# The extra package
# (a collection of various convenience functions for hansl programming)

The GRETL team*

August 2026

## Contents

**1 Usage** **2**

**2 Matrix-related functions** **2**
2.1 combinations . . . 2
2.2 convex_hull . . . 2
2.3 drawbootres . . . 2
2.4 drill . . . 3
2.5 duplicate . . . 3
2.6 eliminate . . . 4
2.7 mat2latex . . . 4
2.8 nearPSD . . . 5
2.9 qformvech . . . 5
2.10 zeroifclose . . . 5

**3 Other functions working without a dataset in place** **6**
3.1 get_settings . . . 6
3.2 multi_instrings . . . 6
3.3 onemode . . . 6
3.4 powerset . . . 6
3.5 scores2x2 . . . 6
3.6 splitfname . . . 7
3.7 truncnorm . . . 8
3.8 WSRcritical . . . 8
3.9 WSRpvalue . . . 8

**4 Functions requiring a dataset** **8**
4.1 combine . . . 8
4.2 correspondence . . . 9
4.3 fracorder . . . 9
4.4 gap_filler . . . 10
4.5 winsor . . . 10

*Currently coordinated by Sven Schreiber.

5 Authors 10

6 Changelog 11

# 1 Usage

This package is intended for hansl scripting, not for gretl's GUI. (But of course other contributed function packages that make use of functions in extra.gfn can provide GUI access for themselves.)

The usual one-time requirement is to do `pkg install extra.zip` to get a copy on the local system (or install it via gretl's graphical mechanism), and then in the respective hansl script have a line `include extra.gfn`.

Note that functions that are exact lookalikes of Matlab/Octave functions do not live here, but would go into the `matlab_utilities` package.

# 2 Matrix-related functions

## 2.1 combinations

*Arguments*: `matrix x`, `int h`

*Return type*: `matrix`

This function returns a matrix whose rows are all the possible subsets of $x$ containing $h$ elements; for $h > 2$, a recursive algorithm is used.

For example: `combinations({1,2,3}, 2)` returns

```
1   2
1   3
2   3
```

The argument `x` must be a (row or column) vector. The returned matrix will have $\binom{n}{k}$ rows if successful, otherwise it will be a 1x1 matrix with an NA value.

*Nota bene*: The recursive algorithm used may be a little slow if the number of elements of $x$ is large.

## 2.2 convex_hull

*Arguments*: `const matrix P`

*Return type*: `matrix`

This function finds the convex hull for a cloud of 2-dimensional points, contained in the 2-column matrix $P$, using the so-called "Jarvis march" algorithm.

## 2.3 drawbootres

*Arguments*: `const matrix U`, `bundle bparams` (optional), `int bootcode` (optional)

*Return type*: `matrix`

Construct a new draw of innovations (residuals) for bootstrapping, based on the input matrix `U`, interpreted to be T-by-K (observations in rows). The return value has the same shape. `U` can be original residuals or some pre-processed input – the pre-processing is not done here to avoid doing it repeatedly during the bootstrap.

Two ways of choosing the bootstrap type are supported: Either by adding a string element to the `bparams` bundle under the key 'btypestr', or by using the associated number code in the `bootcode` argument, where the bundle-based choice takes precedence. Both ways are optional, the general default is plain resampling. Here is a brief description of the meanings of the codes (see also the SVAR addon's documentation):

1. plain resampling – A standard draw with replacement based on libgretl's `resample` function; so if this is all that's needed, you might be better off using `resample` directly. (Allowed string code: any word starting with "re", e.g. "resampling".)

2. wild, Normal – A draw for the wild bootstrapping scheme, where each row of the input matrix is scaled randomly according to Gaussian noise. (string code "wildN" or simply "wild")

3. wild, Rademacher – Here the signs of all rows are flipped randomly. (string code "wildR")

4. wild, Mammen – Here the multiplicative factors for the rows are the ones proposed by Mammen (see the SVAR documenation; string code "wildM").

5. moving blocks – Provides a draw of moving blocks of the input, see for example Brüggemann, Jentsch & Trenkler (2016; Journal of Econometrics, vol. 191, issue 1, 69-85). By default a block length of 10% of the input length `T` is used (rounded, at least 2). The block length can be overridden by supplying the 'moveblocklen' scalar element inside the `bparams` bundle argument. (Allowed string codes: any word starting with "mov", or "MBB".)

Note that this function is included in extra's version 1.7 and made public for convenience of potential users, but it may be moved to another separate addon in the future.

### 2.4 drill

*Arguments*: `matrices` array, `matrix rowspec` (optional), `matrix colspec` (optional)
*Return type*: `matrix`

This function "drills through" a matrix array and returns a matrix; if one sees a matrix array whose elements are equally sized as a 3-way tensor, this function can be used for extracting what are known as *fibers* and/or *slices* in tensor algebra, that is matrices with subsets of the tensor entries.[1]

for example, `drill(x, 2, 3)` returns a vector with the `[2,3]` elements of all matrices in the `x` array (a "fiber"). Omitting one of rowspec, colspec or entering "0" means to select all rows or columns respectively; the matrix thus obtained is a "slice". Of course, at least one of `rowspeca` and `colspec` must be specified.

*Nota bene*: all matrices in the array must be of the same dimensions.

### 2.5 duplicate

*Arguments*: `matrix vechA`
*Return type*: `matrix`

[1] See eg Kolda, T. G., and Bader, B. W. (2009). Tensor decompositions and applications. SIAM review, 51(3), 455–500.

The input is a vector assumed to come from an operation like vech(A). Returns vec(A), which is the result of pre-multiplying vech(A) with the "duplication" matrix $D_m$. If vechA has several columns, each column is treated separately as described above (and the results stacked side-by-side).

### 2.6 eliminate

*Arguments*: `matrix vecA`

*Return type*: `matrix`

Each column of the input vecA is assumed to come from the operation vec(A) on a square matrix, thus rows(vecA) must be a square number. Returns vech(A), which is the result of pre-multiplying vec(A) with the "elimination" matrix $L_m$. If vecA has several columns, each column is treated separately as described above (and the results stacked side-by-side).

### 2.7 mat2latex

*Arguments*: `matrix X`, `bundle opts` (optional)

*Return type*: `string`

Produces a string containing the representation of matrix `X` as a LaTeX `tabular` environment. For example,

```
eval mat2latex(mshape(seq(1,6), 2, 3))
```

produces

```
\begin{tabular}{lccc}
\hline
 &          &         Col 1  &         Col 2  &         Col 3 \\ \hline
Row 1 & 1.000         & 3.000         & 5.000 \\
Row 2 & 2.000         & 4.000         & 6.000 \\
\hline
\end{tabular}
```

Note that, if a matrix possesses row or column names, they will be automatically used as labels. Some features of the results can be tweaked by setting appropriate keys in the `opts` bundle:

**format** : a string, to be used for aligning columns. The default is to have the first column left-aligned, and the subsequent ones centered, as in "`lccc`".

**decimals** The number of decimals to use (default=3).

**nacode** The string to use for missing entries (default: empty).

**rnames** A string array for row headings.

**cnames** A string array for column headings.

For example, the code

```
open credscore.gdt
xtab OwnRent Selfempl --quiet
s = mat2latex($result, _(decimals=0))
print s
```

produces

```
\begin{tabular}{lccc}
\hline
 &            0  &             1  &          TOTAL \\ \hline
   0 & 60          & 4           & 64 \\
   1 & 35          & 1           & 36 \\
TOTAL & 95          & 5           & 100 \\
\hline
\end{tabular}
```

which looks, when compiled, as

| | 0 | 1 | TOTAL |
|---|---|---|---|
| 0 | 60 | 4 | 64 |
| 1 | 35 | 1 | 36 |
| TOTAL | 95 | 5 | 100 |

### 2.8 nearPSD

*Arguments*: `matrix` pointer `*m`, `scalar epsilon` (optional)

*Return type*: `scalar`

Forces the matrix $m$ into the positive semi-definite region. Algorithm ported from "DomPazz" in Stackoverflow, apparently mimicking the nearPD() function in R. Because of re-scaling (to correlation matrix), the `epsilon` criterion value should implicitly apply to the correlation-based eigenvalues. The return value 0 or 1 indicates whether `m` was altered or not.

### 2.9 qformvech

*Arguments*: `matrix Xt`

*Return type*: `matrix`

This function relies on the relation $vech(X'AX) = P(X \otimes X)'Q\ vech(A) = G\ vech(A)$, where $P$ and $Q$ are certain interim results. It takes the matrix $X'$ and returns the matrix $G$ such that the right-hand side of the equalities becomes feasible, which should be numerically more efficient than the direct application of the left-hand side.

### 2.10 zeroifclose

*Arguments*: `matrix` pointer `*m`, `scalar thresh` (optional)

*Return type*: `scalar`

Sets elements of `m` to zero if they are really close. The return value 0 or 1 indicates whether `m` was altered or not.

The default value for the threshold has been `1e-12` since extra version 0.7. In some applications smaller (in absolute value) but mathematically truly non-zero results may occur, in which case a smaller threshold can be chosen.

# 3 Other functions working without a dataset in place

## 3.1 get_settings

*Arguments*: `string key` (optional)

*Return type*: `bundle`

The returned bundle contains information on either one or (almost) all of the gretl state variables that can be configured via the `set` command ("libset variables"). The bundle members are named for the keys of the libset variables and their values are either strings or scalars.

If no argument is given, all libset variable values are represented, with the exception of a few that take the form of matrices (`initvals`, `initcurv`), and a few others of no interest from a programming point of view (`echo`, `messages`, `verbose`). If an argument is given, it should be the key for a particular libset variable, in which case the returned bundle has a single element.

## 3.2 multi_instrings

*Arguments*: `strings lookinhere`, `strings tofind`

*Return type*: `matrix`

Returns in a column vector the positions (indices) in 'lookinhere' where any of the strings from 'tofind' occur. If there are duplicates in 'tofind' then the output may also contain duplicate indices. Use `uniq()` or `values()` afterwards if needed.

## 3.3 onemode

*Arguments*: `matrix v`

*Return type*: `matrix`

Finds the mode of the empirical distribution of the input data. If the data are multimodal, details of internal computer arithmetic can influence which of the modes is actually found. Returns a 2-element column vector with the modal value and its absolute frequency. If `v` is an empty matrix (comprises only nan values) a $1 \times 1$ matrix with nan is returned.

## 3.4 powerset

*Arguments*: `strings S`

*Return type*: `strings (array)`

Computes the powerset of the input S, i.e. all possible combinations of the string elements in S. (Including the empty set / empty string `""`.) Each combination yields one string in the output array. Being a set, the ordering is not defined and arbitrary.

## 3.5 scores2x2

*Arguments*: `matrix in, bool verbose` (optional)

*Return type*: `matrix`

Computes some standard score measures for a $2 \times 2$ contingency table of the form:

| | | Observed | |
|---|---|---|---|
| | | 1 | 0 |
| Predicted | 1 | h(its) | f(alse) |
| | 0 | m(iss) | z(eros) |

and $n = h + f + m + z$ (total observations). Returns a column vector with the elements listed in Table 1. The input is always sanitized by taking the upper 2x2 part, using absolute values, and integer-ization. Warnings are issued if `verbose` is 1.

| Number | Acronym | Description | Formula |
|---|---|---|---|
| 1 | POD | prob of detection | $\frac{h}{h+m}$ |
| 2 | POFD | prob of false detection | $\frac{f}{f+z}$ |
| 3 | HR | hit rate | $\frac{h+z}{n}$ |
| 4 | FAR | false alarm rate | $\frac{f}{h+f}$ |
| 5 | CSI | critical success index | $\frac{h}{h+f+m}$ |
| 6 | OR | odds ratio | $\frac{h \cdot z}{f \cdot m}$ |
| 7 | BIAS | bias score | $\frac{h+f}{h+m}$ |
| 8 | TSS | true skill stat ($POD - POFD$); also known as the Hanssen-Kuipers score | $\frac{h}{h+m} - \frac{f}{f+z}$. |
| 9 | HSS | Heidke skill score | $2\frac{h \cdot z - f \cdot m}{(h+m)\cdot(m+z)+(h+f)\cdot(f+z)}$ |
| 10 | ETS | equitable threat score | $\frac{h \cdot z - f \cdot m}{(f+m)\cdot n+(h \cdot z - f \cdot m)}$ |
| 11 | PRC | precision | $\frac{h}{h+f}$ |
| 12 | FSC | $F$-Score | $2\frac{PRC \cdot POD}{PRC+POD} = 2\frac{h}{1+h+m}$ |

Table 1: Elements returned by the `scores2x2` function

### 3.6 splitfname

*Arguments*: `string fn`

*Return type*: `strings (array)`

The idea is to take a file name or full path and extract three components:

1. The path prefix (may be empty; without the trailing / or \)
2. The "base" component of the file name, without the extension and without the path prefix
3. The file extension (without the dot; may be empty)

In principle this should work with both forward slashes and backslashes, and also with doubled slashes.

Example:
Input string: `"/what/on/earth/isthisfile.gdt"`
Output equivalent to:

```
defarray("/what/on/earth", "isthisfile", "gdt")
```

### 3.7 truncnorm

*Arguments*: `int n, scalar m, scalar sigma, scalar below, scalar above`

*Return type*: `matrix`

Generates *n* truncated normal random values. Specify mean `m` and standard deviation `sigma`, and the left/right truncation values `below` and `above`. (Pass NA for any one of them to skip the respective truncation.) Returns a column vector of values.

### 3.8 WSRcritical

*Arguments*: `int n, scalar prob` (optional), `bool forcenorm` (optional)

*Return type*: `matrix`

Concerns the distribution of Wilcoxon's signed rank test statistic for `n` trials (at least 4). Tries to find the critical values (low/hi) where the two-sided area to the outside is as close as possible to the given `prob` (default: 0.05). (Note that "outside" means including the critical values themselves in the exact/discrete case.) If we end up in the interior region not covered by the exact table (for `prob` far away from 0 and also from 1), we fall back to the normal approximation. The function returns a column vector `{lo; hi; epv}`, where `epv` is the actual probability mass (close to `prob` but not equal in general for small samples). `lo` and `hi` can be non-integers in the normal approximation case. The normal approximation instead of the exact table values can be enforced with the `forcenorm` argument (default: zero, do not enforce).

See also the sister function `WSRpvalue`.

### 3.9 WSRpvalue

*Arguments*: `int n, scalar W, bool forcenorm` (optional)

*Return type*: `scalar`

Concerns the distribution of Wilcoxon's signed rank test statistic for `n` trials (at least 4), returns $P(X \geq W)$. In the interior region not covered by the exact table, the true value is $\geq$ 12.5% (and $\leq$87.5%) according to the table used,[2] so typically based on such values H0 would not be rejected. We fall back to the normal approximation in this region. In the extreme outer regions not explicitly covered by the table, the deviation from 0 or 1 will be smaller than 0.5% = 0.005. We return values 0.001 or 0.999 as an approximation here. The test statistic `W` should usually be an integer, but in case of bindings it could be fractional as well; in this case we also fall back to the normal approximation.

The normal approximation instead of the exact table values can be enforced with the `forcenorm` argument (default: zero, do not enforce).

See also the sister function `WSRcritical`.

## 4 Functions requiring a dataset

### 4.1 combine

*Arguments*: `series a, series b`

*Return type*: `series`

[2] Source of the table: Wilfrid J Dixon and Frank J. Massey, Jr., Introduction to Statistical Analysis, 2nd ed. (New York: McGraw-Hill, 1957), pp. 443-444.

This function takes as arguments two discrete series and computes all the combinations of their values that occur in the selected sample of the currently open dataset. These are stored in the resulting series.

For example, suppose you have a dataset of trade flows, with two series `ic` and `ec` for the importing and exporting countries, respectively. Then `combine(ic,ec)` will generate a series in which each pair has a distinct encoding.

If the two input series are both string-valued, then the output series will also be string-valued, as long as it's possible to assign unique labels to each value.

### 4.2 correspondence

*Arguments*: `series a, series b`

*Return type*: `scalar`

This function takes two series and establishes if there's a 1-to-1 relationship between them, in which case it returns 2. If there's a 1-to-n relationship such that `a` could be interpreted as a (mathematical) function of `b`, it returns 1. If there's no relationship – for example several different values of series `a` appear together with some value of `b` – it returns 0.

One of the possible use cases is to check whether two discrete series encode the same variable. For example, the code:

```
open grunfeld.gdt
c = correspondence($unit, firm)
```

sets `c` to 2, indicating that the variable `firm` is in fact the panel cross-sectional identifier.

### 4.3 fracorder

*Arguments*: `series x, int order` (optional), `bool verbosity` (optional)

*Return type*: `matrix`

Meta function to invoke all the various ways in gretl to estimate the order of fractional integration of the input series, namely the Local Whittle estimator, the one by Geweke & Porter-Hudak (GPH), and the Hurst exponent minus 0.5. The first two are executed through gretl's command `fractint`, the latter via `hurst`.[3]

Returns a matrix with three rows corresponding to the methods above; the four columns contain (1) the point estimate, (2) its standard error, (3) the test statistic for the null hypothesis of integration order zero, (4) the associated p-value. For example, to obtain the standard error of the Local Whittle estimator one picks the 1,2-element of the output matrix. The optional 'verbosity' switch is set to 0 (OFF) by default, otherwise the standard output of the underlying commands is printed out.

The optional 'order' argument only applies to the Local Whittle and GPH estimators and overrides gretl's default lag order of $\min(T/2, T^{0.6})$.

For the Hurst method a minimum of 128 observations is required, and test results are never available. Also note that by construction this estimator can only take values between $-0.5$ and 0.5.

[3] Another estimation approach for the Hurst exponent is provided in the user-contributed function package gen_hurst.

### 4.4 gap_filler

*Arguments*: `series x, int method` (optional)

*Return type*: `series`

Simple convenience function to crudely get rid of missing values interspersed between valid observations. The function is meant to be used with time series, or panel datasets with a time dimension. An error is returned if the function is used with a cross-sectional dataset.

Apart from the first argument (series), it accepts an integer parameter as second argument, whose meaning is: 0: do nothing, leave the gaps; 1: NAs are replaced with previous observations; 2: NAs are replaced with a linear interpolation (this uses the internal function `interpol()`). Returns the filled series.

The very existence of the "0" method for interpolation may look bizarre at first sight, but it may make sense in the context of batch processing, as in the following example (hopefully, self-explanatory):

```
k = 1
loop foreach i X
   series z_$i = gap_filler($i, action[k++])
endloop
```

Note that the function only replaces NAs between valid observations; therefore, if the origin series has missing values at the beginning or the end of the sample, they will be in the returned series too.

### 4.5 winsor

*Arguments*: `series x, scalar p` (optional), `scalar phi` (optional)

*Return type*: `series`

Returns a trimmed ("winsorized") version of the series, where outliers are replaced with implicit threshold values. Truncation quantiles are determined according to relative tail frequencies `p` and `phi`. Default lower and upper frequencies are 0.05, but re-settable with `p`. Pass `phi` in addition to `p` for an asymmetric trimming, then `p` determines only the lower frequency and `phi` the upper.

## 5 Authors

- gap_filler, eliminate, duplicate, truncnorm, powerset, drill, correspondence, combinations, qformvech, mat2latex, combine: Jack Lucchetti
- nearPSD, zeroifclose, scores2x2, WSRcritical, WSRpvalue, onemode, splitfname, multi_instrings, fracorder, put_outofsmpl: Sven Schreiber
- winsor: Sven Schreiber, original code JoshuaHe
- drawbootres: Jack Lucchetti and Sven Schreiber

# 6 Changelog

- March 2024: add the get_settings function
- September 2023: add the combine function
- July 2023: adopt the drawbootres function from the SVAR addon to make it publicly available
- January 2023: introduce the put_outofsmpl function; internal refactoring of the mat2latex() function
- July 2022: introduce the mat2latex function
- June 2022: retire the commute function (now in libgretl)
- January 2022: retire the mat2list function (now in libgretl), update gap_filler
- October 2021: add qformvech
- June 2021: add combinations, and increase gretl version requirement to 2020c
- November 2020: add mat2list
- September 2020: add fracorder, remove bwritejson
- July 2020: add multi_instrings and correspondence, add deprecation warning to bwritejson, efficiency improvement for zeroifclose
- January 2020: add drill, bwritejson, onemode, splitfname; finally remove the retired sepstr2arr (use native strsplit instead); slightly revise gap_filler; rearrange the documentation a little
- October 2018: fix small commute bug; retire sepstr2arr; add powerset, eliminate, duplicate
- February 2018: allow non-integer input in WSRpvalue
- January 2018: add WSRcritical, WSRpvalue
- December 2017: add scores2x2; switch to pdf help document
- September 2017: add winsor
- July 2017: initial release