (complex matrix)

Returns a real matrix of the same dimensions as *C*, holding the imaginary part of the input matrix. See also Re.

imaxc

Output: row vector
Arguments: *X* (matrix)
skip_na (boolean, optional)

Returns the row indices of the maxima of the columns of *X*. For columns containing NAs the result is also set to NA, unless the optional argument *skip_na* is nonzero, in which case the index for the maximum valid entry will be returned.

See also imaxr, iminc, maxc.

imaxr

Output: column vector
Arguments: *X* (matrix)
skip_na (boolean, optional)

Returns the column indices of the maxima of the columns of *X*. For rows containing NAs the result is also set to NA, unless the optional argument *skip_na* is nonzero, in which case the index for the maximum valid entry will be returned.

See also imaxc, iminr, maxr.

imhof

Output: scalar
Arguments: *M* (matrix)
x (scalar)

Computes Prob($u'Au < x$) for a quadratic form in standard normal variates, u, using the procedure developed by Imhof (1961).

If the first argument, *M*, is a square matrix it is taken to specify *A*, otherwise if it's a column vector it is taken to be the precomputed eigenvalues of *A*, otherwise an error is flagged.

See also pvalue.

iminc

Output: row vector
Arguments: *X* (matrix)
skip_na (boolean, optional)

Returns the row indices of the minima of the columns of *X*. For columns containing NAs the result is also set to NA, unless the optional argument *skip_na* is nonzero, in which case the index for the minimum valid entry will be returned.

See also iminr, imaxc, minc.

iminr

Output:	column vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the column indices of the minima of the rows of *X*. For rows containing NAs the result is also set to NA, unless the optional argument *skip_na* is nonzero, in which case the index for the minimum valid entry will be returned.

See also iminc, imaxr, minr.

inbundle

Output:	integer
Arguments:	*b* (bundle)
	key (string)

Checks whether bundle *b* contains a data-item with name *key*. The value returned is an integer code for the type of the item: 0 for no match, 1 for scalar, 2 for series, 3 for matrix, 4 for string, 5 for bundle, 6 for array and 7 for list. The function typestr may be used to get the string corresponding to this code.

infnorm

Output:	scalar
Argument:	*X* (matrix)

Returns the ∞-norm of the $r \times c$ matrix X, namely,

$$\|X\|_\infty = \max_i \sum_{j=1}^{c} |X_{ij}|$$

See also onenorm.

inlist

Output:	integer
Arguments:	*L* (list)
	y (series)

Returns the (1-based) position of *y* in list *L*, or 0 if *y* is not present in *L*.

The second argument may be given as the name of a series or alternatively as an integer ID number. If you know that a series of a certain name (say foo) exists, then you can call this function as, for example,

```
pos = inlist(L, foo)
```

Here you are, in effect, asking "Give me the position of series foo in list L (or 0 if it is not included in L)." However, if you are unsure whether a series of the given name exists, you should place the name in quotes:

```
pos = inlist(L, "foo")
```

In this case you are asking, "If there's a series named foo in L give me its position, otherwise return 0."

instring

Output:	integer
Arguments:	*s1* (string)
	s2 (string)
	ign_case (boolean, optional)

This is a boolean relative of strstr: it returns 1 if *s1* contains *s2*, 0 otherwise. So the conditional expression

```
if instring("cattle", "cat")
```

is logically equivalent to, but more efficient than,

```
if strlen(strstr("cattle", "cat")) > 0
```

If the optional argument *ign_case* is nonzero, the search is case-insensitive. For example,

```
instring("Cattle", "cat")
```

returns 0, but

```
instring("Cattle", "cat", 1)
```

returns 1.

instrings

Output:	see below
Arguments:	*S* (array of strings)
	test (string)
	simple (boolean, optional)

Checks the elements of the strings array *S* for equality with *test*. By default, returns a column vector of length equal to the number of matches, holding the positions of the matches within the array—or an empty matrix in case of no matches.

Example:

```
strings S = defarray("A", "B", "C", "B")
eval instrings(S, "B")
2
4
```

If a non-zero value is given for the optional *simple* argument, the return value is a scalar: 1 if *test* is found in *S*, 0 otherwise. In this case the implementation is able to take a shortcut, so it's more efficient if you just want a boolean answer.

int

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the integer part of *x*, truncating the fractional part, or NA if the result cannot be represented as a 32-bit signed integer (does not lie in the interval [−2147483648, 2147483647]).

Note: `int` and floor differ in their effect for negative arguments: `int(-3.5)` gives −3, while `floor(-3.5)` gives −4. See also ceil, floor, round.

interpol

Output: series
Argument: *x* (series)

Returns a series in which missing values in *x* are imputed via linear interpolation, for time series data or in the time dimension of a panel dataset. Extrapolation is not performed; missing values are replaced only if they are both preceded and followed by valid observations.

inv

Output: matrix
Argument: *A* (square matrix)

Returns the inverse of *A*. If *A* is singular or not square, an error message is produced and nothing is returned. Note that gretl checks automatically the structure of *A* and uses the most efficient numerical procedure to perform the inversion.

The matrix types gretl checks for are: identity; diagonal; symmetric and positive definite; symmetric but not positive definite; and triangular.

Note: it makes sense to use this function only if you plan to use the inverse of *A* more than once. If you just need to compute an expression of the form $A^{-1}B$, you'll be much better off using the "division" operators \ and /. See chapter 17 of the *Gretl User's Guide* for details.

See also ginv, invpd.

invcdf

Output: same type as input
Arguments: *d* (string)
\dots {} (see below)
u (scalar, series or matrix)

Inverse cumulative distribution function calculator. For a continuous distribution, returns x such that $P(X \leq x) = u$, for u in the interval 0 to 1. For a discrete distribution (Binomial or Poisson), returns the smallest x such that $P(X \leq x) \geq u$.

The distribution of X is determined by the string *d*. Between the arguments *d* and *u*, zero or more additional scalar arguments are required to specify the parameters of the distribution, as follows.

Distribution	*d*	*Arg* 2	*Arg* 3	*Arg* 4
Standard normal	`z`, `n` or `N`	–	–	–
Gamma	`g` or `G`	shape	scale	–
Student's *t* (central)	`t`	degrees of freedom	–	–
Chi square	`c`, `x` or `X`	degrees of freedom	–	–
Snedecor's *F*	`f` or `F`	df (num.)	df (den.)	–
Binomial	`b` or `B`	*p*	*n*	–
Poisson	`p` or `P`	λ	–	–
Laplace	`l` or `L`	mean	scale	–
Standardized GED	`E`	shape	–	–
Non-central χ^2	`ncX`	df	non-centrality	–
Non-central *F*	`ncF`	df (num.)	df (den.)	non-centrality
Non-central *t*	`nct`	df	non-centrality	–

See also cdf, critical, pvalue.

invmills

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the inverse Mills ratio at *x*, that is the ratio between the standard normal density and the complement to the standard normal distribution function, both evaluated at *x*.

This function uses a dedicated algorithm which yields greater accuracy compared to calculation using dnorm and cnorm, but the difference between the two methods is appreciable only for very large negative values of *x*.

See also cdf, cnorm, dnorm.

invpd

Output: square matrix
Arguments: *A* (positive definite matrix)
&logdet (reference to scalar, optional)

Returns the inverse of the symmetric, positive definite matrix *A*. This function is slightly faster than inv for large matrices, since no check for symmetry is performed; for that reason it should be used with care.

If the optional argument *&logdet* is present, the corresponding scalar will contain on successful exit the log determinant of *A*. This may be convenient to have in some cases, for example in the context of the evaluation of a Gaussian log-likelihood, because the log determinant is a by-product of the inversion algorithm and retrieving it via the *&logdet* argument avoids extra computations.

Note: if you're interested in the inversion of a matrix of the form $X'X$, where X is a large matrix, it is preferable to compute it via the prime operator `X'X` rather than using the more general syntax `X'*X`. The former expression uses a specialized algorithm which has the double advantage of being more efficient computationally and of ensuring that the result will be free by construction of machine precision artifacts that may render it numerically non-symmetric.

irf

Output: matrix
Arguments: *target* (integer)
shock (integer)
alpha (scalar between 0 and 1, optional)
sys (bundle, optional)

Provides estimated impulse response functions pertaining to a VAR or VECM, traced out over a certain forecast horizon. Without the final optional argument, this function works only when the last model estimated was a VAR or VECM. Alternatively, information on such a system can be saved as a bundle via the $system accessor and subsequently passed to `irf`.

The *target* and *shock* arguments take the form of 1-based indices of the endogenous variables in the system, with 0 taken to mean "all". The responses (expressed in the units of the *target* variable) are to an innovation of one standard deviation in the *shock* variable. If *alpha* is given a suitable positive value the estimates include a $1 - \alpha$ confidence interval (so, for example, give 0.1 for a 90 percent interval).

The following code fragment illustrates usage. In the first example the matrix `ir1` holds the responses of `y1` to innovations in each of `y1`, `y2` and `y3` (point estimates only since *alpha* is omitted). In the second, `ir2` holds the responses of all targets to an innovation in `y2`, with 90 percent confidence intervals. In this case the returned matrix will have 9 columns: each response path occupies 3 adjacent columns giving point estimate, lower bound and upper bound. The last example produces a matrix with 27 columns: 3 per response for each target times each shock.

```
var 4 y1 y2 y3
matrix ir1 = irf(1, 0)
matrix ir2 = irf(0, 2, 0.1)
matrix ir3 = irf(0, 0, 0.1)
```

The number of periods (rows) over which the response is traced is determined automatically based on the frequency of the data, but this can be overridden via the set command, as in `set horizon 10`.

When confidence intervals are produced they are derived via bootstrapping, with resampling of the original residuals. It is assumed that the lag order of the VAR or VECM is sufficient to eliminate serial correlation of the residuals. By default the number of bootstrap replications is 1999, but that can be adjusted via set, as in

```
set boot_iters 2999
```

See also fevd, vma.

irr

Output: scalar
Argument: *x* (series or vector)

Returns the Internal Rate of Return for *x*, considered as a sequence of payments (negative) and receipts (positive). See also npv.

iscomplex

Output: scalar
Argument: *name* (string)

Tests whether *name* is the identifier for a complex matrix. The return value is one of the following:

NA: *name* does not identify a matrix.

0: *name* identifies a real matrix, composed entirely of regular floating-point numbers ("doubles", in C parlance).

1: *name* identifies a "nominally" complex matrix, composed of numbers with both a real and an imaginary part, but in which all imaginary parts are zero.

2: the matrix in question holds at least one "genuinely" complex value, with a non-zero imaginary part.

isconst

Output:	integer
Arguments:	*y* (series or vector)
	panel-code (integer, optional)

Without the optional second argument, returns 1 if *y* has a constant value over the current sample range (or over its entire length if *y* is a vector), otherwise 0.

The second argument is accepted only if the current dataset is a panel and *y* is a series. In that case a *panel-code* value of 0 calls for a check for time-invariance, while a value of 1 means check for cross-sectional invariance (that is, in each time period the value of *y* is the same for all groups).

If *y* is a series, missing values are ignored in checking for constancy.

isdiscrete

Output:	integer
Argument:	*name* (string)

If *name* is the identifier for a currently defined series, returns 1 if the series is marked as discrete-valued, otherwise 0. If *name* does not identify a series, returns NA.

isdummy

Output:	integer
Argument:	*x* (series or vector)

If all the values contained in *x* are 0 or 1 (or missing), returns the number of ones, otherwise 0.

isnan

Output:	same type as input
Argument:	*x* (scalar or matrix)

Given a scalar argument, returns 1 if *x* is "Not a Number" (NaN), otherwise 0. Given a matrix argument, returns a matrix of the same dimensions with 1s in positions where the corresponding element of the input is NaN and 0s elsewhere.

isoconv

Output:	integer
Arguments:	*date* (series)
	&year (reference to series)
	&month (reference to series)
	&day (reference to series, optional)

Given a series *date* holding dates in ISO 8601 "basic" format (YYYYMMDD), this function writes the year, month and (optionally) day components into the series named by the second and subsequent arguments. An example call, assuming the series `dates` contains suitable 8-digit values:

```
series y, m, d
isoconv(dates, &y, &m, &d)
```

The nominal return value is 0 on successful completion; in case of failure an error is flagged.

isocountry

Output:	same type as input
Arguments:	*source* (string or array of strings)
	output (integer, optional)

This function maps between the four designations for countries present in ISO 3166, namely

1. Country name
2. Alpha-2 code (two uppercase letters)
3. Alpha-3 code (three uppercase letters)
4. Numeric code (3 digits)

Given a country's designation in one form, the return value is its designation in the form (1 to 4) selected by the optional *output* argument or, if this argument is omitted, a default conversion as follows: when *source* is a country name the return value is the country's 2-letter code; otherwise the return value is the country name. Various valid calls are illustrated below in interactive form.

```
? eval isocountry("Bolivia")
BO
? eval isocountry("Bolivia", 3)
BOL
? eval isocountry("GB")
United Kingdom of Great Britain and Northern Ireland
? eval isocountry("GB", 3)
GBR
? strings S = defarray("ES", "DE", "SD")
? strings C = isocountry(S)
? print C
Array of strings, length 3
[1] "Spain"
[2] "Germany"
[3] "Sudan"
? matrix m = {4, 840}
? C = isocountry(m)
? print C
Array of strings, length 2
[1] "Afghanistan"
[2] "United States of America"
```

When *source* is in form 4 (numeric code) this can be given as a string or array of strings (for example, "032" for Argentina) or in numeric form. In the latter case *source* may be given as a series or vector, though an error will be flagged if any of the numbers are out of the range 0 to 999.

In all cases (even when output form 4 is selected) a string, or array of strings, is returned; if numeric values are required these may be obtained using atof. If *source* is not matched by any entry in the ISO 3166 table the return value is an empty string, in which case a warning is printed.

isodate

Output:	see below
Arguments:	*ed* (scalar, series or matrix)
	as-string (boolean, optional)

The argument *ed* is interpreted as an epoch day, which equals 1 for the first of January in the year 1 AD on the proleptic Gregorian calendar. The default return value (of the same type as *ed*) is an 8-digit number, or a series of such numbers, on the pattern YYYYMMDD (ISO 8601 "basic" format), giving the Gregorian calendar date corresponding to the epoch day.

If the optional second argument *as-string* is non-zero, the return value is not numeric but rather a string on the pattern YYYY-MM-DD (ISO 8601 "extended" format), or a string-valued series if *ed* is a series, or an array of strings if *ed* is a vector. For a more flexible means of obtaining string representations of epoch days, see strfday.

For the inverse function, see epochday; also see juldate.

isoweek

Output:	see below
Arguments:	*year* (scalar or series)
	month (scalar or series)
	day (scalar or series)

Returns the ISO 8601 week number corresponding to the date(s) specified by the three arguments, or NA if the date is invalid. Note that all three arguments must be of the same type, either scalars (integers) or series.

ISO weeks are numbered from 01 to 53; most years have 52 weeks but on average 71 out of 400 years have 53 weeks. The ISO 8601 definition for week 01 is the week containing the year's first Thursday on the Gregorian calendar. For a full account see https://en.wikipedia.org/wiki/ISO_week_date.

An alternative call is also supported: if a single argument is given, it is taken to be a date (or series of dates) in ISO 8601 "basic" numeric format, YYYYMMDD. So the following two calls produce the same result, namely 13.

```
eval isoweek(2022, 4, 1)
eval isoweek(20220401)
```

iwishart

Output:	matrix
Arguments:	*S* (symmetric matrix)
	v (integer)

Given S (a positive definite $p \times p$ scale matrix), returns a drawing from the Inverse Wishart distribution with v degrees of freedom, where v must not be smaller than p. The returned matrix is also $p \times p$. The algorithm of Odell and Feiveson (1966) is used.

jsonget

Output:	string
Arguments:	*buf* (string)
	path (string)
	&nread (reference to scalar, optional)

The argument *buf* should be a JSON buffer, as may be retrieved from a suitable website via the curl function, and the *path* argument should be a JsonPath specification.

This function returns a string representing the data found in the buffer at the specified path. Data types of double (floating-point), int (integer) and string are supported. In the case of doubles or ints, their string representation is returned (using the "C" locale for doubles). If the object to which *path* refers is an array, the members are printed one per line in the returned string.

By default an error is flagged if *path* is not matched in the JSON buffer, but this behavior is modified if you pass the third, optional argument: in that case the argument retrieves a count of the matches and an empty string is returned if there are none. Example call:

```
ngot = 0
ret = jsonget(jbuf, "$.some.thing", &ngot)
```

However, an error is still flagged in case of a malformed query.

An accurate account of JsonPath syntax can be found at http://goessner.net/articles/JsonPath/. However, please note that the back-end for `jsonget` is provided by `json-glib`, which does not necessarily support all elements of JsonPath. Moreover, the exact functionality of `json-glib` may differ depending on the version you have on your system. See https://wiki.gnome.org/Projects/JsonGlib if you need details.

That said, the following operators should be available to `jsonget`:

- root node, via the `$` character
- recursive descent operator: `..`
- wildcard operator: *
- subscript operator: `[]`
- set notation operator, for example `[i,j]`
- slice operator: `[start:end:step]`

jsongetb

Output:	bundle
Arguments:	*buf* (string)
	path (string, optional)

The argument *buf* should be a JSON buffer, as may be retrieved from a suitable website via the curl function. The specification and effect of the optional *path* argument are described below.

The return value is a bundle whose structure basically mirrors that of the input: JSON objects become gretl bundles and JSON arrays become gretl arrays, each of which can hold strings, bundles or arrays. JSON "value" nodes become either members of bundles or elements of arrays; in the latter case numerical values are converted to strings using `sprintf`. Note that although the JSON specification allows arrays of mixed type these cannot be handled by `jsongetb` since gretl arrays must be of a single type.

The *path* argument can be used to limit the JSON elements included in the returned bundle. This is not a "JsonPath" as described in the help for jsonget; it is a simple construct subject to the following specification.

- *path* is a slash-separated array of elements where slash ("/") indicates moving to one level "deeper" in the JSON tree represented by *buf*. A leading slash is allowed but not required; implicitly the path always starts at the root. No extraneous white-space characters should be included.
- Each slash-separated element must take one of the following forms: (a) a single name, in which case only a JSON element whose name matches at the given structural level will be included; or (b) "*" (asterisk), in which case all elements at the given level are included; or (c) an array of comma-separated names, enclosed in braces ("" and ""), in which case only JSON elements whose names match one of the given names will be included.

See also the string-oriented jsonget; depending on your purpose one of these functions may be more helpful than the other.

juldate

Output:	see below
Arguments:	*ed* (scalar, series or matrix)
	as-string (boolean, optional)

This function works just like isodate except that on output the dates are relative to the Julian calendar rather than the Gregorian.

kdensity

Output:	matrix
Arguments:	*x* (series, list or matrix)
	scale (scalar, optional)
	control (boolean, optional)

Computes a kernel density estimate (or set of estimates) for the argument *x*, which may be a single series or vector or a list or matrix with more than column. The returned matrix has $k + 1$ columns, where k is the number of elements (series or columns) in *x*. The first column holds a set of evenly spaced abscissae and the rest hold the estimated density or densities at each of these points.

The formula used to compute the estimated density at each reference point, x, is

$$f(x) = (1/nh) \sum_{t-1}^{n} k\left((x - x_t)/h\right)$$

where n denotes the number of data points, h is a "bandwidth" parameter, and $k()$ is the kernel function. The larger the value of the bandwidth parameter, the smoother the estimated density.

The optional *scale* parameter can be used to adjust the bandwidth relative to the default of 1.0, which corresponds to the rule of thumb proposed by Silverman (1986), namely

$$h = 0.9\text{min}(s, \text{IQR}/1.349)n^{-1/5}$$

where s denotes the standard deviation of the data and IQR is the inter-quartile range. The *control* parameter acts as a boolean: 0 (the default) means that the Gaussian kernel is used; a non-zero value switches to the Epanechnikov kernel.

A plot of the results may be obtained using the gnuplot command, as illustrated below. Note that the column containing the abscissae should come last for plotting.

```
matrix d = kdensity(x)
# if x has a single element
gnuplot 2 1 --matrix=d --with-lines --fit=none
# if x has two elements
gnuplot 2 3 1 --matrix=d --with-lines --fit=none
```

kdsmooth

Output:	integer
Arguments:	*&kb* (reference to bundle)
	MSE (boolean, optional)

Performs disturbance smoothing for a Kalman bundle previously set up by means of ksetup and returns 0 on successful completion or non-zero if numerical problems are encountered. The return value should be checked before making using of results.

On successful completion, the smoothed disturbances will be available as `kb.smdist`.

The optional *MSE* argument determines the contents of the `kb.smdisterr` key. If 0 or omitted, this matrix will contain the unconditional standard errors of the smoothed disturbances, which are normally used to compute the so-called *auxiliary residuals*. Otherwise, `kb.smdisterr` will contain the estimated root mean square deviations of the auxiliary residuals from their true value.

For more details see chapter 36 of the *Gretl User's Guide*.

See also ksetup, kfilter, ksmooth, ksimul.

kfilter

Output:	scalar
Argument:	*&kb* (reference to bundle)

Performs a forward, filtering pass on a Kalman bundle previously set up by means of ksetup and returns 0 on successful completion or 1 if numerical problems are encountered.

On successful completion, the one-step-ahead prediction errors will be available as `kb.prederr` and the sequence of their covariance matrices as `kb.pevar`. Moreover, the key `kb.llt` gives access to a T-vector containing the log-likelihood by observation.

For more details see chapter 36 of the *Gretl User's Guide*.

See also kdsmooth, ksetup, ksmooth, ksimul.

kmeier

Output:	matrix
Arguments:	*d* (series or vector)
	cens (series or vector, optional)

Given a sample of duration data, *d*, possibly accompanied by a record of censoring status, *cens*, computes the Kaplan–Meier nonparametric estimator of the survival function (Kaplan and Meier (1958)). The returned matrix has three columns holding, respectively, the sorted unique values in *d*, the estimated survival function corresponding to the duration value in column 1 and the (large sample) standard error of the estimator, calculated via the method of Greenwood (1926).

If the *cens* series is given, the value 0 is taken to indicate an uncensored observation while a value of 1 indicates a right-censored observation (that is, the period of observation of the individual in question has ended before the duration or spell has been recorded as terminated). If *cens* is not given, it is assumed that all observations are uncensored. (Note: the semantics of *cens* may be extended at some point to cover other types of censoring.)

See also naalen.

kpsscrit

Output:	matrix
Arguments:	*T* (scalar)
	trend (boolean)

Returns a row vector containing critical values at the 10, 5 and 1 percent levels for the KPSS test for stationarity of a time series. *T* should give the number of observations and *trend* should be 1 if the test includes a trend, 0 otherwise.

The critical values given are based on response surfaces estimated in the manner set out by Sephton (1995). See also the kpss command.

ksetup

Output:	bundle
Arguments:	*Y* (series, matrix or list)
	Z (scalar or matrix)
	T (scalar or matrix)
	Q (scalar or matrix)
	R (matrix, optional)

Sets up a Kalman bundle, that is an object which contains all the information needed to define a linear state space model of the form

$$y_t = Z\alpha_t + u_t$$

where $\mathrm{Var}(u) = R$, and state transition equation

$$\alpha_{t+1} = T\alpha_t + v_t$$

where $\mathrm{Var}(v) = Q$.

Objects created via this function can be later used via the dedicated functions kfilter for filtering, ksmooth and kdsmooth for smoothing and ksimul for performing simulations.

The class of models that gretl can handle is in fact much wider than the one implied by the representation above: it is possible to have time-varying models, models with diffuse priors and exogenous variable in the measurement equation and models with cross-correlated innovations. For further details, see chapter 36 of the *Gretl User's Guide*.

See also kdsmooth, kfilter, ksmooth, ksimul.

ksimul

Output:	matrix
Arguments:	*&kb* (reference to bundle)
	U (matrix)
	extra (boolean, optional)

Uses a Kalman bundle previously set up by means of ksetup to perform simulation, the disturbances being taken from the matrix *U*. By default the returned matrix (which will have as many rows as *U*) contains simulated values of the observable(s), but if a non-zero value is given for *extra* the simulated state is also included. In the latter case each row holds the state first, then the observable(s).

For details see chapter 36 of the *Gretl User's Guide*.

See also ksetup, kfilter, ksmooth.

ksmooth

Output: integer
Argument: *&kb* (reference to bundle)

Performs a fixed-point smoothing (backward) pass on a Kalman bundle previously set up by means of ksetup and returns 0 on successful completion or non-zero if numerical problems are encountered. The return value should be checked before making using of results.

On successful completion, the smoothed states will be available as `kb.state` and the sequence of their covariance matrices as `kb.stvar`. For more details see chapter 36 of the *Gretl User's Guide*.

See also ksetup, kdsmooth, kfilter, ksimul.

kurtosis

Output: scalar
Argument: *x* (series)

Returns the excess kurtosis of the series *x*, skipping any missing observations.

lags

Output: list or matrix
Arguments: *p* (scalar or vector)
y (series, list or matrix)
bylag (boolean, optional)

If the first argument is a scalar, generates lags 1 to *p* of the series *y*, or if *y* is a list, of all series in the list, or if *y* is a matrix, of all columns in the matrix. If $p = 0$ and *y* is a series or list, the maximum lag defaults to the periodicity of the data; otherwise *p* must be positive.

If a vector is given as the first argument, the lags generated are those specified in the vector. Common usage in this case would be to give *p* as, for example, `seq(3,7)`, hence omitting the first and second lags. However, it is OK to give a vector with gaps, as in `{3,5,7}`, although the lags should always be given in ascending order.

In the case of list output, the generated variables are automatically named according to the template *varname* _ *i* where *varname* is the name of the original series and *i* is the specific lag. The original portion of the name is truncated if necessary, and may be adjusted in case of non-uniqueness in the set of names thus constructed.

When *y* is a list, or a matrix with more than one column, and the lag order is greater than 1, the default ordering of the terms in the return value is by variable: all lags of the first input series or column followed by all lags of the second, and so on. The optional third argument can be used to change this: if *bylag* is non-zero then the terms are ordered by lag: lag 1 of all the input series or columns, then lag 2 of all the series or columns, and so on.

See also mlag for use with matrices.

lastobs

Output: integer
Arguments: *y* (series)
insample (boolean, optional)

Returns the 1-based index of the last non-missing observation for the series *y*. By default the whole data range is examined, so if subsampling is in effect the value returned may be larger than the accessor $t2. But if a non-zero value is given for *insample* only the current sample range is considered. See also firstobs.

ldet

Output: scalar
Argument: *A* (square matrix)

Returns the natural log of the determinant of *A*, computed via the LU factorization. Note that this is more efficient than calling det and taking the log of the result. Moreover, in some cases `ldet` is able to return a valid result even if the determinant of *A* is numerically "infinite" (exceeds the C library's maximum double-precision number). See also rcond, cnumber.

ldiff

Output: same type as input
Argument: *y* (series or list)

Computes log differences; starting values are set to `NA`.

When a list is returned, the individual variables are automatically named according to the template `ld_`*varname* where *varname* is the name of the original series. The name is truncated if necessary, and may be adjusted in case of non-uniqueness in the set of names thus constructed.

See also diff, sdiff.

lincomb

Output: series
Arguments: *L* (list)
b (vector)

Computes a new series as a linear combination of the series in the list *L*. The coefficients are given by the vector *b*, which must have length equal to the number of series in *L*.

See also wmean.

linearize

Output: series
Argument: *x* (series)

Depends on having TRAMO installed. Returns a "linearized" version of the input series; that is, a series in which any missing values are replaced by interpolated values and outliers are adjusted. TRAMO's fully automatic mechanism is used; consult the TRAMO documentation for details.

Note that if the input series has no missing values and no values that TRAMO regards as outliers, this function will return a copy of the original series.

ljungbox

Output: scalar
Arguments: *y* (series)
p (integer)

Computes the Ljung–Box Q' statistic for the series *y* using lag order *p*, over the currently defined sample range. The lag order must be greater than or equal to 1 and less than the number of available observations.

This statistic may be referred to the chi-square distribution with *p* degrees of freedom as a test of the null hypothesis that the series *y* is not serially correlated. See also pvalue.

lngamma

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the log of the gamma function of *x*.

See also bincoeff and gammafun.

loess

Output:	series
Arguments:	*y* (series)
	x (series)
	d (integer, optional)
	q (scalar, optional)
	robust (boolean, optional)

Performs locally-weighted polynomial regression and returns a series holding predicted values of *y* for each non-missing value of *x*. The method is as described by Cleveland (1979).

The optional arguments *d* and *q* specify the order of the polynomial in *x* and the proportion of the data points to be used in local estimation, respectively. The default values are $d = 1$ and $q = 0.5$. The other acceptable values for *d* are 0 and 2. Setting $d = 0$ reduces the local regression to a form of moving average. The value of *q* must be greater than 0 and cannot exceed 1; larger values produce a smoother outcome.

If a non-zero value is given for the *robust* argument the local regressions are iterated twice, with the weights being modified based on the residuals from the previous iteration so as to give less influence to outliers.

See also nadarwat, and in addition see chapter 40 of the *Gretl User's Guide* for details on nonparametric methods.

log

Output:	same type as input
Argument:	*x* (scalar, series, matrix or list)

Returns the natural logarithm of *x*; produces `NA` for non-positive values. Note: `ln` is an acceptable alias for `log`.

When a list is returned, the individual variables are automatically named according to the template `l_`*varname* where *varname* is the name of the original series. The name is truncated if necessary, and may be adjusted in case of non-uniqueness in the set of names thus constructed.

Note that in case of matrix input the function acts element by element. For the matrix logarithm function, see mlog.

log10

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the base-10 logarithm of *x*; produces `NA` for non-positive values.

log2

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the base-2 logarithm of *x*; produces NA for non-positive values.

logistic

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the logistic CDF of the argument *x*, that is, $\Lambda(x) = 1/(1+e^{-x})$. If *x* is a matrix, the function is applied element by element.

lpsolve

Output: bundle
Argument: *specs* (bundle)

Solves a linear programming problem using the lpsolve library. See `gretl-lpsolve.pdf` for details and examples of usage.

lower

Output: square matrix
Argument: *A* (matrix)

Returns an $n \times n$ lower triangular matrix B for which $B_{ij} = A_{ij}$ if $i \geq j$, and 0 otherwise.

See also upper.

lrcovar

Output: matrix
Arguments: *A* (matrix)
demean (boolean, optional)

Returns the long-run variance-covariance matrix of the columns of *A*. The data are first demeaned unless the second (optional) argument is set to zero. The kernel type and lag truncation parameter (window size) can be chosen before calling this function with the HAC-related options that the set command offers, such as `hac_kernel`, `hac_lag`, `hac_prewhiten`. See also the section on Time series data and HAC covariance matrices in chapter 22 of the *Gretl User's Guide*.

See also lrvar.

lrvar

Output: scalar
Arguments: *y* (series or vector)
k (integer, optional)
mu (scalar, optional)

Returns the long-run variance of *y*, calculated using a Bartlett kernel with window size *k*. If the second argument is omitted, or given a negative value, the window size defaults to the integer part of the cube root of the sample size.

In formulae:

$$\hat{\omega}^2(k) = \frac{1}{T}\sum_{t=k}^{T-k}\left[\sum_{i=-k}^{k} w_i(y_t - \mu)(y_{t-i} - \bar{Y})\right]$$

with

$$w_i = 1 - \frac{|i|}{k+1}$$

For the variance calculation, the series *y* is centered around the optional parameter *mu*; if this is omitted or NA, the sample mean is used.

For a multivariate counterpart, see lrcovar.

Lsolve

Output:	matrix
Arguments:	*L* (matrix)
	B (matrix)

Solves for x in $AX = B$, where L is the lower triangular Cholesky factor of the positive definite matrix A, satisfying $LL' = A$. Suitable L can be obtained using the cholesky function with A as argument.

The following two calculations should produce the same result (up to machine precision), but the first variant allows for reuse of a precomputed Cholesky factor and so should be substantially faster if you are solving repeatedly for given A and several values of B. The speed-up will be greater, the greater the dimension of A.

```
# variant 1
matrix L = cholesky(A)
matrix X = Lsolve(L, B)
# variant 2
matrix X = A \ B
```

mat2list

Output:	list
Arguments:	*X* (matrix)
	prefix (string, optional)

A convenience function for making a list of series using the columns of a suitable matrix as input. The row dimension of *X* must equal either the length of the current dataset or the number of observations in the current sample range.

The naming of the series in the returned list proceeds as follows. First, if the optional *prefix* argument is supplied, the series created from column i of *X* is named by appending i to the given string, as in `myprefix1`, `myprefix2` and so on. Otherwise, if *X* has column names set (see cnameset) these names are used. Finally, if neither of the above conditions is satisfied, the names are `column1`, `column2` and so on. Note that this policy may result in overwriting existing series; if you don't want that to happen, take charge of naming the columns explicitly via `cnameset`, or supply *prefix*.

Here is an illustrative example of usage:

```
matrix X = mnormal($nobs, 8)
list L = mat2list(X, "xnorm")
# or alternatively, if you don't need X as such
list L = mat2list(mnormal($nobs, 8), "xnorm")
```

This will add to the dataset eight full-length series named `xnorm1`, `xnorm2` and so on.

max

Output:	depends on input
Arguments:	*x* (scalar, series or matrix)
	y (scalar, series or matrix, optional)

This function has two primary modes plus a special case.

The first mode is activated if a single argument of type scalar, series or matrix is given: the return value is a scalar, the maximum valid value "within" the argument: if *x* is a series, its maximum value within the current sample range, or if *x* is a matrix, its greatest element, missing values being ignored. The case of a scalar argument is supported for the sake of completeness; you just get its value back.

The second mode is activated if two arguments are given. The arguments *x* and *y* must be of the same type, and must be scalars, series or matrices (and if they are matrices, they must be of the same dimensions). The return value is an object of the same type as the arguments, holding the "between" or "cross" maximum or maxima. If the arguments are scalars you get the greater of the two; if they're series you get a series holding the greater of the values of the two series at each observation in the current sample range; if they're matrices you get a matrix holding the greater of their elements in each row and column. For each of the pairwise comparisons if either term is missing the result is also a missing value.

The special case

This arises if a single list argument is given. The return value is a series, containing at each observation in the current sample range the greatest of the values of the series in the list at that observation.

See also min.

maxc

Output:	row vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns a row vector containing the maxima of the columns of *X*. For columns containing `NA`s the result is also set to `NA`, unless the optional argument *skip_na* is nonzero, in which case the maximum valid entry will be returned.

See also imaxc, maxr, minc.

maxr

Output:	column vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns a column vector containing the maxima of the rows of *X*. For rows containing `NA`s the result is also set to `NA`, unless the optional argument *skip_na* is nonzero, in which case the maximum valid entry will be returned.

See also imaxr, maxc, minr.

mcorr

Output: matrix
Argument: *X* (matrix)

Computes a (Pearson) correlation matrix treating each column of *X* as a variable.

See also corr, cov, mcov.

mcov

Output: matrix
Arguments: *X* (matrix)
dfcorr (integer, optional)

Computes a covariance matrix treating each column of *X* as a variable. The divisor is $n - 1$, where n is the number of rows of *X*, unless the optional second argument is supplied, in which case $n -$ *dfcorr* is used.

See also corr, cov, mcorr.

mcovg

Output: matrix
Arguments: *X* (matrix)
u (vector, optional)
w (vector, optional)
p (integer)

Returns the matrix covariogram for a $T \times k$ matrix *X* (typically containing regressors), an (optional) T-vector *u* (typically containing residuals), an (optional) (p+1)-vector of weights *w*, and a lag order *p*, which must be greater than or equal to 0.

The returned matrix is given by

$$\sum_{j=-p}^{p} \sum_{j} w_{|j|} (X_t u_t u_{t-j} X'_{t-j})$$

If *u* is given as `null` the *u* terms are omitted, and if *w* is given as `null` all the weights are taken to be 1.0.

For example, the following piece of code

```
set seed 123
X = mnormal(6,2)
Lag = mlag(X,1)
Lead = mlag(X,-1)
print X Lag Lead
eval X'X
eval mcovg(X, , , 0)
eval X'(X + Lag + Lead)
eval mcovg(X, , , 1)
```

produces this output:

```
? print X Lag Lead
X (6 x 2)
```

```
  -0.76587      -1.0600
  -0.43188       0.30687
  -0.82656       0.40681
   0.39246       0.75479
   0.36875        2.5498
   0.28855      -0.55251

Lag (6 x 2)

    0.0000        0.0000
  -0.76587      -1.0600
  -0.43188       0.30687
  -0.82656       0.40681
   0.39246       0.75479
   0.36875        2.5498

Lead (6 x 2)

  -0.43188       0.30687
  -0.82656       0.40681
   0.39246       0.75479
   0.36875        2.5498
   0.28855      -0.55251
    0.0000        0.0000

? eval X'X
    1.8295        1.4201
    1.4201        8.7596

? eval mcovg(X,,, 0)
    1.8295        1.4201
    1.4201        8.7596

? eval X'(X + Lag + Lead)
    3.0585        2.5603
    2.5603        10.004

? eval mcovg(X,,, 1)
    3.0585        2.5603
    2.5603        10.004
```

mean

Output:	scalar or series
Arguments:	*x* (series or list)
	partial (boolean, optional)

If *x* is a series, returns the (scalar) sample mean, skipping any missing observations.

If *x* is a list, returns a series *y* such that y_t is the mean of the values of the variables in the list at observation *t*. By default the mean is recorded as NA if there are any missing values at *t*, but if you pass a non-zero value for *partial* any non-missing values will be used to form the statistic.

The following example illustrates the working of the function

```
open denmark.gdt
eval mean(LRM)
list L = dataset
```

```
eval mean(L)
```

The first call will return the scalar mean value (scalar) of the series LRM, and the second one returns a series.

See also median, sum, max, min, sd, var.

meanc

Output:	row vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the means of the columns of *X*. If a non-zero value is given for the optional second argument missing values are ignored, otherwise the result is NA for any columns that contain missing values.

For example, the following piece of code

```
matrix m = mnormal(5, 2)
m[1,2] = NA
print m
eval meanc(m)
```

produces this output:

```
? print m
m (5 x 2)

-0.098299          nan
   1.1829      -1.2817
  0.46037     -0.92947
   1.4896     -0.91970
  0.91918      0.47748

? eval meanc(m)
  0.79075          nan
```

See also meanr, sumc, maxc, minc, sdc, prodc.

meanr

Output:	column vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the means of the rows of *X*. If a non-zero value is given for the optional second argument missing values are ignored, otherwise the result is NA for any rows that contain missing values. See also meanc, sumr.

median

Output:	scalar or series
Argument:	*x* (series or list)

If x is a series, returns the (scalar) sample median, skipping any missing observations.

If x is a list, returns a series y such that y_t is the median of the values of the variables in the list at observation t, or NA if there are any missing values at t.

The following example illustrates the working of the function

```
set verbose off
open denmark.gdt
eval median(LRM)
list L = dataset
series m = median(L)
```

The first call will return the scalar median value (scalar) of the series LRM, and the second one returns a series.

See also mean, sum, max, min, sd, var.

mexp

Output:	square matrix
Argument:	A (square matrix)

Computes the matrix exponential,

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!} = \frac{I}{0!} + \frac{A}{1!} + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

(This series is sure to converge.) If A is a real matrix algorithm used is 11.3.1 from Golub and Van Loan (1996) is used. If A is complex the algorithm uses eigendecomposition and A must be diagonalizable.

See also mlog.

mgradient

Output:	matrix
Arguments:	*p* (integer)
	theta (vector)
	type (integer or string)

Analytical derivatives for MIDAS weights. Let k denote the number of elements in the vector of hyper-parameters, *theta*. This function returns a $p \times k$ matrix holding the gradient of the vector of weights (as calculated by mweights) with respect to the elements of *theta*. The first argument represents the desired lag order and the last argument specifies the type of parameterization. See mweights for an account of the acceptable *type* values.

See also midasmult, mlincomb, mweights.

midasmult

Output:	matrix
Arguments:	*mod* (bundle)
	cumulate (boolean)
	v (integer)

Computes MIDAS multipliers. The *mod* argument must be a bundle containing a MIDAS model, as the one produced by the midasreg command and accessible via the $model keyword. The function

returns a matrix with the implicit MIDAS multipliers for variable *v* in its first column and the corresponding standard errors in the second one. If the *cumulate* argument is nonzero, the multipliers are cumulated.

Note that the returned matrix is automatically endowed with appropriate row labels, so it is suitable to be used as the first argument to the modprint command. For example, the code

```
open gdp_midas.gdt
list dIP = ld_indpro*
smpl 1985:1 ;
midasreg ld_qgdp 0 ; mds(dIP, 0, 6, 2)
matrix ip_m = midasmult($model, 0, 1)
modprint ip_m
```

produces the following output:

```
            coefficient   std. error      z       p-value
  ------------------------------------------------------------
  dIP_0      0.343146      0.0957752     3.583     0.0003   ***
  dIP_1      0.402547      0.0834904     4.821     1.43e-06 ***
  dIP_2      0.176437      0.0673776     2.619     0.0088   ***
  dIP_3      0.0601876     0.0621927     0.9678    0.3332
  dIP_4      0.0131263     0.0259137     0.5065    0.6125
  dIP_5      0.000965260   0.00346703    0.2784    0.7807
  dIP_6      0.00000       0.00000      NA        NA
```

See also mgradient, mweights, mlincomb.

min

Output:	depends on input
Arguments:	*x* (scalar, series or matrix)
	y (scalar, series or matrix)

Please see the help for max; this function works in exactly the same way except that it returns a minimum or minima.

minc

Output:	row vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the minima of the columns of *X*. For columns containing NAs the result is also set to NA, unless the optional argument *skip_na* is nonzero, in which case the minimum valid entry will be returned.

See also iminc, maxc, minr.

minr

Output:	column vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the minima of the rows of *X*. For rows containing NAs the result is also set to NA, unless the optional argument *skip_na* is nonzero, in which case the minimum valid entry will be returned.

See also iminr, maxr, minc.

missing

Output: same type as input
Argument: *x* (scalar, series or list)

Returns a binary variable holding 1 if *x* is NA. If *x* is a series, the comparison is done element by element; if *x* is a list of series, the output is a series with 1 at observations for which at least one series in the list has a missing value, and 0 otherwise. For example, the following code

```
nulldata 3
series x = normal()
x[2] = NA
series x_ismiss = missing(x)
print x x_ismiss --byobs
```

sets a missing value at the second observation of *x*, and creates a new boolean series *x_ismiss* which identifies the missing observation

```
                y     y_ismiss

1     -1.551247            0
2                          1
3     -2.244616            0
```

See also misszero, ok, zeromiss.

misszero

Output: same type as input
Argument: *x* (scalar, series or matrix)

Converts NAs to zeros. If *x* is a series or matrix, the conversion is done element by element. For example, the following code

```
nulldata 3
series x = normal()
x[2] = NA
y = misszero(x)
print x y --byobs
```

sets a missing value at the second observation of *x*, and creates a new series *y* for which the missing observation is replaced by zero:

```
          x            y

1     0.7355250    0.7355250
2                      0.000
3    -0.2465936   -0.2465936
```

See also missing, ok, zeromiss.

mlag

Output:	matrix
Arguments:	*X* (matrix)
	p (scalar or vector)
	m (scalar, optional)

Shifts up or down the rows of *X*. If *p* is a positive scalar, the returned matrix *Y* has typical element $Y_{i,j} = X_{i-p,j}$ for $i \geq p$ and zero otherwise. In other words, the columns of *X* are shifted down by *p* rows and the first *p* rows are filled with the value *m*. If *p* is a negative number, *X* is shifted up and the last rows are filled with the value *m*. If *m* is omitted, it is understood to be zero.

If *p* is a vector the operation described above is carried out for each element in *p* and the resulting matrices are joined horizontally. The following code illustrates this usage, for input *X* with two columns and input *p* calling for lags 1 and 2. Missing values are set to NA as opposed to the default of 0.

```
matrix X = mnormal(5, 2)
print X
eval mlag(X, {1, 2}, NA)
```

```
m (5 x 2)

 1.5953    -0.070740
-0.52713    -0.47669
 -2.2056    -0.28112
 0.97753      1.4280
 0.49654     0.18532

     nan         nan         nan         nan
  1.5953   -0.070740         nan         nan
-0.52713    -0.47669      1.5953   -0.070740
 -2.2056    -0.28112    -0.52713    -0.47669
 0.97753      1.4280     -2.2056    -0.28112
```

See also lags.

mlincomb

Output:	series
Arguments:	*hfvars* (list)
	theta (vector)
	type (integer or string)

A convenience MIDAS function which combines lincomb with mweights. Given a list *hfvars*, it constructs a series which is a weighted sum of the elements of the list, the weights based on the vector of hyper-parameters *theta* and the type of parameterization: see `mweights` for details. Note that hflags is generally the best way to create a list suitable as the first argument to this function.

To be explicit, the call

```
series s = mlincomb(hfvars, theta, 2)
```

is equivalent to

```
matrix w = mweights(nelem(hfvars), theta, 2)
series s = lincomb(hfvars, w)
```

but use of `mlincomb` saves on some typing and also some CPU cycles.

mlog

Output:	square matrix
Argument:	*A* (square matrix)

Computes the matrix logarithm of *A*. The algorithm employed relies on eigendecomposition, which requires that *A* be diagonalizable. See also mexp.

mnormal

Output:	matrix
Arguments:	*r* (integer)
	c (integer, optional)

Returns a matrix with *r* rows and *c* columns, filled with standard normal pseudo-random variates. If omitted, the number of columns defaults to 1 (column vector). See also normal, muniform.

mols

Output:	matrix
Arguments:	*Y* (matrix)
	X (matrix)
	&U (reference to matrix, or `null`)
	&V (reference to matrix, or `null`)

Returns a $k \times n$ matrix of parameter estimates obtained by OLS regression of the $T \times n$ matrix Y on the $T \times k$ matrix X.

If the third argument is not `null`, the $T \times n$ matrix U will contain the residuals. If the final argument is given and is not `null` then the $k \times k$ matrix V will contain (a) the covariance matrix of the parameter estimates, if Y has just one column, or (b) $X'X^{-1}$ if Y has multiple columns.

By default, estimates are obtained via Cholesky decomposition, with a fallback to QR decomposition if the columns of X are highly collinear. The use of SVD can be forced via the command `set svd on`.

See also mpols, mrls.

monthlen

Output:	same type as input
Arguments:	*month* (scalar or series)
	year (scalar or series)
	weeklen (integer)

Returns the number of (relevant) days in the specified month in the specified year, on the proleptic Gregorian calendar. The *weeklen* argument, which must equal 5, 6 or 7, gives the number of days in the week that should be counted (a value of 6 omits Sundays, and a value of 5 omits both Saturdays and Sundays).

The return value is a scalar if both *month* and *year* are scalars, otherwise a series.

For example, if you have a monthly dataset open, the call

```
series wd = monthlen($obsminor, $obsmajor, 5)
```

will return a series containing the number of working days for each month in the sample.

movavg

Output:	series
Arguments:	*x* (series)
	p (scalar)
	control (integer, optional)
	y0 (scalar, optional)

Depending on the value of the parameter *p*, returns either a simple or an exponentially weighted moving average of the input series *x*.

If $p > 1$, a simple p-term moving average is computed, that is, $\frac{1}{p}\sum_{i=0}^{p-1} x_{t-i}$. If a non-zero value is supplied for the optional *control* parameter the MA is centered, otherwise it is "trailing". The optional *y0* argument is ignored.

If $0 < p < 1$, an exponential moving average is computed:

$$y_t = px_t + (1-p)y_{t-1}$$

This is the formula of Roberts (1959). By default the output series y is initialized using the first valid value of x, but the *control* parameter may be used to specify the number of initial observations that should be averaged to produce y_0. A zero value for *control* indicates that all the observations should be used. Alternatively, an initializer may be specified using the optional *y0* argument; in that case the *control* argument is ignored.

mpiallred

Output:	integer
Arguments:	*&object* (reference to object)
	op (string)

Available only when gretl is in MPI mode (see gretl + MPI). Must be called by all processes. This function works like mpireduce except that all processes, not just the root process, get a copy of the "reduced" object in place of the original. It is therefore equivalent to `mpireduce` followed by a call to mpibcast, but more efficient.

mpibarrier

Output: integer

Available only when gretl is in MPI mode (see gretl + MPI). Takes no arguments. Enforces synchronization of MPI processes: no process can continue beyond the barrier until it has been reached by all.

```
# nobody gets past until everyone gets here
mpibarrier()
```

mpibcast

Output:	integer
Arguments:	*&object* (reference to object)
	root (integer, optional)

Available only when gretl is in MPI mode (see gretl + MPI). Must be called by all processes. Broadcasts the *object* argument, which must be given in pointer form, to all processes. The object in question (a matrix, bundle, scalar, array, string or list) must be declared in all processes prior to the broadcast. No process can continue beyond a call to `mpibcast` until all processes have successfully executed it.

By default "root", the source of the broadcast, is the MPI process with rank 0, but this can be adjusted via the optional second argument, which must be an integer from 0 to the number of MPI processes minus 1.

A simple example follows. On successful completion every process will have a copy of the matrix `X` defined at rank 0.

```
matrix X
if $mpirank == 0
    X = mnormal(T, k)
endif
mpibcast(&X)
```

mpirecv

Output: object
Argument: *src* (integer)

Available only when gretl is in MPI mode (see gretl + MPI). See mpisend, with which `mpirecv` must always be paired, for an explanation. The *src* argument specifies the rank of the process from which the object is to be received, in the range 0 to the number of MPI processes minus 1.

mpireduce

Output: integer
Arguments: *&object* (reference to object)
op (string)
root (integer, optional)

Available only when gretl is in MPI mode (see gretl + MPI). Must be called by all processes. This function gathers objects (scalars, matrices or arrays) of a specified name, given in pointer form, from all processes and "reduces" them to a single object at the root node.

The `op` argument specifies the reduction operation or method. The methods supported for scalars are `sum`, `prod` (product), `max` and `min`. For matrices the methods are `sum`, `prod` (Hadamard product), `hcat` (horizontal concatenation) and `vcat` (vertical concatenation). For arrays only `acat` (concatenation) is supported.

By default "root", the target of the reduction, is the MPI process with rank 0, but this can be adjusted via the optional third argument, which must be an integer from 0 to the number of MPI processes minus 1.

An example follows. On successful completion of the above, the root process will have a matrix `X` which is the sum of the matrices `X` at all processes.

```
matrix X
X = mnormal(T, k)
mpireduce(&X, sum)
```

mpiscatter

Output:	integer
Arguments:	*&M* (reference to matrix)
	op (string)
	root (integer, optional)

Available only when gretl is in MPI mode (see gretl + MPI). Must be called by all processes. This function distributes chunks of a matrix in the root process to all processes. The matrix must be declared in all processes prior to the call to `mpiscatter`, and must be given in pointer form.

The `op` argument must be either `byrows` or `bycols`. Let q denote the quotient of the number of rows in the matrix to be scattered and the number of processes. In the `byrows` case root sends the first q rows to process 0, the next q to process 1, and so on. If there is a remainder from the division of rows it is added to the last allotment. The `bycols` case is exactly analogous but splitting of the matrix is by columns.

An example follows. If there are 4 processes, each one (including root) will each get a 2500×10 share of the original `X` as it existed in the root process. If you want to preserve the full matrix in the root process, it is necessary to make a copy of it before calling `mpiscatter`.

```
matrix X
if $mpirank == 0
    X = mnormal(10000, 10)
endif
mpiscatter(&X, byrows)
```

mpisend

Output:	integer
Arguments:	*object* (object)
	dest (integer)

Available only when gretl is in MPI mode (see gretl + MPI). Sends the named object (a matrix, bundle, array, scalar, string or list) from the current process to the one identified by the integer *dest* (from 0 to the number of MPI processes minus 1).

A call to this function must always be paired with a call to mpirecv in the *dest* process, as in the following example which sends a matrix from rank 2 to rank 3.

```
if $mpirank == 2
    matrix C = cholesky(A)
    mpisend(C, 3)
elif $mpirank == 3
    matrix C = mpirecv(2)
endif
```

mpols

Output:	matrix
Arguments:	*Y* (matrix)
	X (matrix)
	&U (reference to matrix, or `null`)

Works exactly as mols, except that the calculations are done in multiple precision using the GMP library.

By default GMP uses 256 bits for each floating point number, but you can adjust this using the environment variable GRETL_MP_BITS, e.g. GRETL_MP_BITS=1024.

mrandgen

Output:	matrix
Arguments:	*d* (string)
	p1 (scalar or matrix)
	p2 (scalar or matrix, conditional)
	p3 (scalar, conditional)
	rows (integer)
	cols (integer)
Examples:	`matrix mx = mrandgen(u, 0, 100, 50, 1)`
	`matrix mt14 = mrandgen(t, 14, 20, 20)`
	`matrix D = mrandgen(dir, {0.5,1,2,4}, 30)`

With one exception (see below), this function works like randgen except that the return value is a matrix rather than a series. The initial arguments to this function (the number of which depends on the selected distribution) are as described for `randgen`, but they must be followed by two integers to specify the row and column dimensions of the desired random matrix. If *p1* or *p2* are given in matrix form they must have a number of elements equal to the product of *rows* and *cols*.

The exceptional case is the Dirichlet distribution. This is a multivariate distribution, and invoking `mrandgen` with "dir" as first parameter triggers special syntax: the second argument must be a k-element positive vector a, and the third a scalar r. The function will return an $r \times k$ matrix where each row is an independent draw from a Dirichlet distribution with parameter a.

The first example above calls for a column vector of length 50 holding draws from a continuous uniform distribution on [0,100]. The second example specifies a 20×20 random matrix with draws from the t distribution with 14 degrees of freedom; and the third returns a 30×4 matrix holding 30 draws from a specified Dirichlet distribution.

See also mnormal, muniform.

mread

Output:	matrix
Arguments:	*fname* (string)
	import (boolean, optional)

Reads a matrix from a file named *fname*. If the file name does not contain a full path specification, it will be looked for in several "likely" locations, beginning with the currently set workdir. However, if a non-zero value is given for the optional *import* argument, the input file is looked for in the user's "dot" directory. This is intended for use with the matrix-exporting functions offered in the context of the foreign command. In this case the *fname* argument should be a plain filename, without any path component.

Currently, the function recognizes four file formats:

Native text format

These files are identified by the extension "`.mat`", and are fully compatible with the Ox matrix file format. If the filename has the suffix "`.gz`" it is assumed that gzip compression has been applied in writing the data.

The file is assumed to be plain text, conforming to the following specification:

- It starts with zero or more comments, defined as lines that start with the hash mark, #; such lines are ignored.
- The first non-comment line contains two integers, separated by a tab character, indicating the number of rows and columns, respectively.
- The columns are separated by tabs.
- The decimal separator is the dot character, ".".

Binary files

Files with the suffix ".bin" are assumed to be in binary format. The ".gz" suffix, for gzip compression, is also recognized. The first 19 bytes contain the characters gretl_binary_matrix, the next 8 bytes contain two 32-bit integers giving the number of rows and columns, and the remainder of the file contains the matrix elements as little-endian "doubles", in column-major order. If gretl is run on a big-endian system, the binary values are converted to little endian on writing, and converted to big endian on reading.

Delimited text files

If the name of the file to be read has extension ".csv" the rules governing the format of the file are different, and more relaxed. In this case the actual data should *not* be preceded by a line giving the number of rows and columns. Gretl will try to figure out the delimiter (comma, semicolon or space) and do its best to import the matrix, allowing for use of comma as decimal separator if need be. Note that the delimiter should not be the tab character, on pain of confusing such files with those in gretl's "native" matrix format.

Gretl dataset files

Files with extension ".gdt" or ".gdtb" are treated as gretl native data files, as created by the store command. In this case, the matrix returned contains the numerical values of the series of the dataset, arranged by column. Note that string-valued series are not read as such; the matrix will just contain their numeric encodings.

See also bread, mwrite.

mreverse

Output:	matrix
Arguments:	*X* (matrix)
	bycol (boolean, optional)

Returns a matrix containing the rows of *X* in reverse order, or the columns in reverse order if the optional second argument has a non-zero value.

mrls

Output:	matrix
Arguments:	*Y* (matrix)
	X (matrix)
	R (matrix)
	q (column vector)
	&U (reference to matrix, or null)
	&V (reference to matrix, or null)

Restricted least squares: returns a $k \times n$ matrix of parameter estimates obtained by least-squares regression of the $T \times n$ matrix Y on the $T \times k$ matrix X subject to the linear restriction $RB = q$, where B denotes the stacked coefficient vector. R must have kn columns; each row of this matrix represents a linear restriction. The number of rows in q must match the number of rows in R.

If the fifth argument is not `null`, the $T \times n$ matrix U will contain the residuals. If the final argument is given and is not `null` then the $k \times k$ matrix V will hold the restricted counterpart to the matrix $X'X^{-1}$. The variance matrix of the estimates for equation i can be constructed by multiplying the appropriate sub-matrix of V by an estimate of the error variance for that equation.

mshape

Output:	matrix
Arguments:	X (matrix)
	r (integer)
	c (integer, optional)

Rearranges the elements of X into a matrix with r rows and c columns. Elements are read from X and written to the target in column-major order. If X contains fewer than $k = rc$ elements, the elements are repeated cyclically; otherwise, if X has more elements, only the first k are used.

If the third argument is omitted, c defaults to 1 if X is 1×1 otherwise to N/r where N is the total number of elements in X. However, if N is not an integer multiple of r an error is flagged.

See also cols, rows, unvech, vec, vech.

msortby

Output:	matrix
Arguments:	X (matrix)
	j (integer)

Returns a matrix in which the rows of X are reordered by increasing value of the elements in column j. This is a stable sort: rows that share the same value in column j will not be interchanged.

msplitby

Output:	array of matrices
Arguments:	X (matrix)
	v (scalar or matrix)
	bycol (boolean)

Returns an array of matrices, the result of splitting X horizontally or vertically under the control of the arguments v and *bycol*. If *bycol* is nonzero, the matrix will be split by columns; otherwise, as per default, by rows.

The argument v can be either a vector or a scalar.

- vector: must be of length equal to the relevant (row or column) dimension of X, and must contain positive integers. The greatest integer sets the length of the array that is returned. Each element of v indicates the array index of the matrix to which the corresponding row of X should be assigned.
- scalar: the relevant dimension of X (row or column, as dictated by *bycol*) must be an exact multiple of the scalar value. X will be split in chunks with v rows or columns each.

In the following example we split a 4×3 matrix into three matrices: the first two rows are assigned to the first matrix; the second matrix is left empty; the third and fourth matrices gets row 3 and 4 of *X*, respectively

```
matrix X = {1,2,3; 4,5,6; 7,8,9; 10,11,12}
matrices M = msplitby(X, {1,1,3,4})
print M
```

The print statement gives

```
Array of matrices, length 4
[1] 2 x 3
[2] null
[3] 1 x 3
[4] 1 x 3
```

The next example splits *X* evenly:

```
matrix X = {1,2,3; 4,5,6; 7,8,9; 10,11,12}
matrices MM = msplitby(X, 2)
print MM[1]
print MM[2]
```

which gives

```
? print MM[1]
1   2   3
4   5   6

? print MM[2]
7    8    9
10   11   12
```

See flatten for the inverse operation.

muniform

Output:	matrix
Arguments:	*r* (integer)
	c (integer, optional)

Returns a matrix with *r* rows and *c* columns, filled with uniform (0,1) pseudo-random variates. If omitted, the number of columns defaults to 1 (column vector). Note: the preferred method for generating a scalar uniform r.v. is to use the randgen1 function.

See also mnormal, uniform.

mweights

Output:	matrix
Arguments:	*p* (integer)
	theta (vector)
	type (integer or string)

Returns a p-vector of MIDAS weights to be applied to p lags of a high-frequency series, based on the vector *theta* of hyper-parameters.

The *type* argument identifies the type of parameterization, which governs the required number of elements, k, in *theta*: 1 = normalized exponential Almon (k at least 1, typically 2); 2 = normalized beta with zero last ($k = 2$); 3 = normalized beta with non-zero last lag ($k = 3$); and 4 = Almon polynomial (k at least 1). Note that in the normalized beta case the first two elements of *theta* must be positive.

The *type* may be given as an integer code, as shown above, or by one of the following strings (respectively): `nealmon`, `beta0`, `betan`, `almonp`. If a string is used, it should be placed in double quotes. For example, the following two statements are equivalent:

```
W = mweights(8, theta, 2)
W = mweights(8, theta, "beta0")
```

See also mgradient, midasmult, mlincomb.

mwrite

Output:	integer
Arguments:	*X* (matrix)
	fname (string)
	export (boolean, optional)

Writes the matrix *X* to a file named *fname*. By default this file will be plain text; the first line will hold two integers, separated by a tab character, representing the number of rows and columns; on the following lines the matrix elements appear, in scientific notation, separated by tabs (one line per row). To avoid confusion on reading, files to be written in this format should be named with the suffix "`.mat`". See below for alternative formats.

If a file *fname* already exists, it will be overwritten. The nominal return value is 0 on successful completion; if writing fails an error is flagged.

The output file will be written in the currently set workdir, unless the *filename* string contains a full path specification. However, if a non-zero value is given for the *export* argument, the output file will be written into the user's "dot" directory, where it is accessible by default via the matrix-loading functions offered in the context of the foreign command. In this case a plain filename, without any path component, should be given for the second argument.

Matrices stored via the `mwrite` function in its default form can be easily read by other programs; see chapter 17 of the *Gretl User's Guide* for details.

Three mutually exclusive inflections of this function are available, as follows:

- If *fname* has the suffix "`.gz`" then the file is written in the format described above but with gzip compression.
- If *fname* has the suffix "`.bin`" then the matrix is written in binary format. In this case the first 19 bytes contain the characters `gretl_binary_matrix`, the next 8 bytes contain two 32-bit integers giving the number of rows and columns, and the remainder of the file contains the matrix elements as little-endian "doubles", in column-major order. If gretl is run on a big-endian system, the binary values are converted to little endian on writing, and converted to big endian on reading.
- If *fname* has the suffix "`.csv`" then the matrix is written in comma-separated format, without a header line indicating the number of rows and columns to follow. This may be easier for third-party programs to handle, but it is not recommended if the matrix file is intended for reading by gretl.

Note that if the matrix file is to be read by a third-party program it is not advisable to use the gzip or binary options. But if the file is intended for reading by gretl the alternative formats save space, and the binary format allows for much faster reading of large matrices. The gzip format is not recommended for very large matrices, since decompression can be quite slow.

See also mread. And for writing a matrix to file as a dataset, see store.

mxtab

Output:	matrix
Arguments:	*x* (series or vector)
	y (series or vector)

Returns a matrix holding the cross tabulation of the values contained in *x* (by row) and *y* (by column). The two arguments should be of the same type (both series or both column vectors). It is generally expected (though not required) that the arguments will be discrete-valued, with fewer distinct values than observations. Otherwise the cross-tabulation may be very large and not very informative.

See also values, corresp.

naalen

Output:	matrix
Arguments:	*d* (series or vector)
	cens (series or vector, optional)

Given a sample of duration data, *d*, possibly accompanied by a record of censoring status, *cens*, computes the Nelson–Aalen nonparametric estimator of the hazard function (Nelson (1972); Aalen (1978)). The returned matrix has three columns holding, respectively, the sorted unique values in *d*, the estimated cumulated hazard function corresponding to the duration value in column 1, and the standard error of the estimator.

If the *cens* series is given, the value 0 is taken to indicate an uncensored observation while a value of 1 indicates a right-censored observation (that is, the period of observation of the individual in question has ended before the duration or spell has been recorded as terminated). If *cens* is not given, it is assumed that all observations are uncensored. (Note: the semantics of *cens* may be extended at some point to cover other types of censoring.)

See also kmeier.

nadarwat

Output:	series
Arguments:	*y* (series)
	x (series)
	h (scalar, optional)
	LOO (boolean, optional)
	trim (scalar, optional)

Computes the Nadaraya–Watson nonparametric estimator of the conditional mean of *y* given *x*. The return value is a series holding $m(x_i)$, the estimate of $E(y_i|x_i)$ for each non-missing element of the series *x*.

$$m(x_i) = \frac{\sum_{j=1}^{n} y_j \cdot K_h(x_i - x_j)}{\sum_{j=1}^{n} K_h(x_i - x_j)}$$

where the kernel function $K_h(\cdot)$ is given by

$$K_h(x) = \exp\left(-\frac{x^2}{2h}\right)$$

for $|x| < \tau$ and zero otherwise. (τ = trimming parameter.)

The three optional arguments inflect the behavior of the estimator as described below.

Bandwidth

The argument *h* can be used to control the bandwidth, a positive real number. This is usually small; larger values of *h* make $m(x)$ smoother. A popular choice is to make *h* proportional to $n^{-0.2}$. If *h* is omitted or set to zero, the bandwidth defaults to a data-determined value using the proportionality just mentioned but incorporating the dispersion of the *x* data as measured by the inter-quartile range or standard deviation; see chapter 40 of the *Gretl User's Guide* for more details.

Leave-one-out

"Leave-one-out" is a variant of the algorithm which omits the *i*-th observation when evaluating $m(x_i)$. This makes the Nadaraya–Watson estimator more robust numerically and is generally advised when the estimator is computed for inference purposes. This variant is not enabled by default, but is activated if a non-zero value is given for the *LOO* argument.

In formulae, this estimator is

$$m(x_i) = \frac{\sum_{j\neq i} y_j \cdot K_h(x_i - x_j)}{\sum_{j\neq i} K_h(x_i - x_j)}$$

Trimming

The *trim* argument can be used to control the degree of "trimming", which is imposed to prevent numerical problems when the kernel function is evaluated too far away from zero. This parameter is expressed as a multiple of *h*, the default value being 4. In some cases a value greater than 4 may be preferable. Again see chapter 40 of the *Gretl User's Guide* for details.

See also loess.

nelem

Output: integer
Argument: *L* (list, matrix, bundle or array)

Returns the number of elements in the argument, which may be a list, a matrix, a bundle, an array or a string, but not a series. In the case of a string argument the number of bytes (which may not be equal to the number of characters in the string) is returned; see also strlen.

ngetenv

Output: scalar
Argument: *s* (string)

If an environment variable by the name of *s* is defined and has a numerical value, returns that value; otherwise returns NA. See also getenv.

nlines

Output: scalar
Argument: *buf* (string)

Returns a count of the complete lines (that is, lines that end with the newline character) in *buf*.

Example:

```
string web_page = readfile("http://gretl.sourceforge.net/")
scalar number = nlines(web_page)
print number
```

NMmax

Output:	scalar
Arguments:	*&b* (reference to matrix)
	f (function call)
	maxfeval (integer, optional)

Numerical maximization via the Nelder–Mead derivative-free simplex method. On input the vector *b* should hold the initial values of a set of parameters, and the argument *f* should specify a call to a function that calculates the (scalar) criterion to be maximized, given the current parameter values and any other relevant data. On successful completion, `NMmax` returns the maximized value of the criterion, and *b* holds the parameter values which produce the maximum.

The optional third argument may be used to set the maximum number of function evaluations; if it is omitted or set to zero the maximum defaults to 2000. As a special signal to this function the *maxfeval* value may be set to a negative number. In this case the absolute value is taken, and `NMmax` flags an error if the best value found for the objective function at the maximum number of function evaluations is not a local optimum. Otherwise non-convergence in this sense is not treated as an error.

If the object is in fact minimization, either the function call should return the negative of the criterion or alternatively `NMmax` may be called under the alias `NMmin`.

For more details and examples chapter 37 of the *Gretl User's Guide*. See also simann.

NMmin

Output: scalar

An alias for NMmax; if called under this name the function acts as a minimizer.

nobs

Output:	scalar or series
Argument:	*x* (series or list)

If *x* is a series, returns the number of non-missing observations for this series in the currently selected sample.

If *x* is a list, returns a series y such that y_t is the count of the series in the list that have a non-missing value at observation t.

See also pnobs, pxnobs.

normal

Output:	series
Arguments:	*$\mu $* (scalar)
	$\sigma $ (scalar)

Generates a series of Gaussian pseudo-random variates with mean μ and standard deviation σ. If no arguments are supplied, standard normal variates $N(0,1)$ are produced. The values are produced using the Ziggurat method (Marsaglia and Tsang, 2000).

See also randgen, mnormal, muniform.

normtest

Output:	matrix
Arguments:	*y* (series or vector)
	method (string, optional)

Carries out one or more tests for normality of *y*. By default the Doornik–Hansen test is performed but the optional *method* argument can be used to select an alternative: use `swilk` to get the Shapiro–Wilk test, `jbera` for Jarque–Bera test, or `lillie` for the Lilliefors test. Or give `all` for the *method* argument to carry out all four tests.

The second argument may be given in either quoted or unquoted form. In the latter case, however, if the argument is the name of a string variable the value of the variable is substituted.

The returned matrix is 1×2 for a single test, or 4×2 if all tests are performed. Test statistics are found in the first column and p-values in the second. The test statistic does not follow the same distribution is all cases. For Doornik–Hansen and Jarque–Bera it is chi-square(2); for the other methods it is an idiosyncratic statistic whose p-value requires special calculation.

See also the normtest command.

npcorr

Output:	matrix
Arguments:	*x* (series or vector)
	y (series or vector)
	method (string, optional)

Calculates a measure of correlation between *x* and *y* using a nonparametric method. If given, the third argument should be either `kendall` (for Kendall's tau, version b, the default method) or `spearman` (for Spearman's rho).

The return value is a 3-vector holding the correlation measure plus a test statistic and p-value for the null hypothesis of no correlation. Note that if the sample size is too small the test statistic and/or p-value may be `NaN` (not a number, or missing).

See also corr for Pearson correlation.

npv

Output:	scalar
Arguments:	*x* (series or vector)
	r (scalar)

Returns the Net Present Value of *x*, considered as a sequence of payments (negative) and receipts (positive), evaluated at annual discount rate *r*, which must be expressed as a decimal fraction, not a percentage (0.05 rather than 5%). The first value is taken as dated "now" and is not discounted. To emulate an NPV function in which the first value is discounted, prepend zero to the input sequence.

Supported data frequencies are annual, quarterly, monthly, and undated (undated data are treated as if annual).

See also irr.

NRmax

Output:	scalar
Arguments:	*&b* (reference to matrix)
	f (function call)
	g (function call, optional)
	h (function call, optional)

Numerical maximization via the Newton–Raphson method. On input the vector *b* should hold the initial values of a set of parameters, and the argument *f* should specify a call to a function that calculates the (scalar) criterion to be maximized, given the current parameter values and any other relevant data. If the object is in fact minimization, this function should return the negative of the criterion. On successful completion, `NRmax` returns the maximized value of the criterion, and *b* holds the parameter values which produce the maximum.

The optional third and fourth arguments provide means of supplying analytical derivatives and an analytical (negative) Hessian, respectively. The functions referenced by *g* and *h* must take as their first argument a predefined matrix that is of the correct size to contain the gradient or Hessian, respectively, given in pointer form. They also must take the parameter vector as an argument (in pointer form or otherwise). Other arguments are optional. If either or both of the optional arguments are omitted, a numerical approximation is used.

For more details and examples see chapter 37 of the *Gretl User's Guide*. See also BFGSmax, fdjac.

NRmin

Output: scalar

An alias for NRmax; if called under this name the function acts as a minimizer.

nullspace

Output:	matrix
Argument:	*A* (matrix)

Computes the right nullspace of *A*, via the singular value decomposition: the result is a matrix *B* such that

- $AB = [0]$, except when *A* has full column rank, in which case an empty matrix is returned. Otherwise, if *A* is $m \times n$, *B* will be an $n \times (n - r)$ matrix, where r is the rank of *A*.
- If *A* is not of full column rank, then the vertical concatenation of *A* and B' produces a full rank matrix.

Example:

```
A = mshape(seq(1,6),2,3)
B = nullspace(A)
C = A | B'

print A B C

eval A*B
eval rank(C)
```

Produces

```
? print A B C
A (2 x 3)

1   3   5
2   4   6

B (3 x 1)

-0.5
   1
-0.5

C (3 x 3)

   1      3      5
   2      4      6
-0.5      1   -0.5

? eval A*B
-4.4409e-16
-4.4409e-16

? eval rank(C)
3
```

See also rank, svd.

numhess

Output:	matrix
Arguments:	*b* (column vector)
	fcall (function call)
	d (scalar, optional)

Calculates a numerical approximation to the Hessian associated with the n-vector b and the objective function specified by the argument *fcall*. The function call should take b as its first argument (either straight or in pointer form), followed by any additional arguments that may be needed, and it should return a scalar result. On successful completion `numhess` returns an $n \times n$ matrix holding the Hessian, which is exactly symmetric by construction.

The method used is Richardson extrapolation, with four steps. The optional third argument can be used to set the fraction d of the parameter value used in setting the initial step size; if this argument is omitted the default is $d = 0.01$.

Here is an example of usage:

```
matrix H = numhess(theta, myfunc(&theta, X))
```

See also BFGSmax, fdjac.

obs

Output: series

Returns a series of consecutive integers, setting 1 at the start of the dataset. Note that the result is invariant to subsampling. This function is especially useful with time-series datasets. Note: you can write `t` instead of `obs` with the same effect.

See also obsnum.

obslabel

Output: string or array of strings
Argument: *t* (scalar or vector)

If *t* is a scalar, returns a single string, the observation label for observation *t*. The inverse function is provided by obsnum.

If *t* is a vector, returns an array of strings, the observation labels for the observations given by the elements of *t*.

In either case the *t* values must be integers, valid as 1-based indices of observations in the current dataset, otherwise an error is flagged.

obsnum

Output: integer
Argument: *s* (string)

Returns an integer corresponding to the observation specified by the string *s*. Note that the result is invariant to subsampling. This function is especially useful with time-series datasets. For example, the following code

```
open denmark
k = obsnum(1980:1)
```

yields `k = 25`, indicating that the first quarter of 1980 is the 25th observation in the `denmark` dataset.

See also obs, obslabel.

ok

Output: see below
Argument: *x* (scalar, series, matrix or list)

If *x* is a scalar, returns 1 if *x* is not `NA`, otherwise 0. If *x* is a series, returns a series with value 1 at observations with non-missing values and zeros elsewhere. If *x* is a list, the output is a series with 0 at observations for which at least one series in the list has a missing value, and 1 otherwise.

If *x* is a matrix the function returns a matrix of the same dimensions as *x*, with 1s in positions corresponding to finite elements of *x* and 0s in positions where the elements are non-finite (either infinities or not-a-number, as per the IEEE 754 standard).

See also missing, misszero, zeromiss. But note that these functions are not applicable to matrices.

onenorm

Output: scalar
Argument: *X* (matrix)

Returns the 1-norm of the $r \times c$ matrix X:

$$\|X\|_1 = \max_j \sum_{i=1}^{r} |X_{ij}|$$

See also infnorm, rcond.

ones

Output:	matrix
Arguments:	*r* (integer)
	c (integer, optional)

Outputs a matrix with r rows and c columns, filled with ones. If omitted, the number of columns defaults to 1 (column vector).

See also seq, zeros.

orthdev

Output:	series
Argument:	*y* (series)

Only applicable if the currently open dataset has a panel structure. Computes the forward orthogonal deviations for variable y, that is

$$\tilde{y}_{i,t} = \sqrt{\frac{T_i - t}{T_i - t + 1}} \left(y_{i,t} - \frac{1}{T_i - t} \sum_{s=t+1}^{T_i} y_{i,s} \right)$$

This transformation is sometimes used instead of differencing to remove individual effects from panel data. For compatibility with first differences, the deviations are stored one step ahead of their true temporal location (that is, the value at observation t is the deviation that, strictly speaking, belongs at $t - 1$). That way one loses the first observation in each time series, not the last.

See also diff.

pdf

Output:	same type as input
Arguments:	*d* (string)
	\dots {} (see below)
	x (scalar, series or matrix)
Examples:	`f1 = pdf(N, -2.5)`
	`f2 = pdf(X, 3, y)`
	`f3 = pdf(W, shape, scale, y)`

Probability density function calculator. Returns the density at x of the distribution identified by the code d. See cdf for details of the required (scalar) arguments. The distributions supported by the `pdf` function are the normal, Student's t, chi-square, F, Gamma, Beta, Exponential, Weibull, Laplace, Generalized Error, Binomial and Poisson. Note that for the Binomial and the Poisson what's calculated is in fact the probability mass at the specified point. For Student's t, chi-square, F the noncentral variants are supported too.

For the normal distribution, see also dnorm.

pergm

Output:	matrix
Arguments:	*x* (series or vector)
	bandwidth (scalar, optional)

If only the first argument is given, computes the sample periodogram for the given series or vector. If the second argument is given, computes an estimate of the spectrum of x using a Bartlett lag window of the given bandwidth, up to a maximum of half the number of observations ($T/2$).

Returns a matrix with two columns and $T/2$ rows: the first column holds the frequency, ω, from $2\pi/T$ to π, and the second the corresponding spectral density.

pexpand

Output:	series
Arguments:	*v* (vector)
	by_individual (boolean, optional)

Only applicable if the currently open dataset has a panel structure. By default, performs the inverse operation of pshrink. That is, given a vector of length equal to the number of individuals in the current panel sample, it returns a series in which each value is repeated T times, for T the time-series length of the panel. The resulting series is therefore non-time varying.

If a non-zero value is given for *by_individual*, the length of *v* should equal T and repetition is across the individuals in the panel.

pmax

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the current dataset has a panel structure. Returns a series holding the maxima of variable *y* for each cross-sectional unit (repeated for each time period).

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

See also pmin, pmean, pnobs, psd, pxsum, pshrink, psum.

pmean

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the current dataset has a panel structure. Computes the time-mean of variable *y* for each cross-sectional unit; that is,

$$\bar{y}_i = \frac{1}{T_i} \sum_{t=1}^{T_i} y_{i,t}$$

where T_i is the number of valid observations for unit i.

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

See also pmax, pmin, pnobs, psd, pxsum, pshrink, psum.

pmin

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the current dataset has a panel structure. Returns a series holding the minima of variable *y* for each cross-sectional unit (repeated for each time period).

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

See also pmax, pmean, pnobs, psd, pshrink, psum.

pnobs

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the current dataset has a panel structure. Returns a series holding the number of valid observations of variable *y* for each cross-sectional unit (repeated for each time period).

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

See also pmax, pmin, pmean, psd, pshrink, psum.

polroots

Output:	matrix
Argument:	*a* (vector)

Finds the roots of a polynomial. If the polynomial is of degree p, the vector *a* should contain $p + 1$ coefficients in ascending order, i.e. starting with the constant and ending with the coefficient on x^p.

The return value is a complex column vector of length p.

polyfit

Output:	series
Arguments:	*y* (series)
	q (integer)

Fits a polynomial trend of order q to the input series *y* using the method of orthogonal polynomials. The series returned holds the fitted values.

princomp

Output:	matrix
Arguments:	*X* (matrix)
	p (integer)
	covmat (boolean, optional)

Let the matrix X be $T \times k$, containing T observations on k variables. The argument p must be a positive integer less than or equal to k. This function returns a $T \times p$ matrix, P, holding the first p principal components of X.

The optional third argument acts as a boolean switch: if it is non-zero the principal components are computed on the basis of the covariance matrix of the columns of X (the default is to use the correlation matrix).

The elements of P are computed as

$$P_{tj} = \sum_{i=1}^{k} Z_{ti}\, v_i^{(j)}$$

where Z_{ti} is the standardized value (or just the centered value, if the covariance matrix is used) of variable i at observation t, $Z_{ti} = (X_{ti} - \bar{X}_i)/\hat{\sigma}_i$, and $v_i^{(j)}$ is the jth eigenvector of the correlation (or covariance) matrix of the X_is, with the eigenvectors ordered by decreasing value of the corresponding eigenvalues.

See also eigensym.

prodc

Output:	row vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the product of the elements of *X*, by column. If a non-zero value is given for the optional second argument missing values are ignored, otherwise the result is NA for any columns that contain missing values. Note that specifying *skip_na* is equivalent to treating missing values as if they were 1s.

See also prodr, meanc, sdc, sumc.

prodr

Output:	column vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the product of the elements of *X*, by row. If a non-zero value is given for the optional second argument missing values are ignored, otherwise the result is NA for any rows that contain missing values. Note that specifying *skip_na* is equivalent to treating missing values as if they were 1s.

See also prodc, meanr, sumr.

psd

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the current dataset has a panel structure. Computes the per-unit sample standard deviation for variable y, that is

$$\sigma_i = \sqrt{\frac{1}{T_i - 1} \sum_{t=1}^{T_i} (y_{i,t} - \bar{y}_i)^2}$$

The above formula holds for $T_i \geq 2$, where T_i is the number of valid observations for unit i; if $T_i = 0$, NA is returned; if $T_i = 1$, 0 is returned.

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

Note: this function makes it possible to check whether a given variable (say, X) is time-invariant via the condition `max(psd(X)) == 0`.

See also pmax, pmin, pmean, pnobs, pshrink, psum.

psdroot

Output:	square matrix
Arguments:	*A* (symmetric matrix)
	psdcheck (boolean, optional)

Performs a generalized variant of the Cholesky decomposition of the matrix A, which must be positive semidefinite (but may be singular). If the input matrix is not square an error is flagged, but symmetry is assumed and not tested; only the lower triangle of A is read. The result is a lower-triangular matrix L which satisfies $A = LL'$. Indeterminate elements in the solution are set to zero.

To force a check on the positive semidefiniteness of A, give a non-zero value for the optional second argument. In that case an error is flagged if the maximum absolute value of $A - LL'$ exceeds 1.0e-8. Such a check can also be performed manually:

```
L = psdroot(A)
chk = maxc(maxr(abs(A - L*L')))
```

For the case where A is positive definite, see cholesky.

pshrink

Output:	matrix
Argument:	*y* (series)

Only applicable if the current dataset has a panel structure. Returns a column vector holding the first valid observation for the series y for each cross-sectional unit in the panel, over the current sample range. If a unit has no valid observations for the input series it is skipped.

This function provides a means of compacting the series returned by functions such as pmax and pmean, in which a value pertaining to each cross-sectional unit is repeated for each time period.

See pexpand for the inverse operation.

psum

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

This function is applicable only if the current dataset has a panel structure. It computes the sum over time of variable y for each cross-sectional unit; that is,

$$S_i = \sum_{t=1}^{T_i} y_{i,t}$$

where T_i is the number of valid observations for unit i.

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

See also pmax, pmean, pmin, pnobs, psd, pxsum, pshrink.

pvalue

Output:	same type as input
Arguments:	*c* (character)
	\dots {} (see below)
	x (scalar, series or matrix)
Examples:	`p1 = pvalue(z, 2.2)`
	`p2 = pvalue(X, 3, 5.67)`
	`p2 = pvalue(F, 3, 30, 5.67)`

P-value calculator. Returns $P(X > x)$, where the distribution of X is determined by the character *c*. Between the arguments *c* and *x*, zero or more additional arguments are required to specify the parameters of the distribution; see cdf for details. The distributions supported by the `pvalue` function are the standard normal, *t*, Chi square, *F*, gamma, binomial, Poisson, Exponential, Weibull, Laplace and Generalized Error.

See also critical, invcdf, urcpval, imhof.

pxnobs

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the current dataset has a panel structure. Returns a series holding the number of valid observations of *y* in each time period (this count being repeated for each unit).

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

Note that this function works in a different dimension from the pnobs function.

pxsum

Output:	series
Arguments:	*y* (series)
	mask (series, optional)

Only applicable if the currently open dataset has a panel structure. Computes the cross-sectional sum for variable *y* in each period; that is,

$$\tilde{y}_t = \sum_{i=1}^{N} y_{i,t}$$

where *N* is the number of cross-sectional units.

If the optional second argument is provided then observations for which the value of *mask* is zero are ignored.

Note that this function works in a different dimension from the psum function.

qform

Output:	matrix
Arguments:	*x* (matrix)
	A (symmetric matrix)

Computes the quadratic form $Y = xAx'$. Using this function instead of ordinary matrix multiplication guarantees more speed and better accuracy, when A is a generic symmetric matrix. However, in the special case $A = I$, the simple expression `x'x` performs much better than `qform(x',I(rows(x))`.

In the special case when A is a diagonal matrix, the second argument can be given as a vector of the appropriate size, which is understood to contain the main diagonal of A. In this case, a more efficient algorithm is used.

If x and A are not conformable, or A is not symmetric, an error is returned.

qlrpval

Output:	scalar
Arguments:	*X2* (scalar)
	df (integer)
	p1 (scalar)
	p2 (scalar)

P-values for the test statistic from the QLR sup-Wald test for a structural break at an unknown point (see qlrtest), as per Hansen (1997).

The first argument, *X2*, denotes the (chi-square form of) the maximum Wald test statistic and *df* denotes its degrees of freedom. The third and fourth arguments represent, as decimal fractions of the overall estimation range, the starting and ending points of the central range of observations over which the successive Wald tests are calculated. For example if the standard approach of 15 percent trimming is adopted, you would set *p1* to 0.15 and *p2* to 0.85.

See also pvalue, urcpval.

qnorm

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns quantiles for the standard normal distribution. If x is not between 0 and 1, `NA` is returned. See also cnorm, dnorm.

qrdecomp

Output:	matrix
Arguments:	*X* (matrix)
	&R (reference to matrix, or `null`)
	&P (reference to matrix, or `null`)

Computes the "thin" QR decomposition of an $m \times n$ matrix X with $m >= n$, such that $X = QR$ where Q is an $m \times n$ orthogonal matrix and R is an $n \times n$ upper triangular matrix. The matrix Q is returned directly, while R can be retrieved via the optional second argument.

If the optional third argument is supplied the decomposition employs column pivoting, and on successful completion P holds the final ordering of the columns in the form of a row vector. If the columns are not in fact reordered P will compare equal to `seq(1, n)`.

See also eigengen, eigensym, svd.

quadtable

Output:	matrix
Arguments:	*n* (integer)
	type (integer, optional)
	a (scalar, optional)
	b (scalar, optional)

Returns an $n \times 2$ matrix for use with Gaussian quadrature (numerical integration). The first column holds the nodes or abscissae, the second the weights.

The first argument specifies the number of points (rows) to compute. The second argument codes for the type of quadrature: use 1 for Gauss–Hermite (the default); 2 for Gauss–Legendre; or 3 for Gauss–Laguerre. The significance of the optional parameters *a* and *b* depends on the selected *type*, as explained below.

Gaussian quadrature is a method of approximating numerically the definite integral of some function of interest. Let the function be represented as the product $f(x)W(x)$. The types of quadrature differ in the specification of the component $W(x)$: in the Hermite case we have $W(x) = \exp(-x^2)$; in the Laguerre case, $W(x) = \exp(-x)$; and in the Legendre case simply $W(x) = 1$.

For each specification of $W(x)$, one can compute a set of nodes, x_i, and weights, w_i, such that $\sum_{i=1}^{n} f(x_i)w_i$ approximates the desired integral. The method of Golub and Welsch (1969) is used.

When the Gauss–Legendre type is selected, the optional arguments *a* and *b* can be used to control the lower and upper limits of integration, the default values being -1 and 1. (In Hermite quadrature the limits are fixed at $-\infty$ and $+\infty$, while in the Laguerre case they are fixed at 0 and ∞.)

In the Hermite case *a* and *b* play a different role: they can be used to replace the default form of $W(x)$ with the (closely related) normal distribution with mean *a* and standard deviation *b*. Supplying values of 0 and 1 for these parameters, for example, has the effect of making $W(x)$ into the standard normal pdf, which is equivalent to multiplying the default x_i values by $\sqrt{2}$ and dividing the default w_i by $\sqrt{\pi}$.

quantile

Output:	scalar or matrix
Arguments:	*y* (series or matrix)
	p (scalar or vector)

If *y* is a series, returns the *p*-quantile for the series. For example, when $p = 0.5$, the median is returned.

If *y* is a matrix, returns a row vector containing the *p*-quantiles for the columns of *y*; that is, each column is treated as a series.

In addition, for matrix *y* an alternate form of the second argument is supported: *p* may be given as a vector. In that case the return value is an $m \times n$ matrix, where *m* is the number of elements in *p* and *n* is the number of columns in *y*.

Hyndman and Fan (1996) describe nine variant methods for calculating sample quantiles. The default method in gretl is the one they call Q_6 (which is also the default in Python). Method Q_7 (the default in R) or Q_8 (the one recommended by Hyndman and Fan) can be selected instead via the set command, as in

```
set quantile_type Q7 # or Q8
```

The *p*-quantile, Q_p, for a series y of length n is defined as:

$$Q_p = y_{[k]} + (h - k)(y_{[k+1]} - y_{[k]})$$

where k is the integer part of h, a term that differs by method—$h = (n+1)p$ for Q_6, $(n-1)p+1$ for Q_7 and $(n+1/3)p+1/3$ for Q_8—and $y_{[i]}$ is the i-th element of the series when sorted from smallest to largest.

For example, the code

```
set verbose off
matrix x = seq(1,7)'
set quantile_type Q6
printf "Q6: %g\n", quantile(x, 0.45)
set quantile_type Q7
printf "Q7: %g\n", quantile(x, 0.45)
set quantile_type Q8
printf "Q8: %g\n", quantile(x, 0.45)
```

produces the following output:

```
Q6: 3.6
Q7: 3.7
Q8: 3.63333
```

randgen

Output:	series
Arguments:	*d* (string)
	p1 (see below)
	p2 (scalar or series, conditional)
	p3 (scalar, conditional)
Examples:	`series x = randgen(u, 0, 100)`
	`series t14 = randgen(t, 14)`
	`series y = randgen(B, 0.6, 30)`
	`series g = randgen(G, 1, 1)`
	`series P = randgen(P, mu)`

All-purpose random number generator. The argument *d* is a string (in most cases just a single character) which specifies the distribution from which the pseudo-random numbers should be drawn. The arguments *p1* to *p3* specify the parameters of the selected distribution; the number of such parameters (and, in some cases, their nature) depends on the distribution.

For distributions other than the beta-binomial and the generic discrete, the parameters *p1* and (if applicable) *p2* may be given as either scalars or series: if they are given as scalars the output series is identically distributed, while if a series is given for *p1* or *p2* the distribution is conditional on the parameter value at each observation.

The two special cases have the following requirements:

- beta-binomial: all three parameters must be scalar.
- generic discrete: a single parameter is wanted, namely a k-vector whose elements represent the probabilities for an integer-valued random variable with support from 1 to k.

Specifics are given below: the string code for each distribution is shown in parentheses, followed by the interpretation of the arguments *p1* and, where applicable, *p2* and *p3*.

Distribution	*d*	*p1*	*p2*	*p3*
Uniform (continuous)	`u` or `U`	minimum	maximum	–
Uniform (discrete)	`i`	minimum	maximum	–
Normal	`z`, `n` or `N`	mean	standard deviation	–
Student's *t*	`t`	degrees of freedom	–	–
Chi square	`c`, `x` or `X`	degrees of freedom	–	–
Snedecor's *F*	`f` or `F`	df (num.)	df (den.)	–
Gamma	`g` or `G`	shape	scale	–
Binomial	`b` or `B`	*p*	*n*	–
Poisson	`p` or `P`	mean	–	–
Exponential	`exp`	scale	–	–
Logistic	`s`	location	scale	–
Weibull	`w` or `W`	shape	scale	–
Laplace	`l` or `L`	mean	scale	–
Generalized Error	`e` or `E`	shape	–	–
Beta	`beta`	shape1	shape2	–
Beta-Binomial	`bb`	*n*	shape1	shape2
Generic discrete	`disc`	**p**	–	–

See also normal, uniform, mrandgen, randgen1.

randgen1

Output:	scalar
Arguments:	*d* (character)
	p1 (scalar)
	p2 (scalar, conditional)
Examples:	`scalar x = randgen1(z, 0, 1)`
	`scalar g = randgen1(g, 3, 2.5)`

Works like randgen except that the return value is a scalar rather than a series.

The first example above calls for a value from the standard normal distribution, while the second specifies a drawing from the Gamma distribution with shape 3 and scale 2.5.

See also mrandgen.

randint

Output:	integer
Arguments:	*min* (integer)
	max (integer)

Returns a pseudo-random integer in the closed interval [*min*, *max*]. See also randgen.

randperm

Output:	vector
Arguments:	*n* (integer)
	k (integer, optional)

If only the first argument is given, returns a row vector containing a random permutation of the integers from 1 to *n*, without repetition of elements. If the second argument is given it must be a positive integer in the range 1 to *n*; in this case the function returns a row vector containing *k* integers selected randomly from 1 to *n* without replacement.

If you wish to sample *k* rows from a matrix `X` with *n* rows (without replacement), that can be accomplished as shown below:

```
matrix S = X[randperm(n, k),]
```

And if you wish to preserve the original order of the rows in the sample:

```
matrix S = X[sort(randperm(n, k)),]
```

See also resample for resampling with replacement.

randstr

Output: string
Argument: *n* (integer)

Returns a random string of length *n* bytes. The string includes the numerals `0` to `9` and the lowercase letters `a` to `f` with equal probability, and is interpretable as a hexadecimal integer. Intended usage is as a unique identifier. For example, with $n = 16$ the string will one of over 10^{19} possibilities and so unique with probability close to 1.

rank

Output: integer
Arguments: *X* (matrix)
tol (scalar, optional)

Returns the rank of the $r \times c$ matrix *X*, numerically computed via the singular value decomposition.

The result of this operation is the number of singular values of *X* that are found to be numerically greater than 0. The *tol* optional parameter can be used for tweaking this aspect. Singular values are considered to be non-zero if they are greater than mÖtolÖs, where m is the greater of r and c and s is the largest singular value. If the second argument is omitted *tol* is set to machine epsilon (see $macheps). In some cases, you may want to set *tol* to a larger value (eg 1.0e-9) in order to avoid overestimating the rank of *X*, which may lead to numerically unstable results.

See also svd.

ranking

Output: same type as input
Argument: *y* (series or vector)

Returns a series or vector with the ranks of y. The rank for observation i is the number of elements that are less than y_i plus one half the number of elements that are equal to y_i. (Intuitively, you may think of chess points, where victory gives you one point and a draw gives you half a point.) One is added so the lowest rank is 1 instead of 0.

Formally,

$$\text{rank}(y_i) = 1 + \sum_{j \neq i} \left[I(y_j < y_i) + 0.5 \cdot I(y_j = y_i) \right]$$

where I denotes the indicator function.

See also sort, sortby.

rcond

Output:	scalar
Argument:	A (square matrix)

Returns the reciprocal condition number for A with respect to the 1-norm. In many circumstances, this is a better measure of the sensitivity of A to numerical operations such as inversion than the determinant.

Given that A is non-singular, we may define

$$\kappa(A) = ||A||_1 \cdot ||A^{-1}||_1$$

This function returns $\kappa(A)^{-1}$.

See also det, ldet, onenorm.

Re

Output:	matrix
Argument:	C (complex matrix)

Returns a real matrix of the same dimensions as C, holding the real part of the input matrix. See also Im.

readfile

Output:	string
Arguments:	*fname* (string)
	codeset (string, optional)

If a file by the name of *fname* exists and is readable, returns a string containing the content of this file, otherwise flags an error. If *fname* does not contain a full path specification, it will be looked for in several "likely" locations, beginning with the currently set workdir. If the file in question is gzip-compressed, this is handled transparently.

If *fname* starts with the identifier of a supported internet protocol (`http://`, `ftp://` or `https://`), libcurl is invoked to download the resource. See also curl for more elaborate downloading operations.

If the text to be read is not encoded in UTF-8, gretl will try recoding it from the current locale codeset if that is not UTF-8, or from ISO-8859-15 otherwise. If this simple default does not meet your needs you can use the optional second argument to specify a codeset. For example, if you want to read text in Microsoft codepage 1251 and that is not your locale codeset, you should give a second argument of `"cp1251"`.

Examples:

```
string web_page = readfile("http://gretl.sourceforge.net/")
print web_page

string current_settings = readfile("@dotdir/.gretl2rc")
print current_settings
```

Also see the sscanf and getline functions.

regsub

Output:	same type as input
Arguments:	*s* (string, strings array or string-valued series)
	match (string)
	repl (string)

If *s* is a single string, returns a copy of *s* in which all occurrences of the pattern *match* are replaced using *repl*. The arguments *match* and *repl* are interpreted as Perl-style regular expressions. If *s* is an array of strings or string-valued series this operation is performed on each string in the array or series.

See also strsub for simple substitution of literal strings.

remove

Output:	integer
Argument:	*fname* (string)

If a file by the name of *fname* exists and is writable by the user, this function removes (deletes) the file and returns 0. If there is no such file or for some reason the file cannot be deleted, a non-zero error code is returned.

If *fname* does not specify a full path, it is taken to be relative to the current workdir.

replace

Output:	same type as input
Arguments:	*x* (series or matrix)
	find (scalar or vector)
	subst (scalar or vector)

Replaces each element of *x* equal to the *i*-th element of *find* with the corresponding element of *subst*.

If *find* is a scalar, *subst* must also be a scalar. If *find* and *subst* are both vectors, they must have the same number of elements. But if *find* is a vector and *subst* a scalar, then all matches will be replaced by *subst*.

Example:

```
a = {1,2,3;3,4,5}
find = {1,3,4}
subst = {-1,-8, 0}
b = replace(a, find, subst)
print a b
```

produces

```
a (2 x 3)

1   2   3
3   4   5

b (2 x 3)

-1    2   -8
-8    0    5
```

resample

Output:	same type as input
Arguments:	*x* (series or matrix)
	blocksize (integer, optional)
	draws (integer, optional)

The initial description of this function pertains to cross-sectional or time-series data; see below for the case of panel data.

Resamples from *x* with replacement. In the case of a series argument, each value of the returned series, y_t, is drawn from among all the values of x_t with equal probability. When a matrix argument is given, each row of the returned matrix is drawn from the rows of *x* with equal probability. See also randperm for sampling rows from a matrix without replacement.

The optional argument *blocksize* represents the block size for resampling by moving blocks. If this argument is given it should be a positive integer greater than or equal to 2. The effect is that the output is composed by random selection with replacement from among all the possible contiguous sequences of length *blocksize* in the input. (In the case of matrix input, this means contiguous rows.) If the length of the data is not an integer multiple of the block size, the last selected block is truncated to fit.

Number of draws

By default the number of resampled observations in the output is equal to that in the input—if *x* is a series, the length of the current sample range; if *x* is a matrix, its number of rows. In the matrix case *only* this can be adjusted via the optional third argument, which must be a positive integer. Note that if *blocksize* is greater than 1, *draws* refers to the number of individual observations, not the number of blocks.

Panel data

If the argument *x* is a series and the dataset takes the form of a panel, resampling by moving blocks is not supported. The basic form of resampling is supported, but has this specific interpretation: the data are resampled "by individual". Suppose you have a panel in which 100 individuals are observed over 5 periods. Then the returned series will again be composed of 100 blocks of 5 observations: each block will be drawn with equal probability from the 100 individual time series, with the time-series order preserved.

rgbmix

Output:	array of strings
Arguments:	*color1* (string)
	color2 (string)
	f (matrix)
	plot (boolean, optional)

Given two colors and a vector *f* of length *n* containing values in [0,1], this function returns an array of *n* strings, element *i* of which holds the hexadecimal RGB code for a mixture of the form $(1\text{-}f_i) \times$ `color1` $+ f_i \times$ `color2`. The weighted average is taken over the Red, Green and Blue channels of the input colors.

The color arguments can be specified by names known to gnuplot, or as hexadecimal values in the form `0xrrggbb` or `#rrggbb`. Hex values in the first of these forms may be given numerically, otherwise strings are needed. If a non-zero value is given for the *plot* argument, a plot that shows the color mixtures is produced.

This function offers a means of generating a set of related colors for plotting purposes, the primary use case being specification of multiple bands in a plot (for example, to indicate confidence intervals at more than one level). Three examples follow: the first produces successive lightenings of an initial blue; the second progressive darkenings of a pink shade; and the third a transition from red to yellow.

```
f = {0, 0.5, 0.75, 0.875, 0.9375}
mixes = rgbmix(0x1b43dc, "white", f, 1)
print mixes
f = {0, 0.1, 0.2, 0.3, 0.4}
rgbmix(0xefd0d3, "black", f, 1)
f = {0, 0.2, 0.4, 0.6, 0.8, 1}
rgbmix("red", "yellow", f, 1)
```

The output from the `print` command in the first example is

```
[1] "0x1b43dc"
[2] "0x8da1ee"
[3] "0xc6d0f6"
[4] "0xe2e8fb"
[5] "0xf1f3fd"
```

round

Output: same type as input
Argument: *x* (scalar, series or matrix)

Rounds to the nearest integer. Note that when x lies halfway between two integers, rounding is done "away from zero", so for example 2.5 rounds to 3, but `round(-3.5)` gives -4. This is a common convention in spreadsheet programs, but other software may yield different results. See also ceil, floor, int.

rnameget

Output: string or array of strings
Arguments: *M* (matrix)
r (integer, optional)

If the *r* argument is given, retrieves the name for row *r* of matrix *M*. If *M* has no row names attached the value returned is an empty string; if *r* is out of bounds for the given matrix an error is flagged.

If no second argument is given, retrieves an array of strings holding the row names from *M*, or an empty array if the matrix does not have row names attached.

Example:

```
matrix A = { 11, 23, 13 ; 54, 15, 46 }
rnameset(A, "First Second")
string name = rnameget(A, 2)
print name
```

See also rnameset.

rnameset

Output:	integer
Arguments:	*M* (matrix)
	S (array of strings or list)

Attaches names to the rows of the $m \times n$ matrix *M*. If *S* is a named list, the names are taken from the names of the listed series; the list must have m members. If *S* is an array of strings, it should contain m elements. A single string is also acceptable as the second argument; in that case it should contain m space-separated substrings.

The nominal return value is 0 on successful completion; in case of failure an error is flagged. See also cnameset.

Example:

```
matrix M = {1, 2; 2, 1; 4, 1}
strings S = array(3)
S[1] = "Row1"
S[2] = "Row2"
S[3] = "Row3"
rnameset(M, S)
print M
```

rows

Output:	integer
Argument:	*X* (matrix)

Returns the number of rows of the matrix *X*. See also cols, mshape, unvech, vec, vech.

schur

Output:	complex matrix
Arguments:	*A* (complex matrix)
	&Z (reference to matrix, or `null`)
	&w (reference to matrix, or `null`)

Performs the Schur decomposition of the complex matrix *A*, returning a complex upper triangular matrix *T*. If the second argument is given and is not `null` it retrieves a complex matrix *Z* holding the Schur vectors associated with *A* and *T*, such that $A = ZTZ^H$. If the third argument is given it retrieves the eigenvalues of *A* in a complex column vector.

sd

Output:	scalar or series
Arguments:	*x* (series or list)
	partial (boolean, optional)

If *x* is a series, returns the (scalar) sample standard deviation, skipping any missing observations.

If *x* is a list, returns a series y such that y_t is the sample standard deviation of the values of the series in the list at observation t. By default the standard deviation is recorded as `NA` if there are any missing values at t, but if you pass a non-zero value for *partial* any non-missing values will be used to form the statistic.

See also var.

sdc

Output:	row vector
Arguments:	*X* (matrix)
	df (scalar, optional)
	skip_na (boolean, optional)

Returns the standard deviations of the columns of *X*. If *df* is positive it is used as the divisor for the column variances, otherwise the divisor is the number of rows in *X* (that is, no degrees of freedom correction is applied). If a non-zero value is given for the optional third argument missing values are ignored, otherwise the result is NA for any columns that contain missing values.

See also meanc, sumc.

sdiff

Output:	same type as input
Argument:	*y* (series or list)

Computes seasonal differences: $y_t - y_{t-k}$, where k is the periodicity of the current dataset (see $pd or $panelpd). Starting values are set to NA.

When a list is returned, the individual variables are automatically named according to the template sd_*varname* where *varname* is the name of the original series. The name is truncated if necessary, and may be adjusted in case of non-uniqueness in the set of names thus constructed.

See also diff, ldiff.

seasonals

Output:	list
Arguments:	*baseline* (integer, optional)
	center (boolean, optional)

Applicable only if the dataset has a time-series structure with periodicity greater than 1. Returns a list of dummy variables coding for the period or season, named S1, S2 and so on.

The optional *baseline* argument can be used to exclude one period from the set of dummies. For example, if you give a baseline value of 1 with quarterly data the returned list will hold dummies for quarters 2, 3 and 4 only. If this argument is omitted or set to zero a full set of dummies is generated; if non-zero, it must be an integer from 1 to the periodicity of the data.

The *center* argument, if non-zero, calls for the dummies to be centered; that is, to have their population mean subtracted. For example, with quarterly data centered seasonals will have values −0.25 and 0.75 rather than 0 and 1.

With weekly data the precise effect depends on whether the data are dated or not. If they are dated, up to 53 seasonals are created, based on the ISO 8601 week number (see isoweek); if not, the maximum number of series is 52 (and over a long time span the "seasonals" will drift out of phase with the calendar year). In the dated weekly case, if you wish to create monthly seasonals this can be done as follows:

```
series month = $obsminor
list months = dummify(month)
```

See dummify for details.

selifc

Output:	matrix
Arguments:	*A* (matrix)
	b (row vector)

Selects from *A* only the columns for which the corresponding element of *b* is non-zero. *b* must be a row vector with the same number of columns as *A*.

See also selifr.

selifr

Output:	matrix
Arguments:	*A* (matrix)
	b (column vector)

Selects from *A* only the rows for which the corresponding element of *b* is non-zero. *b* must be a column vector with the same number of rows as *A*.

See also selifc, trimr.

seq

Output:	row vector
Arguments:	*a* (scalar)
	b (scalar)
	k (scalar, optional)

Given only two arguments, returns a row vector filled with values from *a* to *b* with an increment of 1, or a decrement of 1 if *a* is greater than *b*.

If the third argument is given, returns a row vector containing a sequence of values starting with *a* and incremented (or decremented, if *a* is greater than *b*) by *k* at each step. The final value is the largest member of the sequence that is less than or equal to *b* (or mutatis mutandis for *a* greater than *b*). The argument *k* must be positive.

See also ones, zeros.

setnote

Output:	integer
Arguments:	*b* (bundle)
	key (string)
	note (string)

Sets a descriptive note for the object identified by *key* in the bundle *b*. This note will be shown when the `print` command is used on the bundle. This function returns 0 on success or non-zero on failure (for example, if there is no object in *b* under the given *key*).

sgn

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the sign function of *x*; that is, 0 if *x* is zero, 1 if *x* is positive, −1 if *x* is negative, or `NA` if *x* is Not a Number.

simann

Output:	scalar
Arguments:	*&b* (reference to matrix)
	f (function call)
	maxit (integer, optional)

Implements simulated annealing, which may be helpful in improving the initialization for a numerical optimization problem.

On input the first argument holds the initial value of a parameter vector and the second argument specifies a function call which returns the (scalar) value of the maximand. The optional third argument specifies the maximum number of iterations (which defaults to 1024). On successful completion, `simann` returns the final value of the maximand and *b* holds the associated parameter vector.

For more details and an example see chapter 37 of the *Gretl User's Guide*. See also BFGSmax, NRmax.

sin

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the sine of *x*. See also cos, tan, atan.

sinh

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the hyperbolic sine of *x*.

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

See also asinh, cosh, tanh.

skewness

Output:	scalar
Argument:	*x* (series)

Returns the skewness value for the series *x*, skipping any missing observations.

sleep

Output:	scalar
Argument:	*ns* (scalar)

Not of any direct use for econometrics, but can be useful for testing parallelization methods. This function simply causes the current thread to "sleep"—that is, do nothing—for *ns* seconds. The argument must be non-negative. On wake-up, the function returns 0.

smplspan

Output:	scalar
Arguments:	*startobs* (string)
	endobs (string)
	pd (integer)

Returns the number of observations from *startobs* to *endobs* (inclusive) for time-series data with frequency *pd*.

The first two arguments should be given in the form preferred by gretl for annual, quarterly or monthly data—for example, `1970`, `1970:1` or `1970:01` for each of these frequencies, respectively—or as ISO 8601 dates, `YYYY-MM-DD`.

The *pd* argument must be 1, 4 or 12 (annual, quarterly, monthly); one of the daily frequencies (5, 6, 7); or 52 (weekly). If *pd* equals 1, 4 or 12, then ISO 8601 dates are acceptable for the first two arguments if they indicate the start of the period in question. For example, `2015-04-01` is acceptable in place of `2015:2` to represent the second quarter of 2015.

If you already have a dataset of frequency *pd* in place, with a sufficient range of observations, then the result of this function could easily be emulated using obsnum. The advantange of `smplspan` is that you can calculate the number of observations without having a suitable dataset (or any dataset) in place. An example follows:

```
scalar T = smplspan("2010-01-01", "2015-12-31", 5)
nulldata T
setobs 5 2010-01-01
```

This produces:

```
? scalar T = smplspan("2010-01-01", "2015-12-31", 5)
Generated scalar T = 1565
? nulldata T
periodicity: 1, maxobs: 1565
observations range: 1 to 1565
? setobs 5 2010-01-01
Full data range: 2010-01-01 - 2015-12-31 (n = 1565)
```

After the above, you can be confident that the last observation in the dataset created via nulldata will be `2015-12-31`. Note that the number 1565 would have been rather tricky to compute otherwise.

sort

Output:	same type as input
Argument:	*x* (series, vector or strings array)

Sorts *x* in ascending order. Observations with missing values are skipped if *x* is a series, but sorted to the end if *x* is a vector. See also dsort, values. For matrices specifically, see msortby.

sortby

Output:	series
Arguments:	*y1* (series)
	y2 (series)

Returns a series containing the elements of *y2* sorted by increasing value of the first argument, *y1*. See also sort, ranking.

sphericorr

Output:	matrix
Arguments:	*X* (matrix)
	mode (integer)
	&J (reference to matrix, or `null`)

Calculates the spherical coordinates representation of a correlation matrix, or its inverse, depending on the value of the *mode* parameter.

When *mode* is 0 or omitted, X is assumed to be an $n \times n$ correlation matrix. The returned value will be a vector with $n(n-1)/2$ elements between 0 and π. In this mode the reference to J is ignored.

When *mode* is 1 or 2 the inverse transformation is performed, so X must be a vector with $n(n-1)/2$ elements between 0 and π. The return value is the correlation matrix R if *mode* equals 1, or its Cholesky factor K if *mode* equals 2. The optional pointer to matrix J, if present, retrieves the Jacobian of vech(R) or vech(K) with respect to X.

Note that the spherical coordinates representation makes it very easy to compute the log-determinant of the correlation matrix R:

```
omega = sphericorr(X)
log_det = 2 * sum(log(sin(omega)))
```

sprintf

Output:	string
Arguments:	*format* (string)
	. . . (see below)

The returned string is constructed by printing the values of the trailing arguments, indicated by the dots above, under the control of *format*. It is meant to give you great flexibility in creating strings. The *format* is used to specify the precise way in which you want the arguments to be printed.

In general, *format* must be an expression that evaluates to a string, but in most cases will just be a string literal (an alphanumeric sequence surrounded by double quotes). Some character sequences in the format have a special meaning: those beginning with the percent character (for the items contained in the argument list; moreover, special characters such as the newline character are represented via a combination beginning with a backslash.

For example, the code below

```
scalar x = sqrt(5)
string claim = sprintf("sqrt(%d) is (roughly) %6.4f.\n", 5, x)
print claim
```

will output

```
sqrt(5) is (roughly) 2.2361.
```

The expression `%d` in the format string indicates that we want an integer at that place in the output; since it is the leftmost "percent" expression, it is matched to the first argument, that is 5. The

second special sequence is `%6.4f`, which stands for a decimal value at least 6 digits wide with 4 digits after the decimal separator. The number of such sequences must match the number of arguments following the format string.

See the help page for the printf command for more details about the syntax you can use in format strings.

sqrt

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the positive square root of *x*; produces `NA` for negative values.

Note that if the argument is a matrix the operation is performed element by element. For the "matrix square root" see cholesky.

square

Output:	list
Arguments:	*L* (list)
	cross-products (boolean, optional)

Returns a list that references the squares of the variables in the list *L*, named on the pattern `sq_`*varname*. If the optional second argument is present and has a non-zero value, the returned list also includes the cross-products of the elements of *L*; these are named on the pattern *var1_var2*. In these patterns the input variable names are truncated if need be, and the output names may be adjusted in case of duplication of names in the returned list.

Note that dummy variables will be skipped when computing squares to avoid producing an identical series, but their product (aka "interaction") with other series in the input list *L* will be computed.

sscanf

Output:	integer
Arguments:	*src* (string or array of strings)
	format (string)
	. . . (see below)

Reads values from *src* under the control of *format* and assigns these values to one or more trailing arguments, indicated by the dots above. Returns the number of values assigned. This is a simplified version of the `sscanf` function in the C programming language, with an extension to the scanning of an entire matrix; this extension is described under the leading "Scanning a matrix" below. Note that giving an array of strings as *src* is acceptable only in the case of matrix scanning.

src may be either a literal string, enclosed in double quotes, or the name of a predefined string variable. *format* is defined similarly to the format string in printf (more on this below). *args* should be a comma-separated list containing the names of predefined variables: these are the targets of conversion from *src*. (For those used to C: one can prefix the names of numerical variables with `&` but this is not required.)

Literal text in *format* is matched against *src*. Conversion specifiers start with `%`, and recognized conversions include `%f`, `%g` or `%lf` for floating-point numbers; `%d` for integers; `%s` for strings. You may insert a positive integer after the percent sign: this sets the maximum number of characters to read for the given conversion. Alternatively, you can insert a literal `*` after the percent to suppress the conversion (thereby skipping any characters that would otherwise have been converted for the given type). For example, `%3d` converts the next 3 characters in *src* to an integer, if possible; `%*g` skips as many characters in *src* as could be converted to a single floating-point number.

In addition to %s conversion for strings, a simplified version of the C format %*N*[*chars*] is available. In this format *N* is the maximum number of characters to read and *chars* is a set of acceptable characters, enclosed in square brackets: reading stops if *N* is reached or if a character not in *chars* is encountered. The function of *chars* can be reversed by giving a circumflex, ^, as the first character; in that case reading stops if a character in the given set is found. (Unlike C, the hyphen does not play a special role in the *chars* set.)

If the source string does not (fully) match the format, the number of conversions may fall short of the number of arguments given. This is not in itself an error so far as gretl is concerned. However, you may wish to check the number of conversions performed; this is given by the return value. Some simple examples follow:

```
# scanning scalar values
scalar x
scalar y
sscanf("123456", "%3d%3d", x, y)
# scanning string values
string s = "one two"
string s1
string s2
sscanf(s, "%s %s", s1, s2)
print s1 s2
```

Scanning a matrix

Matrix scanning must be signaled by the special conversion specification "%m". The maximum number of rows to be read can be specified by inserting an integer between the "%" sign and the "m" for matrix. Two variants are supported: *src* a single string representing a matrix, and *src* an array of strings. We describe these options in turn.

If *src* is a single string argument the scanner reads a line of input and counts the (space- or tab-separated) number of numeric fields. This defines the number of columns in the matrix. By default, reading then proceeds for as many lines (rows) as contain the same number of numeric columns, but the maximum number of rows can be limited via the optional integer value mentioned above.

If *src* is an array of strings the output is necessarily a column vector, each element of which is the numerical conversion of the corresponding string, or NA if the string is not numeric. Here are some simple examples.

```
# scanning a single string
string s = sprintf("1 2 3 4\n5 6 7 8")
print s
matrix m
sscanf(s, "%m", m)
print m
# scanning an array of strings
strings S = defarray("1.1", "2.2", "3.3", "4.4", "5.5")
sscanf(S, "%4m", m)
print m
```

sst

Output: scalar
Argument: *y* (series or vector)

Returns the sum of squared deviations from the mean for the non-missing observations in the series or vector *y*. See also var.

stack

Output:	series
Arguments:	*L* (list)
	n (integer)
	offset (integer, optional)

Designed for manipulation of data into the stacked time series format required by gretl for panel data. The return value is a series obtained by stacking "vertically" *n* observations from each series in the list *L*. By default the first *n* observations are used (corresponding to *offset* = 0) but the starting point can be shifted by supplying a positive value for *offset*. If the resulting series is longer than the existing dataset, observations are added as needed.

This function can handle the case where a data file holds side-by-side time series for a number of cross-sectional units, as well as the case where time runs horizontally and each row represents a cross-sectional unit.

See the section titled "Panel data specifics" in chapter 4 of the *Gretl User's Guide* for details and examples of usage.

stdize

Output:	same type as input
Arguments:	*X* (series, list or matrix)
	v (integer, optional)
	skip_na (boolean, optional)

By default, returns a standardized version of the series, list or matrix: the input is centered and divided by its sample standard deviation (with a degrees of freedom correction of 1). Results are computed by column in the case of matrix input.

The optional second argument can be used to inflect the result. A non-negative value of *v* sets the degrees of freedom correction used in the standard deviation, so $v = 0$ gives the maximum likelihood estimator. As a special case, if *v* equals -1 only centering is performed.

By default missing values are automatically skipped in the case of series or list input but not for matrix input. To have missing values ignored in the matrix case, supply a non-zero value for *skip_na*.

strfday

Output:	depends on input
Arguments:	*epoch_day* (scalar, series or matrix)
	format (string, optional)

This function works like strftime, converting from a numeric value to a string governed by *format*, except that the input is an "epoch day", for the definition of which see epochday. Since the resolution is daily, only date-related formats are handled; time-related formats give undefined results.

If the second argument is omitted the format defaults to ISO 8601 extended, `YYYY-MM-DD`.

strftime

Output:	depends on input
Arguments:	*tm* (scalar, series or matrix)
	format (string, optional)
	offset (scalar, optional)

The argument *tm* is taken to give "Unix time", the number of seconds since the start of the year 1970 according to UTC, and the return value is a string giving the corresponding date and/or time—either in a format specified via the optional second argument or, by default, the "preferred date and time representation for the current locale" as determined by the system C library. See below for more on the format specification.

The optional *offset* argument can be used to specify an offset in seconds relative to UTC, thus selecting a time zone other than the default, which is always local time. For example an offset of 3600 selects Central European Time, while 0 selects GMT. The absolute value of *offset* should not exceed 86400 (24 hours).

The specific type returned depends on that of *tm*: if *tm* is a scalar, vector, or series the output is, respectively, a single string, an array of strings, or a string-valued series.

Values of *tm* suitable for use with this function may be obtained via the $now accessor or the strptime function.

Note that while *tm* is taken as relative to UTC the output of this function is by default "local", relative to the time-zone setting on the host computer. A given *tm* will therefore show a different time, and perhaps a different date, in different time zones. But if you want a string representing UTC rather than local time, gretl can do that; see below.

Format options

The standard formatting options may be found by consulting the `strftime` manual page, on systems which have such pages, or via one of the many websites which present relevant information, such as https://devhints.io/strftime. In addition to the standard formats gretl recognizes a special option: if *format* is just "`8601`", date and time are shown in ISO 8601 format.

stringify

Output:	integer
Arguments:	*y* (series)
	S (array of strings)

Provides a means of defining string values for the series *y*. Two conditions must be satisfied for this to work: the target series must have nothing but integer values, none of them less than 1, and the array *S* must have at least *n* elements where *n* is the largest value in *y*. In addition each element of *S* must be valid UTF-8. If any of these conditions is not met, an error is flagged. The nominal return value is 0 on successful completion.

An alternative to `stringify` that may be useful in some contexts is direct assignment from an array of strings to a series: this creates a series whose values are taken from the array in sequence; the number of elements in the array must equal either the full length of the dataset or the length of the current sample range, and values may be repeated as required.

See also strvals, strvsort.

strlen

Output:	integer
Argument:	*s* (string or array of strings)

If *s* is a single string, returns the number of UTF-8 characters it contains. Note that this will be less than the number of bytes if the string contains any multi-byte (non-ASCII) characters. If you want the number of bytes you can use the nelem function. For example:

```
string s = "¡Olé!"
printf "strlen(s) = %d, nelem(s) = %d\n", strlen(s), nelem(s)
```

should return

```
strlen(s) = 5, nelem(s) = 7
```

If the argument is an array of strings the return value is a column vector holding the number of characters in each string. A string-valued series is also an acceptable argument: in this case the return value is a series holding the length of the string values over the current sample range.

strncmp

Output:	integer
Arguments:	*s1* (string)
	s2 (string)
	n (integer, optional)

Compares the two string arguments and returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*, up to the first *n* characters. If *n* is omitted the comparison proceeds as far as possible.

Note that if you just want to compare two strings for equality, that can be done without using a function, as in `if (s1 == s2) ...`

strpday

Output:	depends on input
Arguments:	*s* (string, strings array or string-valued series)
	format (string, optional)

This function works just like strptime, except that the return value is an "epoch day" value, for the definition of which see epochday. Since the resolution is daily, any time-of-day information in *s* is ignored.

strptime

Output:	depends on input
Arguments:	*s* (string, strings array or string-valued series)
	format (string, optional)

This function is the converse of strftime; it parses one or more date/time strings using the specified *format* and returns the number of seconds since the start of 1970 according to Coordinated Universal Time (UTC). The specific type of the return value depends on that of *s*: if *s* is a string, strings array, or string-valued series the output is, respectively, a scalar, a column vector, or a numeric series.

If *format* is omitted, it defaults to ISO 8601 "extended", `YYYY-MM-DD` (which translates to "`%Y-%m-%d`" as a strptime format).

As a special case, the first argument may be given as an 8-digit integer conforming to the ISO 8601 "basic" date format, `YYYYMMDD` (or a vector or series containing such values). In that case *format* should be omitted.

Note that the first argument to this function is taken as relative to the time-zone setting on the host computer. So for example, the call

```
strptime("13/02/2009 23:31.30", "%d/%m/%Y %H:%M.%S")
```

will produce 1234567890 on output if your system time is set to UTC but if you're in the Central European time zone (UTC+01:00) the output will be 1234564290.

The *format* options may be found by consulting the `strptime` manual page, on systems which have such pages, or via one of the many websites which present relevant information, such as http://man7.org/linux/man-pages/man3/strptime.3.html.

The example below shows how one can convert date information from one format to another.

```
scalar tm = strptime("Thursday 02/07/19", "%A %m/%d/%y")
eval strftime(tm) # default output
eval strftime(tm, "%B %d, %Y")
```

On the East Coast of the USA the result is

```
Thu 07 Feb 2019 12:00:00 AM EST
February 07, 2019
```

strsplit

Output:	string or array of strings
Arguments:	*s* (string)
	sep (string, optional)
	i (integer, optional)

In basic usage, with a single argument, returns the array of strings that results from the splitting of *s* on white space (that is on any combination of the space, tab and/or newline characters).

The optional second argument can be used to specify the separator used for splitting *s*. For example

```
string basket = "banana,apple,jackfruit,orange"
strings S = strsplit(basket, ",")
```

will split the input into an array of four strings using comma as separator.

The backslash-escape sequences "`\n`", "`\r`" and "`\t`" are taken to represent newline, carriage return and tab, respectively, in the optional *sep* argument. If you wish to include a literal backslash as a separator character you should double it, as in "`\\`". Example:

```
string s = "c:\fiddle\sticks"
strings S = strsplit(s, "\\")
```

Regardless of the separator, the members of the returned array are trimmed of any leading or trailing white space. Correspondingly, if *sep* contains non-whitespace characters then it is stripped of any leading or trailing space.

If an integer value greater than zero is given as the third argument the return value is a single string, namely the (1-based) element *i* of the array that would otherwise be produced. If *i* is less than 1 that provokes an error, but if *i* is greater than the implied number of elements an empty string is returned.

strstr

Output:	string
Arguments:	*s1* (string)
	s2 (string)
	ign_case (boolean, optional)

Searches *s1* for an occurrence of the string *s2*. If a match is found, returns a copy of the portion of *s1* that starts with *s2*, otherwise returns an empty string.

Example:

```
string s1 = "Gretl is an econometrics package"
string s2 = strstr(s1, "an")
print s2
```

If the optional argument *ign_case* is nonzero, the search is case-insensitive. For example,

```
strstr("Chicago", "c")
```

returns "cago", but

```
strstr("Chicago", "c", 1)
```

returns "Chicago".

If you just wish to find out if *s1* contains *s2* (boolean test), see instring.

strstrip

Output:	string
Argument:	*s* (string)

Returns a copy of the argument *s* from which leading and trailing white space have been removed.

Example:

```
string s1 = "    A lot of white space.  "
string s2 = strstrip(s1)
print s1 s2
```

strsub

Output:	same type as input
Arguments:	*s* (string, strings array or string-valued series)
	find (string)
	subst (string)

If *s* is a single string, returns a copy of *s* in which all occurrences of *find* are replaced by *subst*. If *s* is an array of strings or string-valued series this operation is performed on each string in the array or series. See also regsub for more complex string replacement via regular expressions.

Example:

```
string s1 = "Hello, Gretl!"
string s2 = strsub(s1, "Gretl", "Hansl")
print s2
```

strvals

Output:	array of strings
Arguments:	*y* (series)
	subsample (boolean, optional)

If the series *y* is string-valued, returns by default an array containing all its distinct values (irrespective of the current setting of the sample range), ordered by the associated numerical values starting at 1. If the dataset is currently subsampled you can give a non-zero value for the optional second argument to obtain an array holding just the string values present in the subsample.

If *y* is not string-valued an empty strings array is returned. See also stringify.

An alternative to `strvals` that may be useful in some contexts is direct assignment of a string-valued series to an array of strings: this provides not just the distinct values, but all values of the series in the current sample range.

strvsort

Output:	integer
Arguments:	*y* (series)
	S (array of strings, optional)

Carries out one or other of two sorts of rearrangment of the series *y*, which must be string-valued. The nominal return value is 0 on successful completion.

Method 1: If the second argument is not given, the effect is to sort *y* in this sense: the distinct string values are alphabetized and then the series is recoded such that 1 is assigned for the first of the ordered strings, 2 for the second, and so on. This can be useful, among other reasons, for ensuring a uniform encoding for multiple series that share the same set of string values.

Method 2: If the second argument is given, it must be an array which contains exactly the distinct string values of *y* (which can be found via strvals), but put into a preferred order. Then the effect is to recode the series such that value 1 is assigned for the first string in *S*, value 2 for the second, and so on. This can be useful for ensuring that the numeric codes "make sense" when string values can be thought of as naturally ordered.

The primary use case for these methods is the handling of string-valued series imported from third-party sources such as comma-separated files. For such data, gretl assigns numeric codes based simply on the order of occurrence of the strings across the rows of the file. So in a series with values `low`, `middle` and `high`, `high` will be assigned code 1 if it happens to occur first, rather than 3, which would clearly be more "natural". This can be fixed using Method 2. Moreover, if two or more series share the same string values, they will be encoded differently unless their distinct values happen to appear in the same order in the data file. This could be fixed by either method.

See also stringify, strvals.

substr

Output:	same type as input
Arguments:	*s* (string, strings array or string-valued series)
	start (integer)
	end (integer)

If *s* is a single string, returns the substring of *s* from the character with (1-based) index *start* to that with index *end*, inclusive, or from *start* to the end of *s* if *end* is −1. If the argument is an array of strings or string-valued series, this operation is performed on each string in the array or series.

For example, the code below

```
string s1 = "Hello, Gretl!"
string s2 = substr(s1, 8, 12)
print s2
```

gives:

```
? print s2
Gretl
```

It should be noted that in some cases you may be willing to trade clarity for conciseness, and use slicing and increment operators, as in

```
string s1 = "Hello, Gretl!"
string s2 = s1[8:12]
string s3 = s1 + 7
print s2
print s3
```

which would give you

```
? print s2
Gretl
? print s3
Gretl!
```

sum

Output:	scalar or series
Arguments:	*x* (series, matrix or list)
	partial (boolean, optional)

If *x* is a series, returns the (scalar) sum of the non-missing observations in *x*. See also sumall.

If *x* is a matrix, returns the sum of the elements of the matrix.

If *x* is a list, returns a series y such that y_t is the sum of the values of the variables in the list at observation t. By default the sum is recorded as NA if there are any missing values at t, but if you pass a non-zero value for *partial* any non-missing values will be used to form the sum.

sumall

Output:	scalar
Argument:	*x* (series)

Returns the sum of the observations of *x* over the current sample range, or NA if there are any missing values. Use sum if you want missing values to be skipped.

sumc

Output:	row vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the sums of the columns of *X*. If a non-zero value is given for the optional second argument missing values are ignored, otherwise the result is NA for any columns that contain missing values. See also meanc, sumr.

sumr

Output:	column vector
Arguments:	*X* (matrix)
	skip_na (boolean, optional)

Returns the sums of the rows of *X*. If a non-zero value is given for the optional second argument missing values are ignored, otherwise the result is NA for any rows that contain missing values. See also meanr, sumc.

svd

Output:	row vector
Arguments:	*X* (matrix)
	&U (reference to matrix, or null)
	&V (reference to matrix, or null)

Performs the singular values decomposition of the $r \times c$ matrix X:

$$X = U \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n, \end{bmatrix} V$$

where $n = \min(r, c)$. U is $r \times n$ and V is $n \times c$, with $U'U = I$ and $VV' = I$.

The singular values are returned in a row vector. The left and/or right singular vectors U and V may be obtained by supplying non-null values for arguments 2 and 3, respectively. For any matrix A, the code

```
s = svd(A, &U, &V)
B = (U .* s) * V
```

should yield B identical to A (apart from machine precision).

See also eigengen, eigensym, qrdecomp.

svm

Output:	series
Arguments:	*L* (list)
	bparms (bundle)
	bmod (reference to bundle, optional)
	bprob (reference to bundle, optional)

This function enables the training of, and prediction based on, an SVM (a Support Vector Machine), using LIBSVM as back-end. The list argument *L* should include the dependent variable followed by the independent variables and the *bparms* bundle is used to pass options to the SVM mechanism. The return value is a series holding the SVM's predictions. The two optional bundle-pointer argument can be used to retrieve additional information after training and/or prediction.

For details, please see the PDF documentation for gretl + SVM.

tan

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the tangent of *x*. See also atan, cos, sin.

tanh

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Returns the hyperbolic tangent of *x*.

$$\tanh x = \frac{e^{2x} - 1}{e^{2x} + 1}$$

See also atanh, cosh, sinh.

tdisagg

Output:	matrix
Arguments:	*Y* (series or matrix)
	X (series, list or matrix, optional)
	s (scalar)
	opts (bundle, optional)
	results (bundle, optional)

Performs temporal disaggregation (conversion to higher frequency) of the time-series data in *Y*. The argument *s* gives the expansion factor (for example, 3 for quarterly to monthly). The argument *X* may contain one or more covariates at the higher frequency to aid in the disaggregation. Several options may be passed in *opts*, and details of the disaggregation may be retrieved via *results*.

See chapter 9 of the *Gretl User's Guide* for details.

toepsolv

Output:	column vector
Arguments:	*c* (vector)
	r (vector)
	b (vector)
	&det (reference to scalar, optional)

Solves a Toeplitz system of linear equations, that is $Tx = b$ where T is a square matrix whose element $T_{i,j}$ equals c_{i-j} for $i \geq j$ and r_{j-i} for $i \leq j$. Note that the first elements of c and r must be equal, otherwise an error is returned. Upon successful completion, the function returns the vector x.

The algorithm used here takes advantage of the special structure of the matrix T, which makes it much more efficient than other unspecialized algorithms, especially for large problems. Warning: in certain cases, the function may spuriously issue a singularity error when in fact the matrix T is nonsingular; this problem, however, cannot arise when T is positive definite.

If the optional argument *det* is supplied (in pointer form), it will contain on exit the determinant of T. For example, the code:

```
A = unvech({3;2;1;3;2;3})    # Build a 3x3 Toeplitz matrix
x = ones(3,1)                # and a 3x1 vector
print A x
eval A\x                     # solution via generic inversion
eval det(A)                  # print the determinant
a = A[1,]
d = 0
eval toepsolv(a, a, x, &d)   # use the dedicated function
print d
```

produces

```
A (3 x 3)

  3   2   1
  2   3   2
  1   2   3

x (3 x 1)

  1
  1
  1

     0.25000
 -3.3307e-17
     0.25000

8
     0.25000
  2.7756e-17
     0.25000

d =  8.0000000
```

tolower

Output: same type as input

Argument: *s* (string, strings array or string-valued series)

If *s* is a single string, returns a copy of *s* in which any upper-case characters are converted to lower case. If *s* is an array of strings or string-valued series this operation is performed on each string in the array or series.

Example:

```
string s1 = "Hello, Gretl!"
string s2 = tolower(s1)
print s2
```

toupper

Output: same type as input
Argument: *s* (string, strings array or string-valued series)

If *s* is a single string, returns a copy of *s* in which any lower-case characters are converted to upper case. If *s* is an array of strings or string-valued series this operation is performed on each string in the array or series.

Examples:

```
string s1 = "Hello, Gretl!"
string s2 = toupper(s1)
print s2
```

tr

Output: scalar
Argument: *A* (square matrix)

Returns the trace of the square matrix *A*, that is, the sum of its diagonal elements. See also diag.

transp

Output: matrix
Argument: *X* (matrix)

Returns the transpose of *X*. Note: this is rarely used; in order to get the transpose of a matrix, in most cases you can just use the prime operator: `X'`.

trigamma

Output: same type as input
Argument: *x* (scalar, series or matrix)

Returns the trigamma function of x, that is $\frac{\mathrm{d}^2}{\mathrm{d}x^2} \log\Gamma(x)$.

See also lngamma, digamma.

trimr

Output: matrix
Arguments: *X* (matrix)
ttop (integer)
tbot (integer)

Returns a matrix that is a copy of *X* with *ttop* rows trimmed at the top and *tbot* rows trimmed at the bottom. The latter two arguments must be non-negative, and must sum to less than the total rows of *X*.

See also selifr.

typename

Output: string
Argument: *expr* (string)

A convenience function which combines typeof and typestr, with a little value added. Basically, the following two statements are equivalent

```
eval typestr(typeof(x))
eval typename(x)
```

except that if *expr* names an array, `typename` returns the specific type of the array, as in

```
strings S = defarray("foo", "bar", "baz")
eval typestr(typeof(S))  # gives "array"
eval typename(S)         # gives "strings"
```

typeof

Output: integer
Argument: *expr* (string)

Returns a numeric code indicating the type of *expr*, if it names a currently defined variable, specifies a sub-object such as a bundle member or array element, or is a valid expression that could stand as the right-hand side of an assignment statement. The codes are 1 for scalar, 2 for series, 3 for matrix, 4 for string, 5 for bundle, 6 for array and 7 for list. A return value of 0 indicates that *expr* names no existing object, or more generally that an assignment with *expr* on the right-hand side would fail.

A few examples follow:

```
strings S = defarray("foo", "bar")
eval typeof(S)            # gives 6 (array)
eval typeof(S[1])         # gives 4 (string)
eval typeof(S[7])         # gives 0 (out of bounds)
eval typeof(S[x])         # gives 0 (invalid index)
eval typeof(1+1)          # gives 1 (scalar)
eval typeof(sqrt("foo"))  # gives 0 (invalid)
```

The function typestr may be used to get the string corresponding to the return value from `typeof`, though if you just want the string result typename may be a more convenient alternative.

typestr

Output: string
Argument: *typecode* (integer)

Given a gretl type code (for example, obtained via typeof or inbundle), returns a string giving the name of the type. The mapping from codes to strings is: 1 = “scalar”, 2 = “series”, 3 = “matrix”, 4 = “string”, 5 = “bundle”, 6 = “array”, 7 = “list”, and 0 = “null”.

See also typename for an alternative.

uniform

Output: series
Arguments: *a* (scalar)
b (scalar)

Generates a series of uniform pseudo-random variates in the interval (*a*, *b*), or, if no arguments are supplied, in the interval (0,1). The algorithm used by default is the SIMD-oriented Fast Mersenne Twister developed by Saito and Matsumoto (2008).

See also randgen, normal, mnormal, muniform.

uniq

Output:	column vector
Argument:	*x* (series or vector)

Returns a vector containing the distinct non-missing elements of *x*, not sorted but in their order of appearance. See values for a variant that sorts the elements.

unvech

Output:	square matrix
Arguments:	*v* (vector)
	d (scalar, optional)

If the second argument is omitted, returns an $n \times n$ symmetric matrix obtained by rearranging the elements of v. The number of elements in v must be a triangular integer—i.e., a number k such that an integer n exists with the property $k = n(n+1)/2$. This is the inverse of the function vech.

If the argument d is given, the function returns an $(n+1) \times (n+1)$ matrix with the extra-diagonal entries filled with the elements of v as above. All the elements of the diagonal are set to d instead.

Example:

```
v = {1;2;3}
matrix one = unvech(v)
matrix two = unvech(v, 99)
print one two
```

returns

```
one (2 x 2)

1   2
2   3

two (3 x 3)

99     1     2
 1    99     3
 2     3    99
```

See also mshape, vech.

upper

Output:	square matrix
Argument:	*A* (square matrix)

Returns an $n \times n$ upper triangular matrix B for which $B_{ij} = A_{ij}$ if $i \leq j$ and 0 otherwise.

See also lower.

urcpval

Output:	scalar
Arguments:	*tau* (scalar)
	n (integer)
	niv (integer)
	itv (integer)

P-values for the test statistic from the Dickey–Fuller unit-root test and the Engle–Granger cointegration test, as per MacKinnon (1996).

The arguments are as follows: *tau* denotes the test statistic; *n* is the number of observations (or 0 for an asymptotic result); *niv* is the number of potentially cointegrated variables when testing for cointegration (or 1 for a univariate unit-root test); and *itv* is a code for the model specification: 1 for no constant, 2 for constant included, 3 for constant and linear trend, 4 for constant and quadratic trend.

Note that if the test regression is "augmented" with lags of the dependent variable, then you should give an *n* value of 0 to get an asymptotic result.

See also pvalue, qlrpval.

values

Output:	column vector
Argument:	*x* (series or vector)

Returns a vector containing the distinct elements of *x* sorted in ascending order, ignoring any missing values. If you wish to truncate the values to integers before applying this function, use the expression `values(int(x))`.

See also uniq, dsort, sort.

var

Output:	scalar or series
Arguments:	*x* (series or list)
	partial (boolean, optional)

If *x* is a series, returns the (scalar) sample variance, skipping any missing observations.

If *x* is a list, returns a series y such that y_t is the sample variance of the values of the variables in the list at observation t. By default the variance is recorded as NA if there are any missing values at t, but if you pass a non-zero value for *partial* any non-missing values will be used to form the statistic.

In each case the sum of squared deviations from the mean is divided by $(n - 1)$ for $n > 1$. Otherwise the variance is given as zero if $n = 1$, or as NA if $n = 0$.

See also sd.

varname

Output:	string
Argument:	*v* (integer or list)

If given an integer argument, returns the name of the variable with ID number *v*, or generates an error if there is no such variable.

If given a list argument, returns a string containing the names of the variables in the list, separated by commas. If the supplied list is empty, so is the returned string. To get an array of strings as

return value, use varnames instead.

Example:

```
open broiler.gdt
string s = varname(7)
print s
```

varnames

Output: array of strings
Argument: *L* (list)

Returns an array of strings containing the names of the variables in the list *L*. If the supplied list is empty, so is the returned array.

Example:

```
open keane.gdt
list L = year wage status
strings S = varnames(L)
eval S[1]
eval S[2]
eval S[3]
```

varnum

Output: integer
Argument: *varname* (string)

Returns the ID number of the variable called *varname*, or NA is there is no such variable.

varsimul

Output: matrix
Arguments: *A* (matrix)
U (matrix)
y0 (matrix)

Simulates a p-order n-variable VAR, that is $y_t = \sum_{i=1}^{p} A_i y_{t-i} + u_t$. The coefficient matrix A is composed by stacking the A_i matrices horizontally; it is $n \times np$, with one row per equation. This corresponds to the first n rows of the matrix `$compan` provided by the `var` and `vecm` commands.

The u_t vectors are contained (as rows) in U ($T \times n$). Initial values are in $y0$ ($p \times n$).

If the VAR contains deterministic terms and/or exogenous regressors, these can be handled by folding them into the U matrix: each row of U then becomes $u_t = B'x_t + e_t$.

The output matrix has $T + p$ rows and n columns; it holds the initial p values of the endogenous variables plus T simulated values.

See also $compan, var, vecm.

vec

Output: column vector
Argument: *X* (matrix)

Stacks the columns of X as a column vector. See also mshape, unvech, vech.

vech

Output:	column vector
Arguments:	*A* (square matrix)
	omit-diag (boolean, optional)

This function rearranges the the elements of *A* on and above the diagonal into a column vector, unless the *omit-diag* is given a non-zero value, in which case only the entries above the diagonal are considered.

Typically, this function is used on symmetric matrices, in which case it can be undone by the function unvech. If the input matrix is not symmetric and it's the lower triangle that contains the "right" values, `vech(A')` will give the desired answer (its elements may have to be re-ordered, however). See also vec.

vma

Output:	matrix
Arguments:	*A* (matrix)
	K (matrix, optional)
	horizon (integer, optional)

This function yields the VMA representation for a VAR system. If $y_t = \sum_{i=1}^{p} A_i y_{t-i} + u_t$, where u_t are the one-step-ahead prediction errors, the corresponding VMA representation is $y_t = C_0 e_t + C_1 e_{t-1} + \ldots$. The relationship between the forecast errors u_t and the structural shocks e_t is given by $u_t = K e_t$. (Note that $C_0 = K$.)

The coefficient matrix A is composed by stacking the A_i matrices horizontally; it is $n \times np$, with one row per equation. This corresponds to the first n rows of the matrix `$compan` provided by gretl's `var` and `vecm` commands. The K matrix is optional, and defaults to the identity matrix if omitted.

The returned matrix will have *horizon* rows and n^2 columns: its i-th row contains C_{i-1} in vectorized form. The *horizon* value defaults to 24 if omitted.

See also irf.

weekday

Output:	same type as input
Arguments:	*year* (scalar or series)
	month (scalar or series)
	day (scalar or series)

Returns the day of the week (from Sunday = 0 to Saturday = 6) for the date(s) specified by the three arguments, or `NA` if the date is invalid. Note that all three arguments must be of the same type, either scalars (integers) or series.

An alternative call is also supported: if a single argument is given, it is taken to be a date (or series of dates) in ISO 8601 "basic" numeric format, `YYYYMMDD`. So the following two calls produce the same result, namely 2 (Tuesday).

```
eval weekday(1990, 5, 1)
eval weekday(19900501)
```

A common alternative numbering for days of the week runs from Monday = 1 to Sunday = 7. If you have a series named `wd` obtained via `weekday` and you want to convert to the alternative you can do

```
altwd = wd == 0 ? 7 : wd
```

Note that if you simply add 1 to wd you get a numbering that's valid but non-standard, namely Sunday = 1 to Saturday = 7.

wmean

Output:	series
Arguments:	*Y* (list)
	W (list)
	partial (boolean, optional)

Returns a series y such that y_t is the weighted mean of the values of the variables in list Y at observation t, the respective weights given by the values of the variables in list W at t. The weights can therefore be time-varying. The lists Y and W must be of the same length and the weights must be non-negative.

By default the result is NA if any values are missing at observation t, but if you pass a non-zero value for *partial* any non-missing values will be used.

See also wsd, wvar.

wsd

Output:	series
Arguments:	*Y* (list)
	W (list)
	partial (boolean, optional)

Returns a series y such that y_t is the weighted sample standard deviation of the values of the variables in list Y at observation t, the respective weights given by the values of the variables in list W at t. The weights can therefore be time-varying. The lists Y and W must be of the same length and the weights must be non-negative.

By default the result is NA if any values are missing at observation t, but if you pass a non-zero value for *partial* any non-missing values will be used.

See also wmean, wvar.

wvar

Output:	series
Arguments:	*X* (list)
	W (list)
	partial (boolean, optional)

Returns a series y such that y_t is the weighted sample variance of the values of the variables in list X at observation t, the respective weights given by the values of the variables in list W at t. The weights can therefore be time-varying. The lists Y and W must be of the same length and the weights must be non-negative.

By default the result is NA if any values are missing at observation t, but if you pass a non-zero value for *partial* any non-missing values will be used.

The weighted sample variance is computed as

$$s_w^2 = \frac{n'}{n'-1} \frac{\sum_{i=1}^{n} w_i (x_i - \bar{x}_w)^2}{\sum_{i=1}^{n} w_i}$$

where n' is the number of non-zero weights and $\bar{x}_w$ is the weighted mean.

See also wmean, wsd.

xmlget

Output:	string
Arguments:	*buf* (string)
	path (string or array of strings)
	&matches (reference to scalar, optional)

The argument *buf* should be an XML buffer, as may be retrieved from a suitable website via the curl function (or read from file via readfile), and the *path* argument should be either a single XPath specification or an array of such.

This function returns a string representing the data found in the XML buffer at the specified path. If multiple nodes match the path expression the items of data are printed one per line in the returned string. If an array of paths is given as the second argument the returned string takes the form of a comma-separated buffer, with column i holding the matches from path i. In this case if a string obtained from the XML buffer contains any spaces or commas it is wrapped in double quotes.

By default an error is flagged if *path* is not matched in the XML buffer, but this behavior is modified if you pass the third, optional argument: in that case the argument retrieves a count of the matches and an empty string is returned if there are none. Example call:

```
ngot = 0
ret = xmlget(xbuf, "//some/thing", &ngot)
```

However, an error is still flagged in case of a malformed query.

A good introduction to XPath usage and syntax can be found at https://www.w3schools.com/xml/xml_xpath.asp. The back-end for `xmlget` is provided by the xpath module of libxml2, which supports XPath 1.0 but not XPath 2.0.

See also jsonget, readfile.

zeromiss

Output:	same type as input
Argument:	*x* (scalar, series or matrix)

Converts zeros to NAs. If *x* is a series or matrix, the conversion is done element by element. See also missing, misszero, ok.

zeros

Output:	matrix
Arguments:	*r* (integer)
	c (integer, optional)

Outputs a zero matrix with r rows and c columns. If omitted, the number of columns defaults to 1 (column vector). See also ones, seq.

Chapter 3

Operators

3.1 Precedence

Table 3.1 lists the operators available in gretl in order of decreasing precedence. That is, the operators on the first row have the highest precedence, those on the second row have the second highest, and so on, while operators on any given row have equal precedence. Where successive operators have the same precedence the order of evaluation is in general left to right. The exceptions are exponentiation and matrix transpose-multiply. The expression `a^b^c` is equivalent to `a^(b^c)`, not `(a^b)^c`, and similarly `A'B'C'` is equivalent to `A'(B'(C'))`.

Table 3.1: Operator precedence

()	[]	.	{}	
!	++	--	^	'
*	/	%	\	**
+	-	~	|	
>	<	>=	<=	..
==	!=			
&&				
||				
?:				

In addition to the basic forms shown in the Table, several operators also have a "dot form" (as in "`.+`" which is read as "dot plus"). These are element-wise versions of the basic operators, for use with matrices exclusively; they have the same precedence as their basic counterparts. The available dot operators are as follows.

`.^` `.*` `./` `.+` `.-` `.>` `.<` `.>=` `.<=` `.=`

Each basic operator is shown once again in the following list along with a brief account of its meaning. Apart from the first three sets of grouping symbols, all operators are binary except where noted.

- `()` Function call
- `[]` Subscripting
- `.` Bundle membership (see below)
- `{}` Matrix definition
- `!` Unary logical NOT
- `++` Increment (unary)
- `--` Decrement (unary)
- `^` Exponentiation

'	Matrix transpose (unary) or transpose-multiply (binary)
*	Multiplication
/	Division, matrix "right division"
%	Modulus
\	Matrix "left division"
**	Kronecker product
+	Addition
-	Subtraction
~	Matrix horizontal concatenation
|	Matrix vertical concatenation
>	Boolean greater than
<	Boolean less than
>=	Greater than or equal
<=	Less than or equal
..	Range from–to (in constructing lists)
==	Boolean equality test
!=	Boolean inequality test
&&	Logical AND
||	Logical OR
?:	Conditional expression

The interpretation of "`.`" as the bundle membership operator is confined to the case where it is immediately preceded by the identifier for a bundle, and immediately followed by a valid identifier (key).

Details on the use of the matrix-related operators (including the dot operators) can be found in the chapter on matrices in the *Gretl User's Guide*.

3.2 Assignment

The operators mentioned above are all intended for use on the right-hand side of an expression which assigns a value to a variable (or which just computes and displays a value—see the `eval` command). In addition we have the assignment operator itself, "=". In effect this has the lowest precedence of all: the entire right-hand side is evaluated before assignment takes place.

Besides plain "=" several "inflected" versions of assignment are available. These may be used only when the left-hand side variable is already defined. The inflected assignment yields a value that is a function of the prior value on the left and the computed value on the right. Such operators are formed by prepending a regular operator symbol to the equals sign. For example,

```
y += x
```

The new value assigned to `y` by the statement above is the prior value of `y` plus `x`. The other available inflected operators, which work in an exactly analogous fashion, are as follows.

```
-=   *=   /=   %=   ^=   ~=   |=
```

In addition, a special form of inflected assignment is provided for matrices. Say matrix `M` is 2×2. If you execute `M = 5` this has the effect of replacing `M` with a 1×1 matrix with single element 5. But if you do `M .= 5` this assigns the value 5 to all elements of `M` without changing its dimensions.

3.3 Increment and decrement

The unary operators ++ and -- follow their operand,[1] which must be a variable of scalar type. Their simplest use is in stand-alone expressions, such as

```
j++  # shorthand for j = j + 1
k--  # shorthand for k = k - 1
```

However, they can also be embedded in more complex expressions, in which case they first yield the original value of the variable in question, then have the side-effect of incrementing or decrementing the variable's value. For example:

```
scalar i = 3
k = i++
matrix M = zeros(10, 1)
M[i++] = 1
```

After the second line, `k` has the value 3 and `i` has value 4. The last line assigns the value 1 to element 4 of matrix `M` and sets `i` = 5.

Warning: as in the C programming language, the unary increment or decrement operator should be not be applied to a variable in conjunction with regular reference to the same variable in a single statement. This is because the order of evaluation is not guaranteed, giving rise to ambiguity. Consider the following:

```
M[i++] = i # don't do this!
```

This is supposed to assign the value of `i` to `M[i]`, but is it the original or the incremented value? This is not actually defined.

[1] The C programming language also supports prefix versions of ++ and --, which increment or decrement their operand before yielding its value. Only the postfix form is supported by gretl.

Chapter 4

Comments in scripts

When a script does anything non-obvious, it's a good idea to add comments explaining what's going on. This is particularly useful if you plan to share the script with others, but it's also useful as a reminder to yourself — when you revisit a script some months later and wonder what it was supposed to be doing.

The comment mechanism can also be helpful when you're developing a script. There may come a point where you want to execute a script, but bypass execution of some portion of it. Obviously you could delete the portion you wish to bypass, but rather than lose that section you can "comment it out" so that it is ignored by gretl.

Two sorts of comments are supported by gretl. The simpler one is this:

- If a hash mark, #, is encountered in a gretl script, everything from that point to the end of the current line is treated as a comment, and ignored.

If you wish to "comment out" several lines using this mode, you'll have to place a hash mark at the start of each line.

The second sort of comment is patterned after the C programming language:

- If the sequence /* is encountered in a script, all the following input is treated as a comment until the sequence */ is found.

Comments of this sort can extend over several lines. Using this mode it is easy to add lengthy explanatory text, or to get gretl to ignore substantial blocks of commands. As in C, comments of this type cannot be nested.

How do these two comment modes interact? You can think of gretl as starting at the top of a script and trying to decide at each point whether it should or should not be in "ignore mode". In doing so it follows these rules:

- If we're not in ignore mode, then # puts us into ignore mode till the end of the current line.
- If we're not in ignore mode, then /* puts us into ignore mode until */ is found.

This means that each sort of comment can be masked by the other.

- If /* follows # on a given line which does not already start in ignore mode, then there's nothing special about /*, it's just part of a #-style comment.
- If # occurs when we're already in ignore mode, it is just part of a comment.

A few examples follow.

```
/* multi-line comment
   # hello
   # hello */
```

In the above example the hash marks are not special; in particular the hash mark on the third line does not prevent the multi-line comment from terminating at */.

```
# single-line comment /* hello
```

Assuming we were not in ignore mode before the line shown above, it is just a single-line comment: the /* is masked, and does not open a multi-line comment.

You can append a comment to a command:

```
ols 1 0 2 3 # estimate the baseline model
```

Example of “commenting out”:

```
/*
# let's skip this for now
ols 1 0 2 3 4
omit 3 4
*/
```

Chapter 5

Options, arguments and path-searching

5.1 Invoking gretl

`gretl` (under MS Windows, `gretl.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 any of several formats (see the *Gretl User's Guide*); the program will try to detect the format of the file and treat it appropriately. See also Section 5.4 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 5.4 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.

`gretl --debug` (or `gretl -g`)

— (MS Windows only) Open a console window to display any messages sent to the "standard output" or "standard error" streams. Such messages are not usually visible on Windows; this may be useful for trouble-shooting.

5.2 Preferences dialog

Various things in gretl are configurable under the "Tools, Preferences" menu. Separate menu items are devoted to the choice of the monospaced font to be used in gretl screen output, and, on some platforms, the font used for menus and other messages. The other options are organized under five tabs, as follows.

[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`

General: Here you can configure the base directory for gretl's shared files. In addition there are several check boxes. 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. Checking "Allow shell commands" makes it possible to invoke shell commands in scripts and in the gretl console (this facility is disabled by default for security reasons).

Programs tab: You can specify the names or paths to various third-party programs that may called by gretl under certain conditions.

Editor tab: Set preferences pertaining to the gretl script editor.

Network tab: Set the server on which to look for gretl databases, and also whether or not you use an HTTP proxy.

HCCME tab: Set preferences regarding robust covariance matrix estimation. See the *Gretl User's Guide* for details.

MPI tab: This is shown only if gretl is built with support for MPI (Message Passing Interface).

Settings chosen via the Preferences dialog are stored from one gretl session to the next.

5.3 Invoking 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 5.4 for path-searching behavior.

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

— Prints a brief summary of usage.

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

— Prints version identification for the program.

`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 5.4 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 5.4 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.

5.4 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 5.1 for the default values; note that PERSONAL is a placeholder that is expanded by Windows in a user- and language-specific way, typically involving "My Documents" on English-language systems).
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 5.1 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 5.1: Default path settings

	Linux	*MS Windows*
User directory	`$HOME/gretl`	`PERSONAL\gretl`
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 "Tools, 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_LOCAL_MACHINE (the main gretl directory and the paths to various auxiliary programs) and HKEY_CURRENT_USER (all other configurable variables).

Chapter 6

Reserved Words

Reserved words, which cannot be used as the names of variables, fall into the following categories:

- Names of constants and data types, plus a few specials: `const`, `NA`, `null`, `empty`, `obs`, `scalar`, `series`, `matrix`, `string`, `list`, `bundle`, `array`, `void`, `for`, `continue`, `next`, `to`.
- Names of gretl commands (see section 1.2).

User-defined functions cannot have names which collide with built-in functions, the names of which are shown in Table 6.1.

Table 6.1: Function names

BFGScmax	BFGScmin	BFGSmax	BFGSmin	GSSmax	GSSmin	I	Im
Lsolve	NMmax	NMmin	NRmax	NRmin	Re	abs	access
acos	acosh	aggregate	argname	array	asin	asinh	asort
assert	atan	atan2	atanh	atof	bcheck	bessel	bin2dec
bincoeff	binperms	bkfilt	bkw	bootci	bootpval	boxcox	bread
brename	bwfilt	bwrite	carg	cdemean	cdf	cdiv	cdummify
ceil	cholesky	chowlin	cmod	cmult	cnameget	cnameset	cnorm
cnumber	cols	commute	complex	conj	contains	conv2d	corr
corresp	corrgm	cos	cosh	cov	cquad	critical	cswitch
ctrans	cum	curl	dayspan	dec2bin	defarray	defbundle	deflist
deseas	det	diag	diagcat	diff	digamma	distance	dnorm
dropcoll	dsort	dummify	easterday	ecdf	eigen	eigengen	eigensym
eigsolve	epochday	errmsg	errorif	exists	exp	fcstats	fdjac
feval	fevalb	fevd	fft	ffti	filter	firstobs	fixname
flatten	floor	fracdiff	fraclag	fzero	gammafun	genseries	geoplot
getenv	getinfo	getkeys	getline	ghk	gini	ginv	grab
halton	hdprod	hfdiff	hflags	hfldiff	hflist	hpfilt	hyp2f1
imaxc	imaxr	imhof	iminc	iminr	inbundle	infnorm	inlist
instring	instrings	int	interpol	inv	invcdf	invmills	invpd
irf	irr	iscomplex	isconst	isdiscrete	isdummy	isnan	isoconv
isocountry	isodate	isoweek	iwishart	jsonget	jsongetb	juldate	kdensity
kdsmooth	kfilter	kmeier	kpsscrit	ksetup	ksimdata	ksimul	ksmooth
kurtosis	lags	lastobs	ldet	ldiff	lincomb	linearize	ljungbox
lngamma	loess	log	log10	log2	logistic	lower	lpsolve
lrcovar	lrvar	mat2list	max	maxc	maxr	mcorr	mcov
mcovg	mean	meanc	meanr	median	mexp	mgradient	midasmult
min	minc	minr	missing	misszero	mlag	mlincomb	mlog
mnormal	mols	monthlen	movavg	mpiallred	mpibarrier	mpibcast	mpirecv
mpireduce	mpiscatter	mpisend	mpols	mrandgen	mread	mreverse	mrls
mshape	msortby	msplitby	muniform	mweights	mwrite	mxtab	naalen
nadarwat	nelem	ngetenv	nlines	nobs	normal	normtest	npcorr
npv	nullspace	numhess	obslabel	obsnum	ok	onenorm	ones
orthdev	pdf	pergm	pexpand	pmax	pmean	pmin	pnobs
polroots	polyfit	princomp	printf	prodc	prodr	psd	psdroot
pshrink	psum	pvalue	pxnobs	pxsum	qform	qlrpval	qnorm
qrdecomp	quadtable	quantile	randgen	randgen1	randint	randperm	randstr
rank	ranking	rcond	readfile	regsub	remove	replace	resample
rgbmix	rnameget	rnameset	round	rows	schur	sd	sdc
sdiff	seasonals	selifc	selifr	seq	setnote	sgn	simann
sin	sinh	skewness	sleep	smplspan	sort	sortby	sphericorr
sprintf	sqrt	square	sscanf	sst	stack	stdize	strfday
strftime	stringify	strlen	strncmp	strpday	strptime	strsplit	strstr
strstrip	strsub	strvals	strvsort	substr	sum	sumall	sumc
sumr	svd	svm	tan	tanh	tdisagg	toepsolv	tolower
toupper	tr	transp	trigamma	trimr	typename	typeof	typestr
uniform	uniq	unvech	upper	urcpval	values	var	varname
varnames	varnum	varsimul	vec	vech	vma	weekday	wmean
wsd	wvar	xmin	xmlget	zeromiss	zeros		

Bibliography

Aalen, O. (1978) 'Nonparametric inference for a family of counting processes', *Annals of Statistics* 6(4): 701–726.

Adkins, L. C., M. S. Waters and R. C. Hill (2015) 'Collinearity diagnostics in gretl'. Presented at fourth gretl conference, Berlin. URL https://learneconometrics.com/pdf/Collin/collin_gretl.pdf.

Agresti, A. (1992) 'A survey of exact inference for contingency tables', *Statistical Science* 7: 131–153.

Akaike, H. (1974) 'A new look at the statistical model identification', *IEEE Transactions on Automatic Control* AC-19: 716–723.

Arellano, M. (2003) *Panel Data Econometrics*, Oxford: Oxford University Press.

Armesto, M. T., K. Engemann and M. Owyang (2010) 'Forecasting with mixed frequencies', *Federal Reserve Bank of St. Louis Review* 92(6): 521–536. URL http://research.stlouisfed.org/publications/review/10/11/Armesto.pdf.

Baltagi, B. H. and Y.-J. Chang (1994) 'Incomplete panels: A comparative study of alternative estimators for the unbalanced one-way error component regression model', *Journal of Econometrics* 62: 67–89.

Beck, N. and J. N. Katz (1995) 'What to do (and not to do) with time-series cross-section data', *The American Political Science Review* 89: 634–647.

Belsley, D., E. Kuh and R. Welsch (1980) *Regression Diagnostics*, New York: Wiley.

Breusch, T. S. and A. R. Pagan (1979) 'A simple test for heteroscedasticity and random coefficient variation', *Econometrica* 47: 1287–1294.

Brock, W. A., W. D. Dechert, J. A. Scheinkman and B. LeBaron (1996) 'A test for independence based on the correlation dimension', *Econometric Reviews* 15(3): 1287–1294.

Brown, M. B. and A. B. Forsythe (1974) 'Robust tests for the equality of variances', *Journal of the American Statistical Association* 69(346): 364–367.

Byrd, R. H., P. Lu, J. Nocedal and C. Zhu (1995) 'A limited memory algorithm for bound constrained optimization', *SIAM Journal on Scientific Computing* 16(5): 1190–1208.

Chatterjee, S. and A. S. Hadi (1986) 'Influential observations, high leverage points, and outliers in linear regression', *Statistical Science* 1(3): 379–416.

Choi, I. (2001) 'Unit root tests for panel data', *Journal of International Money and Finance* 20(2): 249–272.

Chow, G. C. and A.-l. Lin (1971) 'Best linear unbiased interpolation, distribution, and extrapolation of time series by related series', *The Review of Economics and Statistics* 53(4): 372–375. URL https://www.jstor.org/stable/1928739.

Cleveland, W. S. (1979) 'Robust locally weighted regression and smoothing scatterplots', *Journal of the American Statistical Association* 74(368): 829–836.

Cottrell, A. (2015) 'Response surfaces for DF-GLS p-values'. Working paper. URL http://gretl.sourceforge.net/papers/df-gls-pvals.pdf.

Datta, D. D. and W. Du (2012) 'Nonparametric HAC estimation for time series data with missing observations'. Board of Governors of the Federal Reserve System, International Finance Discussion Papers, Number 1060. URL https://www.federalreserve.gov/pubs/ifdp/2012/1060/ifdp1060.pdf.

Davidson, R. and J. G. MacKinnon (1993) *Estimation and Inference in Econometrics*, New York: Oxford University Press.

_____ (2004) *Econometric Theory and Methods*, New York: Oxford University Press.

Doornik, J. A. (1998) 'Approximations to the asymptotic distribution of cointegration tests', *Journal of Economic Surveys* 12: 573–593. Reprinted with corrections in McAleer and Oxley (1999).

Driscoll, J. C. and A. C. Kraay (1998) 'Consistent covariance matrix estimation with spatially dependent panel data', *Review of Economics and Statistics* 80(4): 549–560. URL https://www.jstor.org/stable/2646837.

Edgerton, D. and C. Wells (1994) 'Critical values for the cusumsq statistic in medium and large sized samples', *Oxford Bulletin of Economics and Statistics* 56: 355–365.

Elliott, G., T. J. Rothenberg and J. H. Stock (1996) 'Efficient tests for an autoregressive unit root', *Econometrica* 64: 813–836.

Engle, R. F. and C. W. J. Granger (1987) 'Co-integration and error correction: Representation, estimation, and testing', *Econometrica* 55: 251–276.

Estrella, A. (1998) 'A new measure of fit for equations with dichotomous dependent variables', *Journal of Business & Economic Statistics* 16(2): 198–205. URL https://doi.org/10.1080/07350015.1998.10524753.

Fiorentini, G., G. Calzolari and L. Panattoni (1996) 'Analytic derivatives and the computation of GARCH estimates', *Journal of Applied Econometrics* 11: 399–417.

Geweke, J. (1991) 'Efficient simulation from the multivariate normal and student-t distributions subject to linear constraints'. In *Computer Science and Statistics: Proceedings of the Twenty-third symposium on the Interface*, pp. 571–578. Alexandria, VA: American Statistical Association.

Geweke, J. and S. Porter-Hudak (1983) 'The estimation and application of long memory time series models', *Journal of Time Series Analysis* 4: 221–238.

Godfrey, L. G. (1994) 'Testing for serial correlation by variable addition in dynamic models estimated by instrumental variables', *The Review of Economics and Statistics* 76(3): 550–559.

Golub, G. H. and C. F. Van Loan (1996) *Matrix Computations*, Baltimore and London: The John Hopkins University Press, third edn.

Golub, G. H. and J. H. Welsch (1969) 'Calculation of Gauss quadrature rules', *Mathematics of Computation* 23: 221–230.

Greene, W. H. (2000) *Econometric Analysis*, Upper Saddle River, NJ: Prentice-Hall, fourth edn.

Greenwood, M. (1926) 'The natural duration of cancer', *Ministry of Health Reports on Public Health and Medical Subjects* 33: 1–26.

Halton, J. H. and G. B. Smith (1964) 'Algorithm 247: Radical-inverse quasi-random point sequence', *Communications of the ACM* 7: 701–702.

Hamilton, J. D. (1994) *Time Series Analysis*, Princeton, NJ: Princeton University Press.

Hansen, B. E. (1997) 'Approximate asymptotic p values for structural-change tests', *Journal of Business & Economic Statistics* 15: 60–67.

Heckman, J. (1979) 'Sample selection bias as a specification error', *Econometrica* 47: 153–161.

Hyndman, R. and Y. Fan (1996) 'Sample quantiles in statistical packages', *The American Statistician* 50(4): 361–365.

Im, K. S., M. H. Pesaran and Y. Shin (2003) 'Testing for unit roots in heterogeneous panels', *Journal of Econometrics* 115: 53–74.

Imhof, J. P. (1961) 'Computing the distribution of quadratic forms in normal variables', *Biometrika* 48: 419–426.

Johansen, S. (1995) *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models*, Oxford: Oxford University Press.

Kanzler, L. (1999) 'Very fast and correctly sized estimation of the BDS statistic'. Working paper. URL http://dx.doi.org/10.2139/ssrn.151669.

Kaplan, E. L. and P. Meier (1958) 'Nonparametric estimation from incomplete observations', *Journal of the American Statistical Association* 53(282): 457–481.

Kiviet, J. F. (1986) 'On the rigour of some misspecification tests for modelling dynamic relationships', *Review of Economic Studies* 53: 241–261.

Koenker, R. (1981) 'A note on studentizing a test for heteroscedasticity', *Journal of Econometrics* 17: 107–112.

_____ (1994) 'Confidence intervals for regression quantiles'. In P. Mandl and M. Huskova (eds.), *Asymptotic Statistics*, pp. 349–359. New York: Springer-Verlag.

Koenker, R. and G. Bassett (1978) 'Regression quantiles', *Econometrica* 46: 33–50.

Koenker, R. and J. Machado (1999) 'Goodness of fit and related inference processes for quantile regression', *Journal of the American Statistical Association* 94: 1296–1310.

Koenker, R. and Q. Zhao (1994) 'L-estimation for linear heteroscedastic models', *Journal of Nonparametric Statistics* 3: 223–235.

Kwiatkowski, D., P. C. B. Phillips, P. Schmidt and Y. Shin (1992) 'Testing the null of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?', *Journal of Econometrics* 54: 159–178.

Levene, H. (1960) 'Robust tests for equality of variances'. In I. Olkin (ed.), *Contributions to Probability and Statistics: Essays in honor of Harold Hotelling*, pp. 278–292. Stanford University Press.

Levin, A., C.-F. Lin and J. Chu (2002) 'Unit root tests in panel data: asymptotic and finite-sample properties', *Journal of Econometrics* 108: 1–24.

Locke, C. (1976) 'A test for the composite hypothesis that a population has a gamma distribution', *Communications in Statistics — Theory and Methods* A5: 351–364.

MacKinnon, J. G. (1996) 'Numerical distribution functions for unit root and cointegration tests', *Journal of Applied Econometrics* 11: 601–618.

Maddala, G. S. (1992) *Introduction to Econometrics*, Englewood Cliffs, NJ: Prentice-Hall.

Mandelbrot, B. B. (1983) *The Fractal Geometry of Nature*, New York: W. H. Freeman.

Marsaglia, G. and W. W. Tsang (2000) 'The ziggurat method for generating random variables', *Journal of Statistical Software* 5: 3–30.

McAleer, M. and L. Oxley (1999) *Practical Issues in Cointegration Analysis*, Oxford: Blackwell.

McFadden, D. (1974) 'Conditional logit analysis of qualitative choice behavior'. In P. Zarembka (ed.), *Frontiers in econometrics*, pp. 105–142. New York: Academic Press.

Nelson, W. (1972) 'Theory and applications of hazard plotting for censored failure data', *Technometrics* 14(4): 945–966.

Nerlove, M. (1971) 'Further evidence on the estimation of dynamic economic relations from a time series of cross sections', *Econometrica* 39: 359–382.

Neter, J., W. Wasserman and M. H. Kutner (1990) *Applied Linear Statistical Models*, Boston: Irwin, third edn.

Newey, W. K. and K. D. West (1987) 'A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix', *Econometrica* 55: 703–708.

Ng, S. and P. Perron (2001) 'Lag length selection and the construction of unit root tests with good size and power', *Econometrica* 69(6): 1519–1554.

Odell, P. L. and A. H. Feiveson (1966) 'A numerical procedure to generate a sample covariance matrix', *Journal of the American Statistical Association* 61: 199–203.

Papadopoulos, A. (2023) 'A new matrix statistic for the Hausman endogeneity test under heteroskedasticity', *Econometrics* 11(4): 1–11. URL https://doi.org/10.3390/econometrics11040023.

Parzen, E. (1963) 'On spectral analysis with missing observations and amplitude modulation', *Sankhyā: The Indian Journal of Statistics, Series A* 25(4): 383–392.

Perron, P. and Z. Qu (2007) 'A simple modification to improve the finite sample properties of Ng and Perron's unit root tests', *Economics Letters* 94(1): 12–19.

Pesaran, M. H. (2004) 'General diagnostic tests for cross section dependence in panels'. Cambridge Working Papers in Economics (CWPE 0435). URL https://www.repository.cam.ac.uk/handle/1810/446.

Pesaran, M. H. and L. W. Taylor (1999) 'Diagnostics for IV regressions', *Oxford Bulletin of Economics and Statistics* 61(2): 255–281.

Phillips, P. C. B. and K. Shimotsu (2004) 'Local Whittle estimation in nonstationary and unit root cases', *The Annals of Statistics* 32(2): 659–692. URL http://arxiv.org/pdf/math/0406462.

Ramanathan, R. (2002) *Introductory Econometrics with Applications*, Fort Worth: Harcourt, fifth edn.

Ridders, C. (1979) 'A new algorithm for computing a single root of a real continuous function', *IEEE Transactions on Circuits and Systems* 26(11): 979–980.

Roberts, S. W. (1959) 'Control chart tests based on geometric moving averages', *Technometrics* 1(3): 239–250.

Robinson, P. (1995) 'Gaussian semiparametric estimation of long range dependence', *Annals of Statistics* 22: 1630–1661.

Saito, M. and M. Matsumoto (2008) 'SIMD-oriented Fast Mersenne Twister: a 128-bit pseudorandom number generator'. In A. Keller, S. Heinrich and H. Niederreiter (eds.), *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pp. 607–622. Berlin: Springer.

Sargan, J. D. (1958) 'The estimation of economic relationships using instrumental variables', *Econometrica* 26(3): 393–415. URL https://doi.org/10.2307/1907619.

Satterthwaite, F. E. (1946) 'An approximate distribution of estimates of variance components', *Biometrics Bulletin* 2(6): 110–114.

Schwert, G. W. (1989) 'Tests for unit roots: A Monte Carlo investigation', *Journal of Business and Economic Statistics* 7(2): 5–17.

Sephton, P. S. (1995) 'Response surface estimates of the KPSS stationarity test', *Economics Letters* 47: 255–261.

_____ (2021) 'Finite sample lag adjusted critical values of the ADF-GLS test', *Computational Economics* URL https://doi.org/10.1007/s10614-020-10082-6.

Shapiro, S. and L. Chen (2001) 'Composite tests for the gamma distribution', *Journal of Quality Technology* 33: 47–59.

Silverman, B. W. (1986) *Density Estimation for Statistics and Data Analysis*, London: Chapman and Hall.

Stock, J. H. and M. W. Watson (1999) 'Forecasting inflation', *Journal of Monetary Economics* 44(2): 293–335.

Stock, J. H., J. H. Wright and M. Yogo (2002) 'A survey of weak instruments and weak identification in generalized method of moments', *Journal of Business & Economic Statistics* 20(4): 518–529.

Stock, J. H. and M. Yogo (2003) 'Testing for weak instruments in linear IV regression'. NBER Technical Working Paper 284. URL https://www.nber.org/papers/t0284.

Swamy, P. A. V. B. and S. S. Arora (1972) 'The exact finite sample properties of the estimators of coefficients in the error components regression models', *Econometrica* 40: 261–275.

Welch, B. L. (1951) 'On the comparison of several mean values: An alternative approach', *Biometrika* 38: 330–336.

Windmeijer, F. (2005) 'A finite sample correction for the variance of linear efficient two-step GMM estimators', *Journal of Econometrics* 126: 25–51.