



RIEMANN DOCUMENTATION:	
TYPE	Manual
AUTHOR	Anders Källén
VERSION NO.	1.0
DATE	August 8, 2002

Command Reference to the Riemann Library

Input data

A number of input parameters to Riemann Library procedures are in common. Within each procedure description these will be described only briefly. Here we expand this common description somewhat.

- cfg** a string defining a linear model based on class variables and covariates. Using this you can define interactions in the model, without explicitly constructing those variables. Use variable names in case of data on disk, and `xj` for variable `j` in case of data in memory. **cfg** is discussed in more detail in the Manual to the Riemann Library.
- class** Defines class variables, also called factors. This can be either numeric variables, referring to column numbers in **dataset** (both for data sets on disk and data in memory), or a variable names for a data set on disk. For a data matrix in memory, character data should be inputted with a negative number: -2 means column 2, which is a character data column.
- cvars** column indices or variable names for control variables, i.e. variables that define groups. In general negative numbers should be used for character data. Used interchangeably with keys.
- dataset** This is either a string, referring to a Gauss data set on disk, or a data matrix in memory

- dep** Defines dependent variable(s). These can be either numerics, referring to column number in **dataset** (both for data sets on disk and data in memory), or variable names for a data set on disk. For a data matrix in memory, character data should sometimes be inputted with a negative number: -2 means column 2, which is a character data column. **dep** can also be a string (array), allowing for taking **ln** of one or more variables, in which case column index j should be written x_j (example: `dep = "ln[x1]"`).
- indep** Defines independent variable(s). These can be given either as numerics, referring to column number in **dataset** (both for data sets on disk and data in memory), or variable names for a data set on disk.
- keys** column indices or variable names for key variables, i.e. variables that define groups. In general negative numbers should be used for character data. Used interchangeably with **cvars**.
- q** a **STATEST** procedure has a packed vector as its output, which is input to other procs. Its content can be found using
`vlist(q);`
and an entry obtained using
`x = vread(q,entry);`

Global variables

<code>_FactorNames</code>	string array, defining names for class variables for a <code>STATEST</code> -procedure. These names will be printed in the output, and overrides the actual variable names in the case of data on disk.
<code>__output</code>	Control if prints output or not. If =0, no output is shown.
<code>_RegNames</code>	string array, defining names for the independent variables of a <code>STATEST</code> -procedure. In the linear model case, independent variables means covariates (as opposed to class variables). These names will be printed in the output, and overrides the actual variable names in the case of data on disk.
<code>_rmn_conflevel</code>	Confidence level for confidence intervals. Default is .95, meaning 95%.
<code>_rmn_dep_constraints</code>	Matrix defining either which dependent variables to be used or linear constraints on the dep. vars. If it has the same number of rows as <code>rows(dep)</code> , it is supposed to pick which variables to use, if it has the same number of columns as <code>rows(dep)</code> it is assumed to define linear combinations that are constrained. I.e. $\{0\ 0\ 0\ 1\ 0,\ 1\ 1\ 1\ 1\ 1\}$ means that variable 4 should not be included in the fit and that the sum of all variables is constant. NEEDS TO BE REVISITED!

- `_rmn_events` A 3-column matrix which can be used to rename the event variable in some models (categorical models and censoring variables). A row looks like
`{n bef aft}`
where `n` defines variable, either by column index (negative for categorical data) or (when appropriate) name. Both `bef` and `aft` are strings of the type
`"item1@item2@...@lastitem"`
The `@` is a cell delimiter – a mapping is done row-wise in this so that `item1` in `bef` is replaced by `item2` in `aft`.
- Example:** `_rmn_events = {3 "1@0@2"
"0@1@0"};`
replaces '0' with '1', '1' with '0' and '2' with '0' in column 3.
- `_rmn_GradMethod` Method used to compute numerical gradients:
1 = Forward difference
2 = Central difference
3 = Richardson extrapolation
- `_rmn_GradProc` Pointer to a procedure that computes an analytical gradient for a regression function. Should have the same input structure as the regression function it differentiates.
- `_qn_MaxIters` Maximal number of iterations an iterative method should do.
- `_qn_PrintIters` How progress is shown in an iterative estimation procedure:

- 0 = No display
- 1 = Everything written to output
- 2 = Lowest level shown in Window
- 3 = Multiple levels shown in windows

`_tab_Replace`

A 3-column string array which can be used to re-name character data within some procedures. A row looks like

```
{n bef aft}
```

where `n` defines variable, either by column index or (when appropriate) name. Both `bef` and `aft` are strings of the type

```
"item1@item2@...@lastitem"
```

The `@` is a cell delimiter – a mapping is done row-wise in this so that `item1` in `bef` is replaced by `item2` in `aft`.

Example: `let string _tab_replace =`

```
{1 "0@1@2" "None@1-24@>=25"};
```

replaces '0' with 'None', '1' with '1-24' and '2' with '>=25' in column 1.

`--weight`

Defines case weights in likelihood function. Can be a vector the length of the dataset, or a column number and variable name in the corresponding dataset, which then should contain the weights.

AddOption

Purpose: Adds an option to the global variable `__options`

Format: `AddOption(str);`

Input: `str` the string to be added to `__options` Several options can be added at once, if parted with `,`

Example: `AddOption("option1,option2,option3")`

Source: `options.src`

Ameanc

Purpose: Computes the arithmetic means of a set of variables contained in the columns of a matrix. Pairwise deletion of missings.

Format: `m = Ameanc(x);`

Input: `x` NxK matrix

Output: `m` Kx1 vector containing the arithmetic means of each column of `x`.

Source: `basstat.src`

ANOVA

Purpose: ANOVA either from dataset on disk or matrix in memory.
Model of form $X = A b$.

Format: `q = ANOVA(dataset,dep,class,indep,cfg);`

Input:

- `dataset` string, name of Gauss dataset or name of matrix in memory
- `dep` dependent variables or column indices
- `class` class variables or column indices
- `indep` independent variables or column indices
- `cfg` string, linear model definition

Output: `q` a packed vector

Globals:

- `_rmn_conflevel` confidence level
- `__output` controls output
- `__weight` case weights
- `_FactorNames` string array, names of class variables
- `_RegNames` string array, names of covariates

Source: `Ancova.src`

BinomialProportions

- Purpose:** Compares two independent binomial proportions in terms of a number of indices: difference, relative risk, odds ratio
- Format:** `stat = BinomialProportions(table);`
- Input:** `table` 2x2-table, independent rows
- Output:** `stat` table of result as matrix.
- Options:** `EXCESS` estimate and confidence intervals are calculated for excess rate also
- Remark:** Confidence limits are based on likelihood, assuming asymptotic normality of it (NOT the estimate)
- Source:** `cmh.src`

ByTransformToDisk

Purpose: To transform data within values of control variables

Format: `outds = ByTransformToDisk(dataset,ix,&fnc,outds,nms,z);`

Input:

- dataset** string, name of Gauss data set, or matrix in memory
- ix** indices or names of variables to transform within
- fnc** user-defined procedure defining a function $f(\text{data},q)$ which defines what transformations to do. if a call to `fnc` returns a scalar missing, this will not be written to `outds`
- outds** string, name of transformed data set
- nms** character vector, variable names in `outds`
- z** data passed directly to the function `fnc`

Output: `outds` string, name of transformed data set, or matrix in memory

Remark: When operating on a matrix in memory, the output is the same, and in that case neither of `outds,nms` are used. Operationally, `fnc(data,z)` is computed for each subset `data` of the data set defined by a unique combination of control variables. For data on disk, it is therefore mandatory that data is sorted on key variables, otherwise there is no guarantee that all data are passed to `fnc` in one pass!

Source: `Dstrans.src`

CDFUC

Purpose: This function computes the integral from x to infinity of the distribution for Wilk's lambda

Format: `y = cdfuc(x,p,q,dfe);`

Input: `x` matrix
`p,q,dfe` degrees of freedom

Output: `y` required probability

Remark: Method taken from SAS (version 5.04?)

Source: `lmutil.src`

CensoredRegression

Purpose: To estimate a linear model with censored data.

Format: `q = CensoredRegression(dataset,dep,class,indep,cfg,distr);`

Input:

- dataset** either string with name of dataset on disk or name of matrix already in memory
- dep** dependent variables. If dataset is string, then names of columns, if dataset is matrix, then vector with column numbers.
dep[1] = Time
dep[2] = Event (censored=0, event=1)
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep if indep=0, only a constant is used
- cfg** string giving model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column
- distr** string defining which model/distribution to use. This string should define which distribution family to use. Cox Regression can be called for, using COX...

Output: `q` packed vector output.

Globals:

- `_rmn_strata` stratification variable, used only for COXRegression
- `_rmn_events` transform event variable (dep[2]). Can be used e.g. to censor specific events in a competitive risks situation

continued on next page

Remark: At present a CONSTANT IS ALWAYS added to the model,
except for Cox Proportional Hazards model!!

Source: `survreg.src`

ClearOption

Purpose: Clears the global variable `__options`

Format: `ClearOption;`

Source: `options.src`

CompeteRiskP

Purpose: Computes the influenced transition probabilities for Competing risks model

Format: `out = CompeteRiskP(dataset,dep,keys);`

Input:

dataset data matrix

dep indices or variable names for columns giving
1 = observational start time (optional)
2(1) = observational end time
3(2) = risk variable
The censor variable should take integer values. Only one value means no censoring. More than two values means that a competing risk situation is encountered: THEN 0 MUST MEAN CENSORED in other cases the lowest value is taken to indicate censor

keys column indices or variable names giving key variables (or 0) use negative numbers for character columns

Output: `out` matrix with the following columns:
1:n gives keys, $n=\text{rows}(\text{keys})$
 $n+1$ value of risk variable (except 0)
 $n+2$ jump times
 $n+3$ the estimator $P(t)$
 $n+4$ std of estimator
 $n+5, n+6$ confidence limits for estimator

Options: `ciplain` plain scale for confidence limits

continued on next page

Globals:

<code>_rmn_conflevel</code>	confidence degree (default is 0.95)
<code>_psym</code>	output global, can be used directly to plot censored obs. in a Kaplan-Meier plot
<code>_plmaxtime</code>	key variables (including risk value if relevant) followed by the maximal time point for this group. This global is used by PlotCurves when plotting survival function plots to extend the left part of the curve.

Source: `Countpr.src`

Concatenate

Purpose: To concatenate two matrices with filling out space with missings

Format: `c = Concatenate(a,b);`

Input: Two matrices, one can be allowed to be {}

Output: a b, with missings filled out to get a matrix

Source: `Util.src`

ConfInterval

Purpose: To print confidence intervals for contrasts in ANOVAs

Format: `out = ConfInterval(q,d,lbl);`

Input:

- `q` packed vector, STATEST proc output
- `d` $R \times \text{rows}(\text{beta})$ -matrix. Each row represent a contrast of beta for which a confidence interval is to be computed
- `lbl` string matrix - names of contrasts

Output: `out` Table with statistics, excluding row names

Options:

- `P-VALUE` P-values are outputted
- `NOHEAD` there is no Table Header
- `LOG` exponentiation is done

Remark: if the beta-vector comes from an MANOVA analysis, it will have $\text{nb} > 1$ columns. This proc gives CIs for $\text{vec}(\text{beta})!$ Note that if `d` is adjusted to beta, it will be expanded - but if you want to compare elements from different columns of beta, define `d` relative $\text{vec}(\text{beta})$

Source: `confint.src`

Correlation

- Purpose:** To analyse correlation coefficients
- Format:** `out = Correlation(data);`
- Input:** `data` Nx2-matrix with columns equal to variables
- Globals:** `_rmn_conflevel` confidence degree
- Options:** `Spearman` first rank transform the variables
- Output:** `out` 1x5-matrix containing columns:
1. number of observations
 2. estimated correlation coefficient
 3. lower confidence limits
 4. upper confidence limits
 5. P-value for testing $\rho = 0$
- Source:** `basstat.src`

COXPHCeck

Purpose: To compute various residuals and alike to assess the assumptions made in fitting a Cox regression model and the influence of individuals.

Format: `q = CoxPHCheck(q,g);`

Input:

- q** packed vector output from CensoredRegression with COX-alternative or CoxRegression
- g** vector giving the desired transformation of (sorted survival time (vector case, note that it must have the same number of rows as there are non-censored data and sorted acc. to time) or 0, which uses the Kaplan-Meier estimate obtained from SURVIVALFUNCTION

Output:

- q** Updated packed vector:
 - "rwschres" - Nx(P+1) matrix with Schoenfeld residuals
 - "rwschnms" - names in 'RWSCHRES'
 - "zphtbl" - ZPH-table - the Grambsch-Therneau's test for Proportional Hazards, which also is printed if `--output > 0`
 - "gfunc" - the values of the vector g

Source: `Coxdiag.src`

COXRegression

Purpose: To estimate COX Proportional Hazards model. Time-dependent covariates and recurrent events can be analysed.

Format: `q = COXRegression(dataset,dep,class,indep,cfg,id,covtds,covtnms,options);`

Input:

- dataset** data matrix or name of dataset which can be read into memory.
- dep** 1,2 or 3-vector containing column numbers referring to data for
 - 1 - start of observational time (optional)
 - 2(1) - end of observational time
 - 3(2) - censor variable (optional, provided start time is not included)
- class** names or column numbers for class variables
- indep** names or column numbers for covariates
- cfg** model definition string
- id** column index or variable name pointing to what column of dataset contains an ID variable This is needed if you have multiple occurrences of individuals or want to use time-dependent covariates
- covtds** data matrix or name of dataset which contains the following columns
 - 1 - ID code
 - 2 - Time of observation
 - 3+ - values of covariatesIf a dataset, it must be sorted on ID code

continued on next page

covtnms column indices or variable names for the columns in `covt_ds`. Order is essential. If 0, columns will be used as given.

options string defining specific types of analysis:
WLW or MARG - Wei, Lin , Weissfeldt type of marginal analysis
COND - Conditional analysis
GAP - analysis on gap times
PWP - conditional analysis on gap times
FRAILITY - shared gamma-frailty model

Output: **q** a packed vector

Options: "BRESLOW" use Breslows method for tie handling
"EXACT" use exact method for tie handling. Uses Gauss-Laguerre quadrature - no. of nodes is controlled by the global `_intord` (default 12).

Globals: **_rmn_strata** Lx2-matrix defining stratification variable(s) First column should contain column no., second a name of the stratification variable. For 'data on disk' case only one of these is needed. If 0, no stratification.

_rmn_events a string array with 3 columns. Used to transform event variable.

_rmn_start_b start vector for b in algorithm. Default is 0, which means that start vector consists of zeros

__weight can provide a 0/1 vector by which selection of rows in dataset is done

__output control whether output is on or not

continued on next page

Source: Coxreg.src

DataStack

Purpose: To stack a sequence of datasets

Format: `DataStack(datasets,outfile);`

Input: `datasets` string array, containing names of the datasets
to stack
`outfile` string, giving name of final dataset

Remark: DataStack works sequentially: the first two are stacked so that
outfile is created with the variables in this order:

1. variables from `datasets[1]`
2. variables from `datasets[2]` that do not duplicate names in
`datasets[1]`
3. `_JOINSRC` is the last variable: value = j if record from
`datasets[j]`

This is continued recursively.

Example: `let datasets = "diary_0" "diary_1" "diary_2"
"diary_3"; datastack(datasets,"dagbok");`

Source: `dsopers.src`

DDesc

Purpose: to list variable names and indices in a Gauss dataset

Format: DDESC(fname);

Input: fname string, name of Gauss dataset

Source: Util.src

DeleteFile

Purpose: Delete files on disk

Format: DeleteFile(filename);

Input: filename stringarray of filenames to be deleted. Wildcards are allowed

Source: dllutils.src

DescStat

Purpose: Compute very basic statistics, possibly in subsets defined by control variables.

Format: `{group,n,mean,std,min,max} = DescStat(dataset,vars,cvars);`

Input:

- dataset** either string, defining dataset on disk, or data matrix
- vars** column numbers or variable names defining variables to compute occurrence of different values in. Negative numbers indicate character columns in a data matrix.
- cvars** 0 or column numbers/variable names defining control variables. Negative numbers indicate character columns in a data matrix. 0 means no subgroups.

Output:

- group** $G \times T$ -matrix defining groups defined by control variables.
 $G =$ no. of groups, $T =$ rows(cvars)
- n,mean,std,min,max** $G \times N$ -matrices where $N =$ rows(vars)

Options: `LOG` statistics of logged variables are computed

Globals: `--weight` case weights

Source: `basstat.src`

DesignMatrix

Purpose: Build design matrix for linear model with class, covariates and interactions.

Format: `a = DesignMatrix(data,class,indep,cfg);`

Input:

- `data` name of matrix in memory
- `class` class variables (factors). Given as column numbers.
- `indep` independent variables (covariates). Column numbers
- `cfg` string giving model format. Use x1 to denote column 1 etc

Output: a Design matrix.

Remark: Calls on LMModel for construction.

Source: `lmmodel.src`

Diagnostics

Purpose: Computes influential statistics

Format: `q = Diagnostics(q,dmat);`

Input: `q` packed vector, output from a STATEST procedure
`dmat` contrast matrix, defining which linear constrast of coefficients to compute. If `dmat = 0`, coefficients will be outputted as they are.
If `q` comes from ANOVA `dmat` can also be a pointer to a procedure which computes a function `f(b,cov)`.

Output: `q` updated input

Options: `P-VALUE` and `dmat` contains a number of rows, the p-value for the test corresponding to that matrix of contrasts will be computed instead. At present only implemented when the packed output `q` comes from ANOVA.

Remark: This call must be made in connection to the call to statistical estimation proc, since it uses not only the packed vector output but also the dataset `ds$resid` disk. Diagnostics adds contents to this dataset

Source: `diagnos.src`

DiskSort

Purpose: To sort data file on disk with respect to specified variables.

Format: `DiskSort(infile,outfile,keyvars);`

Input:

- `infile` string, name of input file.
- `outfile` string, name of output file, must be different.
- `keyvars` string, name of key variables. The Gauss convention with upper case for Numeric and lower case for character data MUST be followed

Globals: None

Remark: Similar to to SORTDISK, but takes more than one key

Source: `dsopers.src`

EliminationRate

Purpose: To estimate the terminal elimination rate in pharmacodynamic studies. That data will be written to a GAUSS data set on disk and returned as output.

Format: `kel=EliminationRate(data,vars,keys,dataset);`

Input:

- `data` data matrix
- `vars` 2x1 or 3x1 vector giving column nos in data for 1-Time 2-Conc 3-Operator (optional)
- `keys` vector containing column nos for key variables
- `dataset` - string, defining dataset to write on

Output: `kel` matrix containing keys + Intercept, Elimination rate, Start time, and End time for the regression. `kel` is also saved in `dataset` (if $\neq 0$).

Globals: `__interactive` if $=0$ you will not be asked for manual modification

Remark: If `dataset` is not 0, a log-file is written with this as first name and extension `.log`
Action: First all subjects are automatically analysed. The regression starts with the last three points and then works backwards until the R-squared does not increase. Then one further point is checked. If still not increasing one is done. If increasing the failure point is considered an outlier and the process starts all over again with that point eliminated. The automatic regression fits can then be manually modified.

Source: `pkprocs.src`

FiellerInterval

Purpose: To compute Fieller confidence intervals for quotients of means

Format: `out = FiellerInterval(means,cov,df,lbl);`

Input:

- `means` 2x1-matrix with means
- `cov` covariance matrix for means
- `df` degrees of freedom
- `lbl` string, name of quotient if printed output is required

Globals:

- `_rmn_conflevel` confidence level
- `--output` if 1, printed output

Output: out - 1x3-matrix with

1. quotient
2. lower limit
3. higher limit

Source: `basstat.src`

FindMinimum

Purpose: Engine proc to find a local/global minima of a function

Format: `{x,f,g,h,ret,itors} = FindMinumum(&fct,dataset,x0,q,prt);`

Input:

<code>&fct</code>	pointer to a proc having the following format: <code>{f,g,h} = fct(dataset,x,q,typ);</code> where dataset - auxillary parameter, typically data x - value to compute function in q - another auxillary parameter typ - if 0 only f is computed, else all of f,g,h f - value of function g - gradient h - (approximation to) the hessian Note that fct should be written so that typ has effect
<code>dataset</code>	auxillary parameter, typically data, used only by fct
<code>x</code>	value to compute function in
<code>q</code>	another auxillary parameter - information to be used by fct
<code>prt</code>	print information number: = 0 no print, =1 print to screen and =2 print to window
<code>itors</code>	number of iterations

continued on next page

Output:

x	parameter value giving the minima
f	value of function at the minima
g	gradient at x
h	(approximative) hessian at x
ret	return code
iters	number of iterations

Options:

NEWTON	QuasiNewton method (BFGS)
MARQ	Marquardts compromise
SAorSIMUL	Simulated Annealing
GAorGEN	Genetic Evolution algorithm (not yet implemented)
SIMPLEX	Nelder-Meads simplex method

Remark: This proc is meant as an engine for internal optimisation problems. For simpler problems, consider QNewton or CML. One particular feature of it is that it always goes downward! If it can't, it exits with a non-zero return code.

Source: `optalgr.src`

FindMinOneVariable

Purpose: To find optima of unimodular function of one variable

Format: {theta,vof,itors} = FindMinOneVariable(data,theta,&fct,qq,et

Input:

- data data, passed to fct
- theta start value of variable
- &fct pointer to proc, defining the function that should be minimized:
the proc fct should have the format
{vof,g,h} = fct(dataset,theta,qq,typ);
where theta is the parameter to optimise.
- qq data passed to fct
- etype estimation method:

NR Newton-Raphsons method

BRENT Brents method
prt

Output:

- theta value where minimum is attained
- vof function value at minimum
- itors no of iterations

Source: optalgr.src

FindZero

Purpose: Find the zero value of a function of one parameter

Format: `t = FindZero(&fct,dataset,t,q);`

Input:

<code>&fct</code>	pointer to function of form <code>fct(dataset,t,q);</code>
<code>t</code>	2x1-vector with two start values
<code>dataset,q</code>	parameters for <code>fct</code>

Output: `t` value solving $fct(t) = 0$

Source: `optalgr.src`

Frequency

Purpose: Compute frequencies of values, possibly in subsets defined by control variables.

Format: `{group,freqs,nms} = Frequency(dataset,vars,cvars);`

Input:

- dataset** either string, defining dataset on disk, or data matrix
- vars** column numbers or variable names defining variables to compute occurrence of different values in. Negative numbers indicate character columns in a data matrix.
- cvars** 0 or column numbers/variable names defining control variables. Negative numbers indicate character columns in a data matrix.

Output:

- group** GxT -matrix defining groups defined by control variables. G = no. of groups, T = rows(cvars)
- freqs** GxN -matrix with frequencies. G = no. of groups, N = no of all categories on all variables;
- nms** string matrix defining column in freqs in terms of variable and variable value

Globals: `__weight` case weights, vector or column variable

Source: `basstat.src`

GaussWeights

Purpose: to compute Gauss Weights for various types of numerical quadratures

Format: `out = GaussWeights(n,a,name);`

Input:

- `n` order of quadrature
- `a` auxillary parameter(s)
- `name` what type of weights. The following are implemented: "Hermite", "Lagrange", "Laguerre".

Output: `out` nx2-matrix with points and weights.

Remark: Adapted from 'Numerical Recipies in C, Second Edition'.

Source: `integrat.src`

GEE

Purpose: To estimate either a GEE (Generalized Estimating Equation)

Format: `q = GEE(dataset,dep,indep,fct,vfct,x0,covmthd);`

Input:

- dataset** matrix in memory or a string defining a Gauss dataset on disk
- dep** column number or name of dependent variable in dataset
- indep** column numbers or names of independent variables in dataset The FIRST one MUST be the cluster variable (typically PATIENT)
- fct** Pointer to a proc that computes the regression function for individual data. It can take either the form `f(data)` or `f(b,data)`. Here 'data' is individual data excluding subject column and `b` the regression parameters. For a GLM-model, if the input to the link model is linear (obtained from matrix multiplication with the parameter vector) then `fct` can take the form `f(data)`, outputting only the data matrix for the individual. Instead `vfct` should be a string, defining the appropriate link function.
- vfct** pointer a proc that computes a variance matrix. Could also be a string defining standard variance functions. Implemented are string whose upper-case version starts with one of "BINOM", "MULTI", "POISS".

continued on next page

`x0` Start value for regression parameters. If `x0 = {.}` this is a signal to GEE that we have a 'linear' GLM model. This should be specified as a string in `vfct` and now `fct` should take the form `f(data)` which outputs a matrix that can be multiplied by `x0` to obtain the argument for the response function (= inverse link function).

`covmthd` Method to estimate covariance structure .

Globals:

`_rmn_GradMethod` Method to compute numerical gradient

`_rmn_GradProc` Pointer to proc computing gradient of regression function. Should be either of the form `f(b,data)` or `f(b,data,xi)` depending on the type of the regression function In the former case it should have one output, in the latter two: `{g,c}`, where `g` is gradient wrt `b` and `c` gradient wrt `xi`.

`_rmn_phi` Start vector for auxillary parameters. Missing if none.

`_gee_MaxIters` Number of simple iterations before solving as fixed point problem. Default is 50.

`_qn_MaxIters` Maximal allowed number of iterations on the lowest level

`__weight` Case weights. Allows multinomial models to be analysed from summary tables.

`_qn_PrintIters` Controls how the iterative process is displayed.

`__output` Controls if result is printed or not.

`_IntOrd` Order of Gauss-Hermite approximation. Default is 12.

continued on next page

Source: `gee.src`

GenAdmInfo

Purpose: To help in the construction of a patinfo matrix for use with PKNONPARAMETRICS

Format: `patinfo = GenAdmInfo(data,keys,doseiv,doseniv,adm);`

Input:

<code>data</code>	concentration data, data matrix or string defining data on disk
<code>keys</code>	key variables, positive for numerics, negative for character. Variable names can be used if data is on disk
<code>doseiv</code>	iv doses, either a number or matrix with keys followed by doses (more details below). Variable names can be used if data is on disk.
<code>doseniv</code>	non-iv doses - at present only a single dose is taken care of
<code>adm</code>	a matrix with general administration information. It should have 3 or 4 columns (the fourth is optional):
<code>adm[1]</code>	administration code: "iv", "sd" or "ss"
<code>adm[2]</code>	row number in keys for variable
<code>adm[3]</code>	value of key variable
<code>adm[4]</code>	if iv-administration, give infusion time, is sd give 0 if ss give dosing interval in appropriate time units

Output: `patinfo` data matrix for use in PKNONPARAMETRICS (see this)

Source: `pkprocs.src`

GetRegPars

Purpose: to extract estimated parameters and their covariance matrix from a packed vector output from a statistical estimation procedure

Format: `{beta,cov,df} = GetRegPars(q);`

Input: `q` packed vector, output from a STATEST procedure

Output:

- `beta` estimated regression coefficients
- `cov` covariance matrix of beta
- `df` residual degrees of freedom (or missing)

Source: `lmutil.src`

GLM

Purpose: To estimate a generalized linear model

Format: `q = GLM(dataset,dep,class,indep,cfg,glmmodel);`

Input:

- dataset** either string with name of dataset on disk or data matrix in memory
- dep** dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers. For categorical models character data can be used. Should be defined with a negative number for a matrix in memory.
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep
- cfg** string giving model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column
- glmmodel** string defining which model to use. This string should define which distribution family and link function to use.

continued on next page

Options:

- DISP indicating that overdispersion is assumed. Not an option for Gaussian, Gamma or Inverse Gaussian distributions where it is always in effect.
- NOGROUP If there is a weight variable and data is in memory, data is usually maximally grouped in order to be able to obtain correct deviances tests. This option turns grouping off,
- PEARSON Dispersion is by default estimated using deviance (SAS way). You can use the Pearson estimate instead with this option.

Output: `q` packed vector. Information can be extracted using `vread()`.

Remark: At present a CONSTANT IS ALWAYS added to the model

Source: `glm.src`

GLMCheck

Purpose: To assess the fit of a GLM model.

Format: GLMCheck(q);

Input: **q** packed vector output from proc GLM

Remark: This proc is very much under construction. At present only logistic regression is handled.

Source: glmcheck.src

GLMEM

Purpose: To estimate a Generalized Linear Mixed Effects Model

Format: `q = GLMEM(dataset, dep, indep, fct, vfct, dmat, x0, glmmodel, mthd);`

Input:

dataset	matrix in memory or a string defining a Gauss dataset on disk
dep	column number or name of dependent variable in dataset
indep	column numbers or names of independent variables in dataset The FIRST one MUST be the cluster variabel (typically PATIENT)
fct	Pointer to a proc that computes the regression function for individual data. It can take either the form $f(b, data)$ or $f(b, data, xi)$. In all cases 'data' is individual data excluding subject column and b the regression parameters. The first form is used for a mixed effects model where a non-trivial design matrix is used. More precisely, $f(b, data, xi) = g(dsg(b, data, xi), data)$ where the dsg-proc is not of the simple form $b+xi$. Control is exhibited using <code>_mem_random</code> [see below]. For the first case, see <code>x0</code> below.
vfct	pointer a proc that computes a variance matrix.
dmat	Specifies structure of
x0	Start value for regression parameters.
glmmodel	Specifies the GLM to analyse
mthd	Specifies estimation method

continued on next page

Globals:		
	<code>_mem_random</code>	Defines what parameters are to be random Default: 0 = all parameters are random Simple numerical list. If a regression function of the type $f(b,data,xi)$ is defined, <code>_mem_random = -n</code> , where <code>n</code> is the number random parameters.
	<code>_glc_crit</code>	Criteria for when convergence in GEE is declared. Default 1e-6.
	<code>_gee_MaxIters</code>	Number of simple iterations before solving as fixed point problem. Default is 50.
	<code>_qn_MaxIters</code>	Maximal allowed number of iterations on the lowest level
	<code>__weight</code>	If a binary or multinomial problem is analysed, data can be given as classes with number of observations in each class. This variable should then be the variable number (or name) for this data.
	<code>_qn_PrintIters</code>	Controls how the iterative process is displayed.
	<code>__output</code>	Controls if result is printed or not.
	<code>_mem_d_start</code>	Start matrix for random covariance matrix. If non-zero it overrides the internal computation of a start matrix
	<code>_rmn_phi</code>	Start vector for auxillary parameters. Missing if none.
	<code>__IndEstIters</code>	Number of iterations used within the Vonesh-Carter process of estimating individual random parameters. Default is 3.
	<code>_IntOrd</code>	Order of Gauss-Hermite approximation. Default is 5.

continued on next page

Source: `glmem.src`

GLMEMANOVA

Purpose: To analyse an ANOVA model with mixed effects

Format: `q = GLMemANOVA(dataset,dep,class,indep,cfg,glmmodel,estmthd,`

Input:

- `dataset` either string with name of dataset on disk or name of matrix already in memory
- `dep` dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers
- `class` class variables (factors). Format as for dep
- `indep` independent variables (covariates). Format as for dep
- `cfg` string giving model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column

Globals: `_mem_random` list of parameters that are to be random if =0, only constant is random

Source: `glmem.src`

GLS

Purpose: Generalized Least Squares estimation. Only univariate regression.

Format: `q = GLS(dataset,dep,indep,&rfct,&vfct,b,phi);`

Input:

- dataset** data matrix or string defining dataset on disk. Must be sorted on cluster id (subject number)!
- dep** column number or variable name of dependent variable
- indep** column numbers or variable names of regression variables. The first MUST be cluster id (subject number). If only one subject is analysed, without subject number, let `indep[1] = 0!`
- &rfct** pointer to proc of format `rfct(b,x)`, where `x` is the regression data for one subject (without subject number, so `x` is `dataset[:,indep[2:rows(indep)]]` for one subject)
- &vfct** pointer to proc of format `vfct(m,x,phi)`, where `m = rfct(b,x)` and `x` as above. If `&vfct = 0`, ordinary nonlinear regression will be done.
- b** start value for regression parameters
- phi** start value for auxillary variance parameters

Output: `q` packed vector output

continued on next page

Options:

<code>INDIVIDualestimates</code>	forces GLS to estimate variance parameters individually
<code>REML</code>	Restricted Maximum Likelihood estimation of variance parameters
<code>ABSOLUTEresiduals</code>	estimation of variance parameters using Absolute residuals

Remark: The default action of GLS is a pool estimate of variance parameters using the PseudoLikelihood method.

Source: `gls.src`

GMANOVA

Purpose: MANOVA analysis in the presens of missings in dependent variable. Allows for variable specific baseline corrections.

Format: `q = GMANOVA(dataset,dep,class,indep,cfg,baseline);`

Input:

- dataset** either string with name of dataset on disk or name of matrix already in memory
- dep** dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep
- cfg** string giving basic model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column
- baseline** either string matrix or variables defining baseline variables to the individual dependent variables. If no baseline variables is to be used, set that entry to 0. If `baseline = 0`, no baseline variables are used. Has to be 0 or have the same number of rows as dep!

Output: `q` a packed vector output

continued on next page

Globals:

<code>_FactorNames</code>	names for class variables when reading from memory
<code>--output</code>	0 if no output
<code>_RegNames</code>	names for indep variables when reading from memory
<code>_rmn_conflevel</code>	confidence level
<code>--weight</code>	Case weights

Source: `gmanova.src`

Gmeanc

Purpose: Computes the geometric means of a set of variables contained in the columns of a matrix. Pairwise deletion of missings

Format: `m = Gmeanc(x);`

Input: `x` NxK matrix

Output: `m` Kx1 vector containing the geometric means of each column of `x`.

Source: `basstat.src`

GoldenRatioSearch

Purpose: Minimise 1-parameter functions using golden ratio search

Format: `x = GoldenRatioSearch(&f,dataset,x0,q);`

Input:

<code>&f</code>	pointer to proc defining function <code>f(dataset,x,q)</code>
<code>x0</code>	2x1-vector, start interval
<code>dataset,q</code>	data that is passed to <code>f</code>

Output: `x` minimum value found by the golder ratio search

Source: `optalgr.src`

Graph2D

Purpose: Plotting of curves - calls XY, LOGX, LOGY or LOGLOG

Format: Graph2D(x,y);

Input: same as to XY etc

Globals: `_plstep` scalar or Kx1 vector of 0 and 1:s, defining curves that are to be plotted as STEP Functions Default is 0, no step functions

Options: `logx` x-axis logarithmic, y-axis linear
`logy` y-axis logarithmic, x-axis linear
`loglog` both axis logartihmic

Source: Graphics.src

GTS

Purpose: To do General Two-Step Analysis of individually estimated regression parameters

Format: `q = GTS(q,dfct,data,dmat);`

Input:

- `q` packed vector output from GLS
- `dfct` pointer to proc defining design matrix. Takes one row of data as input. If `dfct = 0`, no design
- `data` extra covariate information for each subject. One row per subject.
- `dmat`

Output: `q` packed vector output

Source: `gts.src`

HodgeLehmannEstimate

Purpose: To compute Hodge-Lehmann estimates of location parameter, with confidence limits

Format: `out = HodgeLehmannEstimate(dataset, dep, indep);`

Input:

- `data` data matrix
- `dep` column no(s). of dependent variable
- `indep` column no. of group variable (or 0)

Globals: `_rmn_conflevel` defines confidence level (default = .95)

Output: `out` 1x3-vector containing estimate, lower/upper confidence limits of $P(X1|X2)$ of two groups, otherwise $G \times 5$ -matrix where the first two columns define which groups are compared

Remark: If `indep=0`, `dep` can be a row of column indices. Then pairwise one-sample estimation is done. If `indep/=0`, `dep` can only be one index, and then pairwise estimates for all groups defined by `indep` is done.

Source: `nonpar.src`

IsOption

Purpose: Returns a boolean 0/1 if a given string is contained in any of the rows of `__options`

Format: `x = IsOption(what);`

Input: `what` string or string array (case is not important)

Output: `x` vector of length `rows(what)`. If `what` is part of any of the rows of `__options`, 1 is returned. Otherwise 0.

Source: `options.src`

LinearRegression

Purpose: Linear Regression either from dataset on disk or matrix in memory. Model of form $X = A b$.

Format: `q = LinearRegression(dataset,dep,indep,cfg);`

Input:

<code>dataset</code>	string, name of Gauss dataset or name of matrix in memory
<code>dep</code>	dependent variable or column index
<code>indep</code>	independent variables or column indices
<code>cfg</code>	string, linear model definition.

Output: `q` a packed vector

Globals:

<code>_RegNames</code>	string array, names of the independent variables
<code>_rmn_conflevel</code>	confidence level
<code>__output</code>	controls output
<code>__weight</code>	case weights

Options: NCON no intercept (default is that intercept is added)

Source: `Ancova.src`

LinearTest

Purpose: To do linear test in linear models

Format: `out = lineartest(q,d,lbl);`

Input:

- `q` packed vector, output from a STATEST procedure
- `d` Rxrows(beta)-matrix defining linear test
- `lbl` string giving name of test

Output: `out` test statistics

Options: `NOHEAD` no table header

Remark: This proc probably needs some reworking !

Source: `lmutil.src`

LinMem

Purpose: To analyse a general Linear Mixed Effects Model

Format: `q = LINMEM(dataset,dep,indep,&fct,&cfn,&Rfct,Dmat};`

continued on next page

Input:

- dataset** string, defining dataset, or data matrix
- dep** scalar(col no) or character, defining dependent variable
- indep** numeric or character vector, defining indep variables The Group Variable MUST be the first entry in indep and dataset must be sorted on this value!
- &fct** pointer to procedure defining the design matrix for regression parameters It's form should be `proc fct(x); retp(a);endp;` where `x = dataset[.,indep[.,2:cols]]` for a subject and `a` is an output matrix with the same number of rows as the input data, and columns corresponding to the regression parameters.
- &cfn** pointer to procedure defining the design matrix for random parameters. It takes the same form as `fct`, except that columns now correspond to random parameters. If 0, a column with ones is used.
- &Rfct** pointer to procedure defining the within group error structure. It is determined up to a proportionality factor. It's form should be `proc Rfct(x,phi); retp(a);endp;` where `x` is the same as for the other function procs and `phi` within-subject parameters. The return value `a` should be a covariance matrix. Note that you have to set `_rmn_phi`. If `Rfct = 0`, i.i.d. error is assumed, an identity matrix is used for `Rfct`.
- Dmat**

Output: **q** packed output vector

continued on next page

Options: ML maximum Likelihood estimation instead of the default,
 which is restricted Maximum Likelihood

Globals:

<code>_rmn_phi</code>	start values for parameters referred to in the proc Rfct Missing if phi-parameter. Note that s^2 should not be included!
<code>_mem_random_structure</code>	scalar, defining restrictions on D 0 - no restrictions 1 - diagonal structure n1—n2... - block structure
<code>_mem_GradMethod</code>	defines how numerical gradients are computed:
<code>_mem_random</code>	see Remark below
<code>_qn_PrintIters</code>	if 1, iteration results are printed
<code>_qn_MaxIters</code>	maximum number of iterations allowed
<code>_qn_RelGradTol</code>	convergence criteria for the relative gradient

Remark: There are two ways to set LINMEM up. Either you define both X and Z in two procs, or you define only X and use `_mem_random` to point to which regression parameters are to be considered random. Note that `_mem_random` must be 0 for this option not to take effect!

Source: Linmem.src

LinMemANOVA

Purpose: To analyse an ANOVA model with mixed effects

Format: `q = LinMemANOVA(dataset,dep,class,indep,cfg,dmat);`

Input:

- dataset** either string with name of dataset on disk or name of matrix already in memory
- dep** dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep
- cfg** string giving model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column

Globals: `_mem_random` - list of parameters that are to be random if =0, only constant is random

Source: `Linmem.src`

LMModel

Purpose: Build design matrix for linear model with class, covariates and interactions.

Format: {aa,n,cfpmat,num, names,nmsindep} =
LMMODEL(dataset,dep,class,indep,cfg);

Input:

- dataset** either string with name of dataset on disk or name of matrix already in memory
- dep** dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep
- cfg** string giving model format. If dataset is string, then cfg is in terms of names of indep; if dataset is matrix, then use x1 to denote column 1

continued on next page

Output:

<code>aa</code>	sums of squares matrix for model with dependent variables appearing first, then constant, then classes and finally covariates
<code>n</code>	2 x 1 matrix with <code>n[1]</code> = no. of observations and <code>n[2]</code> = number of parameters
<code>cfgmat</code>	matrix with each row being a term in the model, as for <code>cfg</code> , class variables occur first
<code>num</code>	matrix with each row being no. of degrees of freedom for the corresponding class term in <code>cfgmat</code>
<code>names</code>	names for class terms corresponding to <code>num</code> and <code>cfgmat</code>
<code>nmsinde</code>	names for covariate terms corresponding to non-class terms in <code>cfgmat</code>

Globals:

<code>__weight</code>	cluster weights
<code>_FactorNames</code>	names for class variables
<code>_RegNames</code>	names for indep variables when reading from memory

Remark: design matrix is always saved in `dpt$desn`

Source: `lmmodel.src`

lvcf

- Purpose:** To transform data according to the principle of last value carried forward (including mean and backward carrying)
- Format:** $y = \text{lvcf}(x);$
- Input:** x NxM-matrix in which rows are transformed
- Output:** y NxM-matrix of transformed data
- Source:** Graphics.src

MakeStringArray

Purpose: to produce a string array on disk from an ASCII-file In the ASCII-file the entries should be separated by @ or ,

Format: `MakeStringArray(infile,dell);`

Input: `infile` name of ASCII-file
`dell` 2x1 vector: `dell[1]` = cell separator, `dell[2]` = row separator

Remark: a .FST-file will be saved on disk in the present library with the same first name as the input file

Source: `Util.src`

MantelHaenszelOR

Purpose: To estimate pooled odds ratio from a series of 2x2-tables. Computes both MantelHaenzels estimate with confidence limits and the ML estimate with exact confidence limits. As default Breslow-Days test for homogeneity is given, though the user can request Zelens test instead for small tables

Format: `qout = MantelHaenszelOR(dataset,dep,indep,wgt,cvars);`

Input: `dataset` Two choices:

1. A Nx4-matrix representing N different 2x2-tables Each table is a row (a b c d). This case is defined by `dep = 0`
2. A data matrix or string defining a Gauss data set

`dep` Dependent (case/control) variable in dataset or 0

`indep` Independent (exposure) variable in dataset

`wgt` Variable defining number of cases in cell

`cvars` List of control variables, conditional on which 2x2-tables are to be formed. If `covars=0` we have only one table to form.

Output: `qout` packed vector output

continued on next page

Globals: `_tab_Replace` String Array which can be used to give strata explicit names. Two cases:
if `dep = 0`, a string vector
else a `Nx3`-array where first entry contains variable number and the second a list of existing codes and the lastnames, separated by `@`.

Options: `EXACT` generates exact confidence limits for pooled OR
`ZELEN` the homogeneity test to be computed is Zelens exact test. This consumes memory and can only be done on small datasets.

Source: `cmh.src`

MeanCI

- Purpose:** To compute confidence limits (and P-values) of variables
- Format:** `out = MeanCI(dataset,dep,cvars);`
- Input:**
- dataset** data matrix or dataset on disk
 - dep** column nos. of dependent variables or variable names
 - cvars** column nos. or variable names of control variables, defining groups within which computations are done. Negative numbers for character columns
- Output:** `out` matrix, depending on input: if there is one dependent variable:
1 - Group (if `cvars/=0`) 2-N 3-Mean 4-CIlow
5-CIhigh 6-P (optional) 6/7-SD/CV
if there are more than one dependent variable, a column is concatenated on the left defining which variable the row corresponds to
- Globals:** `_rmn_conflevel` confidence degree
- Options:**
- `LOG` geometric means with CI's and CV(%) (instead of std) is computed
 - `P-Value` (two-sided) p-value is computed
 - `BOOT` bootstrapped CI:s, std:s and p-values
- Source:** `basstat.src`

MeanDifferenceCI

- Purpose:** To compute confidence limits (and P-values) of variables
- Format:** `{out,outwg} = MeanDifferenceCI(dataset,dep,group,cvars);`
- Input:**
- `dataset` data matrix or dataset on disk
 - `dep` column nos. of dependent variables or variable names
 - `group` column nos. of group variables or variable name
 - `cvars` column nos. or variable names of control variables, defining groups within which computations are done. Negative numbers for character columns
- Output:**
- `out` matrix, depending on input: if there is one dependent variable:
1:G - Group variables (if `cvars/=0`) G+1-N
G+2-Mean G+3-CIlow G+4-CIhigh G+5-Pooled
SD/CV G+6-df G+7-P (optional)
if there are more than one dependent variable, a column is concatenated on the left defining which variable the row corresponds to
 - `outwg` the same output as from `MeanCI` with `cvars = group—cvars`
- Globals:** `_rmn_conflevel` confidence degree

continued on next page

Options:

LOG	geometric means with CI's and CV(%) (instead of std) is computed
P-Value	(two-sided) p-value is computed
BOOT	bootstrapped CI:s, std:s and p-values

Source: basstat.src

Medianc

Purpose: Computes the medians of a set of variables contained in the columns of a matrix. Pairwise deletion of missings.

Format: `m = Medianc(x);`

Input: `x` NxK matrix

Output: `m` Kx1 vector containing the medians of each columns of `x`.

Source: `basstat.src`

Merge

Purpose: To merge two data matrices/string arrays, two data sets or a data set and a data matrix according to key variables

Format: `out = Merge(data1,key1,data2,key2,names);`

Input:

- `data1` first data set
- `key1` indicies/variable names of key variables for data1
- `data2` second data set
- `key2` indicies/variable names of key variables for data2
- `names` 2-column matrix of names for columns of matrices, possibly padded with missings. if 0 columns in data will be called X01,X02,.. and Y01,Y02,... respectively

Output: `out` merged data or name of data set: first key variables, then what remains of first data1 and then data2

Globals:

- `_rmn_prob1` used of nonuniqueness in keyvariables
- `_rmn_names` names of columns in out

Source: `Restruct.src`

MultiHermiteWeights

Purpose: Compute points and weights for a numerical quadrature of a k -dimensional integral using Gauss-Hermite quadrature

Format: `out = MultiHermiteWeights(n,k);`

Input: n order of integration
 k dimension of integral

Output: `out` $n \times (k+1)$ -matrix containing points in the first k columns (one row gives one point) and weight in the last

Source: `integrat.src`

MultiNLREG

Purpose: Estimate a Nonlinear Regression Model

Format: `q = MultiNLREG(dataset,dep,indep,fct,x0);`

Input:

<code>dataset</code>	matrix in memory or a string defining a Gauss dataset on disk
<code>dep</code>	column number or name of dependent variable(s) in dataset
<code>indep</code>	column numbers or names of independent variables in dataset
<code>fct</code>	Pointer to a proc that computes the regression function for individual data. It should take the form $f(b,data)$ where b is the regression parameter and $data$ the matrix of independent data.
<code>x0</code>	start value for regression parameters

Output: `q` a packed vector of output information

continued on next page

Globals:	<code>_rmn_GradMethod</code>	Way to calculate numerical gradient: 0 = forward difference , 1 = central difference 2 = Richardson extrapolation
	<code>_rmn_GradProc</code>	Pointer to proc computing gradient of regression function. Should be of the form f(b,data).
	<code>_qn_MaxIters</code>	Maximal allowed number of iterations on the lowest level
	<code>--weight</code>	Either a column vector of weight or a column index or variable name defining a weight variable in dataset
	<code>_qn_PrintIters</code>	Controls how the iterative process is displayed. = 0 No display = 1 All is displayed on screen = 2 Iterations in window
	<code>--output</code>	Controls if result is printed or not.
Source:	<code>nlreg.src</code>	

NewLinearPars

Purpose: To compute estimate and covariance matrix of `bnew` from the equation $D*b = B*bnew$

Format: `{bnew,covnew,H} = NewLinearPars(b,cov,D,B);`

Input:

- `b` parameter estimates
- `cov` covariance matrix of parameter estimates
- `D` matrix of contrasts
- `B` design matrix for new parameter estimates

Output:

- `bnew` new parameter estimates
- `covnew` covariance matrix of new parameter estimates
- `H` the matrix that relates `bnew` and `b`: $bnew = H*b$

Source: `lmutil.src`

NLConfInterval

Purpose: Given that b is normally distributed with covariance matrix cov , this proc estimates confidence intervals for a function $f(b)$ of b based on profiled likelihood estimation (normal distribution, using the estimated cov as true).

Format: `ci = NLConfInterval(b,cov,df,&fnc,lbl);`

Input:

- `b` parameter estimate
- `cov` covariance matrix for b
- `df` degrees of freedom, missing if not relevant
- `&fnc` pointer to proc defining the function $f(b)$
- `lbl` string, name of parameter estimated

Output: `ci` matrix with output

Options:

- `P-VALUE` P-values are outputted
- `NOHEAD` there is no Table Header
- `LOG` exponentiation is done

Source: `confint.src`

NLREG

Purpose: Estimate a Nonlinear Regression Model.

Format: `q = NLREG(dataset,dep,indep,fct,x0);`

Input:

<code>dataset</code>	matrix in memory or a string defining a Gauss dataset on disk
<code>dep</code>	dependent variable(s) in dataset
<code>indep</code>	independent variables in dataset
<code>fct</code>	Pointer to a proc that computes the regression function for individual data. It should take the form <code>f(b,data)</code> where <code>b</code> is the regression parameter and <code>data</code> the matrix of independent data.
<code>x0</code>	start value for regression parameters

Output: `q` a packed vector of output information

continued on next page

Globals:	<code>__output</code>	Controls if result is printed or not.
	<code>_rmn_dep_constraints</code>	Defines possible constraints on dependent variables.
	<code>_rmn_GradMethod</code>	Method to calculate numerical gradient.
	<code>_rmn_GradProc</code>	Pointer to proc computing gradient of regression function. Should be of the form $g(b, data)$. In the case of multivariate regression, this should be the gradient of $\text{vec}(f(b, data)')$.
	<code>_qn_MaxIters</code>	Maximal allowed number of iterations
	<code>_qn_PrintIters</code>	Display of iterations.
	<code>__weight</code>	case weights, vector or column number/variable

Remark: This proc does listwise deletion, i.e. if there is a missing value in one of the dependent variables, the whole record is deleted.

Source: `nlreg.src`

NONMEM

- Purpose:** To estimate a Nonlinear Mixed Effects Model
- Format:** `q = NONMEM(dataset,dep,indep,fct,vfct,dmat,x0,model);`
- Input:**
- dataset** matrix in memory or a string defining a Gauss dataset on disk
 - dep** column number or name of dependent variable in dataset
 - indep** column numbers or names of independent variables in dataset The FIRST one MUST be the cluster variable (typically PATIENT)
 - fct** Pointer to a proc that computes the regression function for individual data. It can take either the form $f(b,data)$ or $f(b,data,xi)$. In all cases 'data' is individual data excluding subject column and b the regression parameters. The first form is used for a mixed effects model where a non-trivial design matrix is used. More precisely, $f(b,data,xi) = g(dsg(b,data,xi),data)$ where the dsg-proc is not of the simple form $b+xi$. Control is exhibited using `_mem_random` [see below]. For the first case, see `x0` below.
 - vfct** pointer a proc that computes a variance matrix.
 - dmat** Specifies structure of
 - x0** Start value for regression parameters.
 - model** NONMEM approximation method used

continued on next page

Globals:	
<code>_rmn_GradMethod</code>	Method to calculate numerical gradient:
<code>_rmn_GradProc</code>	Pointer to proc computing gradient of regression function. Should be either of the form $f(b,data)$ or $f(b,data,xi)$ depending on the type of the regression function In the former case it should have one output, in the latter two: $\{g,c\}$, where g is gradient wrt b and c gradient wrt xi .
<code>_mem_random</code>	List of parameters that are random. $0 =$ all parameters are random If a regression function of the type $f(b,data,xi)$ is defined, <code>_mem_random = -n</code> , where n is the number random parameters.
<code>_gls_crit</code>	Criteria for when convergence in GEE is declared. Default $1e-6$.
<code>_gee_MaxIters</code>	Number of simple iterations before solving as fixed point problem. Default is 50.
<code>_qn_MaxIters</code>	Maximal allowed number of iterations on the lowest level
<code>__weight</code>	Cluster [NB!] weights
<code>_rmn_phi</code>	Start vector for auxiliary parameters. Missing if none.
<code>_rmn_VCIters</code>	Number of iterations used within the Vonesh-Carter process of estimating individual random parameters. Default is 3.
<code>_IntOrd</code>	Order of Gauss-Hermite approximation. Default is 5.

Source: `nonmem.src`

NonParametricTest

Purpose: Nonparametric hypothesis testing

Format: `q = NonParametricTest(data,dep,indep,test)`

Input:

<code>data</code>	NxK matrix with data
<code>dep</code>	indices of dependent variables
<code>indep</code>	indices of independent variables
<code>test</code>	string defining the nonparametric test The following strings are detected (case unimportant): WILCOX - gives signed Wilcoxon Test if indep=0, Wilcoxon or Kruskal Wallis if indep/=0 NORM or VAN DER W - gives a Wilcoxon test on normal scores (Normal score/van der Waerden test). LOGRANK or SAV - gives a Wilcoxon test on Savage scores KRUSKAL - Kruskal Wallis test (same as "WILCOX") KLOTZ - Klotz test MOOD - Mood test CON - Conover test

continued on next page

Remark: all these are Wilcoxon-type tests, they all call an internal proc WILCOX with different options
JONCK, TERP or JT computes the Jonchhere-Terpstas test for dose response
PAGE - Pages test
ALIGN - the Aligned ranks test
FRIED - Friedmans test
THEIL - Theils test

Output: `q` packed matrix with output information.

Globals: `_rmn_strata` Lx2-vector with column numbers and names of stratification variables. Affects only many-sample tests. Negative column indices for character data. Remark: Calls test-specific intermediate procs, which in turn calls RANKTEST to compute moments of test statistics from output from LINRANK. Thus options affecting these also affects this proc.

Example: A Normal Score test can be called by
`AddOption("Normal scores"); call
NonParametericTest(data,dep,indep,"Wilcoxon test");`
(this is what is done internally), and other linear rank tests can similarly be obtained. The option is passed to LINRANK. Signed Wilcoxon-type tests: if `rows(dep)==2`, differences are computed. By default 0s are included in ranking. If you do not want that, use the option "NOZERO".

continued on next page

Source: `nonpar.src`

PDFEstimate

Purpose: To do kernel estimate of probability distribution of a datavector.

Format: `{x,p} = PDFEstimate(data,bandwidth);`

Input: `data` vector with data
`bandwidth` 0 if determined by crossvalidation, otherwise value

Output: `x` x-values for plotting
`p` $p(x)$

Source: `pdfest.src`

PKMeanCurves

Purpose: Calculates mean concentrations for each key combination using estimated values for all values below LOQ and missing values and interpolating to given scheduled times. The mean concentrations are adjusted for dose and give the mean concentration for dose = 1.

Format: `out = PKMeanCurves(data,vars,keys,patinfo,kel,stimes);`

continued on next page

Input:	data	A matrix containing the data, or string defining dataset on disk.
	vars	3x1 vector with column numbers for the actual sampling time, concentration and operator (1 = j). If data is on disk, variable names can be used. Operator is optional (see remark)
	keys	kx1 vector with column numbers for keys, the first key must refer to patient and be numeric. Other keys can be numeric or character (indicate with - before column number). If data is on disk, variable names can be used
	patinfo	px(k+3) matrix where the first k columns correspond to the keys in data, column k+1 is the dose for that key combination and column k contains the infusion time in case of i.v., 0 for a single dose or the dosing interval (minutes) in case of repeated administration. Last column should define what type of administration ("iv", "sd" or "ss").
	kel	nx(k+4) matrix with terminal elimination rate information containing the output from EliminationRate, i.e. keys, intercept, slope, start and end time. If kel=0, no information is used. IMPORTANT SPECIAL CASE: If kel is estimated from non-iv information, but common to all treatments, the first two parameters MUST be PATIENT, RATE
	stimes	times where the means are to be calculated.

continued on next page

Output: `out` `wx(k+1)` matrix with keys (except patient no.), stimes and estimated concentrations for stimes.
 `--pkestimates` `rx(k+2)` matrix. A global variable containing estimated individual data.

Remark: If operator is included, data will pass through LOQEstimation, otherwise not. The algorithm is described in the written documentation of the procedure.

Source: `pkprocs.src`

PKNonParametrics

Purpose: To estimate basic non-parametric pharmacokinetic parameters

Format: `{ivpars,nivpars} = PKNonParametrics(data,vars,keys,kel,patinfo)`

Input:

<code>data</code>	A matrix containing the data or a string defining a dataset on disk.
<code>vars</code>	(2)3x1 vector with column numbers for the actual sampling time, concentration and operator ($1 = j$). Operator is optional. If data is a dataset on disk, variable names can be used.
<code>keys</code>	kx1 vector with column numbers for keys, the first key must refer to patient and be numeric. Other keys can be numeric or character (indicate with - before column number). Again variable names can be used if data is on disk.
<code>kel</code>	nx(k+4) matrix with terminal elimination rate information containing the output from EliminationRate, i.e. keys, intercept, slope, start and end time. If kel=0, no information is used. A string defining a dataset with with precisely the same content can also be used
<code>patinfo</code>	px(k+3) matrix where the first k columns correspond to the keys in data, column k+1 is the dose for that key combination and column k+2 contains the infusion time in case of i.v., 0 for a single dose or the dosing interval (minutes) in case of repeated administration. The last column indicates what type of administration it is ("iv", "sd" or "ss")

continued on next page

Output: `ivpars` Estimated parameters from iv data, by subject. First line contains column names. `nivpars` - Estimated parameters from non-iv data, by subject and administration. First line contains column names. Option: "DOSE AFTER" has effect on plasma concentrations obtained in steady state. Data obtained after end of dosing interval is deleted before any calculations are done. It is to be used in the situation where the dosing interval over which calculations are done is followed by another dose. The option takes away all data after the end of the dosing interval before any calculations are done. Remarks: A. To construct `patinfo`, see `GenAdmInfo` below B If operator is included, data will pass through `LOQEstimation`, otherwise not. C. Multiple IV:s At present only one case of multiple IV is implemented: the case when a tagged substance is injected simultaneously to the non-iv administration. In order to analyse this case, the `LAST` column of keys must define a character column of data that contains code for drugs (like 'BUD' and 'BUD_D8'). The simultaneous case is identified from the fact that iv and non-iv have different drug codes!

Source: `pkprocs.src`

PlotCurves

Purpose: Transforms data into data appropriate for plotting in GAUSS

Format: `{t,x} = PlotCurves(data,keys,vars);`

Input:

- `data` data matrix
- `keys` column nos. with key data (negative for characters)
- `vars` vector giving column numbers for curves `vars[1]` MUST be jump times remaining columns give curves to plot on these times There is one set of curves for each key combination

Output:

- `t` pxn-matrix of x-data for curves
- `x` pxn-matrix for y-data for curves

Globals:

- `_rmn_names` is an output global, which defines what key combinations defined the different columns of `t` and `x`: `_rmn_names[k,.]` gives key kombination for curve `(t[:,k],x[:,k])`
- `_plmaxtime` time at which curves should stop. This variable is only used with the option 'EXTEND CURVES'

Source: `Graphics.src`

PredictedSurvivalFunction

- Purpose:** To estimate the baseline cumulative hazard and survival function in a Cox Regression Model
- Format:** {surv, means} = PredictedSurvivalFunction(baseline,q)
- Input:**
- baseline** matrix, where each row consists of values of covariates for which the survival function should be computed. If time-dependent covariates are wanted, use `_cp_basefnc`
 - q** packed vector output from CensoredRegression or CoxRegression

continued on next page

Output:

surv in the semi-parametric case a matrix with following columns
1:n key values, n=0,1 or 2. If there are more than one baseline set to compute for, row in baseline is indicated in the first column. If stratification is in effect, stratification value is given next
n+1 jump times
n+2 number at risk at jump time
n+3 number of jumps at jump time
n+4 cumulative hazard function estimator
n+5 standard deviation of the cumulative hazard funct. est.
n+6,n+7 confidence limits for the cum. haz. func. est.
n+8 Survival Function estimator
n+9 standard deviation of the Surv. func. est.
n+10,n+11 confidence limits for the Surv. func. est.

mean matrix with following columns
1:n key values as above
n+1 estimated mean survival time
n+2 estimated standard deviation of this

Options:

ciplain plain scale for confidence limits
ciloglog loglog-survival scale for confidence limits
Peto the lower confidence limit is computed using Petos method
Kaplan-Meier use Kaplan-Meier type of approximation instead of Flemming-Harrington (because this is what SAS uses).

continued on next page

Globals: `_cp_basfnc` pointer to a proc that computes time-dependent covariates. Should be a function of a time vector. It should output the same no of rows as the input vector, and the same no of columns as there are rows in `b`.

Remark: If the case of a frailty model some columns are missing, because I have not yet figured out how to do them. Also, the options do not affect this case, only the counterpart to the `ciloglog`-scale is implemented. The confidence limits for the survival function does not account for the fact that the gamma parameter was estimated!

Source: `Coxdiag.src`

Quantiles

Purpose: Computes quantiles of a distribution

Format: `y = Quantiles(dataset,vars,cvars,e);`

Input:

- dataset** string, dataset name, or NxM matrix of data
- vars** Kx1 vector or scalar zero. If Kx1, character vector of labels selected for analysis, or numeric vector of column numbers in data set of variables selected for analysis. If scalar zero, all columns are selected. If dataset is a matrix, var must be a vector of column numbers
- cvars** column nos. or variable names of control variables, defining groups within which computations are done. Negative numbers for character columns
- e** Lx1 vector, quantile levels or probabilities

Output: `y` LxK matrix, quantiles

Options: `INTERPOL` means that linearly interpolated quantiles are computed (instead of the ones obtained from the Empirical CDF)

Remark: The difference between this and the built-in `quantile`, is that missings are ignored and medians correctly computed.

Source: `quantil.src`

QuasiScoreEstimation

Purpose: To estimate a quasi-likelihood model

Format: `q = QuasiScoreEstimation(dataset,dep,class,indep,cfg,rfnc,vfnc,x0)`

Input:

- dataset** either string with name of dataset on disk or name of matrix already in memory
- dep** dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep
- cfg** string giving model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column
- rfnc** pointer to proc defining the response function
- vfnc** pointer to proc defining the variance structure
- x0** start vector for regression coefficients

Globals:

- `_quasi_varfnc` pointer to proc which estimates auxillary variance parameters, if appropriate.
- `_rmn_phi` if `_quasi_varfnc` $\neq 0$, give start values here
- `__weight` column number or vector of weights

Output: `q` packed vector.

Remark: At present a CONSTANT IS ALWAYS added to the model

Source: `quasi.src`

RecodeInString

Purpose: To replace specified entries with other specified entries

Format: `Outstring = RecodeInString(Instring,what,with);`

Input:

<code>Instring</code>	string array
<code>what</code>	string or string array to be replace by
<code>with</code>	replacement is done row-wise (<code>what[k] - ></code> <code>with[k]</code>).

Output: `Outstring` `Instring` with `what` replaced by `with`

Source: `Util.src`

RelativeDoseAnalysis

Purpose: Compute equivalent dose or relative dose potency with confidence intervals for mean effects.

Format: `{rd,eta} = RelativeDoseAnalysis(beta,cov,df,d,bmat,name);`

Input:

- `beta` parameter estimates
- `cov` covariance matrix of parameter estimates
- `df` residual degrees of freedom
- `d` contrast matrix such that $d \cdot \text{beta}$ are the mean values to which model is to be fitted
- `bmat` design matrix for problem. $T \times 3$ -matrix for T treatments. First two columns define drug belonging whereas doses are given in the third.
- `name` name of parameter to be estimated (for output only)

Output:

- `rd` 1×3 vector with dose/dose potency estimate and confidence limits
- `eta` contains regression parameters - for plotting

Source: `lmutil.src`

RemoveOption

Purpose: Removes an option to the global variable `__options`

Format: `RemoveOption(str);`

Input: `str` the string to be deleted from `__options` Several options can be removed at once, if parted with `,`

Example: `RemoveOption("option1,option2,option3")`

Source: `options.src`

ReplaceInString

Purpose: To scan a string or string array and replace a specified sequence with another specified sequence

Format: `Outstring = ReplaceInString(Instring,what,with);`

Input: `Instring` string or string array
`what` string or string array to be replace by
`with` Replacement is done row-wise (`what[k] - > with[k]`).

Output: `Outstring` `Instring` with `what` replaced by `with`

Source: `Util.src`

ResidualPlot

- Purpose:** To plot residual for diagnostic purposes
- Format:** `ResidualPlot(q);`
- Input:** `q` packed vector output of STATEST proc
- Source:** `diagnos.src`

Restruct

- Purpose:** To transpose data along one variable, within others
- Format:** `out = Restruct(data,keys,ind,vars,names);`
- Input:**
- `data` data matrix, string array or name of Gauss data set on disk
 - `keys` indices/names of key variable
 - `ind` index/name of reconstruction variable
 - `vars` indices/names of variables to restruct according to `ind`
 - `names` if data matrix or string array: vector of names for columns if 0 columns in data will be called X001,X002,.. If data set on disk, name of restructured data set Negative indices for keyvar and `ind` indicates character columns!
- Output:** `out` restructured data matrix
- Globals:**
- `_rmn_prob1` used of nonuniqueness in keyvariables
 - `_rmn_names` names of columns in `out`
- Options:** `NOSORT` don't sort data according to key variables
- Source:** `Restruct.src`

RunR

Purpose: A small proc that runs R and import the output for comparison with the Gauss/Riemann output

Format: `RunR(data,nms,cmdstr);`

Input:

<code>data</code>	data matrix (or 0 if no data matrix is to be used)
<code>nms</code>	character vector defining column names. The GAUSS convention to have numeric data in upper cases and character data in lower cases MUST be adhered to !!! Character data will automatically be transformed to factor elements (this is the only way to get factor elements!)
<code>cmdstr</code>	SPlus command string

Remark: The R output is written to the open output file in GAUSS.

Source: `extern.src`

RUNSAS

Purpose: A small proc that runs SAS and import the output for comparison with the Gauss/Riemann output

Format: RunSAS(data,nms,cmdstr);

Input:

data	data matrix
nms	character vector defining column names. The GAUSS convention to have numeric data in upper cases and character data in lower cases MUST be adhered to !!!
cmdstr	SAS command string

Remark: The SAS output is written to the open output file in GAUSS.

Source: extern.src

RunSPLUS

Purpose: A small proc that runs SPlus and import the output for comparison with the Gauss/Riemann output

Format: RunSPLUS(data,nms,cmdstr);

Input:

<code>data</code>	data matrix (or 0 if no data matrix is to be used)
<code>nms</code>	character vector defining column names. The GAUSS convention to have numeric data in upper cases and character data in lower cases MUST be adhered to !!! Character data will automatically be transformed to factor elements (this is the only way to get factor elements!)
<code>cmdstr</code>	SPlus command string Global: <code>_SPlusVersion:</code> character vector defining version of SPlus

Remark: The SAS output is written to the open output file in GAUSS.

Source: `extern.src`

RunStatXact

Purpose: A small proc that runs StatXact and import the output for comparison with the Gauss/Riemann output

Format: RunStatXact(data,nms,cmdstr);

Input:

data	data matrix
nms	character vector defining column names. The GAUSS convention to have numeric data in upper cases and character data in lower cases MUST be adhered to !!!
cmdstr	StatXact command string

Remark: The StatXact output is written to the open output file in GAUSS. Note!!! When StatXact is started, press F10 to execute the commands.

Source: extern.src

sa2str

Purpose: to convert a string array to a string, where elements in a row is and rows are separated by user defined elements

Format: `str = sa2str(sa,dell);`

Input: `sa` string array
`dell` character/string array defining what characters should delimit elements in row (first element) and what characters should delimit rows (second element)

Output: `str` - string

Source: `Util.src`

SelectFromDisk

Purpose: To select data from a dataset. Typical use is to select data from defined patients.

Format: `out = SelectFromDisk(inds,ix,values,typ,vars);`

Input:

<code>inds</code>	string, defining Gauss data set
<code>ix</code>	variable indices or names of key variables from which selection is done
<code>values</code>	the values of variables <code>ix</code> to select. Should have the same number of columns as there are rows in <code>ix</code> .
<code>typ</code>	1 if selection, -1 if deletion based on key variables
<code>vars</code>	variable indices or names of data which are to be returned. If equal to zero all columns returned

Output: `out` selected data in matrix form

Source: `Dstrans.src`

SortData

Purpose: To sort data according to key variables

Format: `out = SortData(data,keys);`

Input: `data` string, defining Gauss dataset, data matrix or string array
`keys` column indices or variable names on which sorting should be done. Use variable names for Gauss datasets, column indices with negatives for character data for data matrices and column indices for string arrays. In string array case, a positive index will mean that that column is sorted numerically, a negative one that it is sorted as character data.

Output: `out` result, of the same type as data

Remark: Note that a Gauss dataset will be overwritten with the sorted one. If you don't want that, use DiskSort instead.

Source: `Util.src`

str2sa

Purpose: to convert a string into a string array, where elements in a row is and rows (in the string) are separated by user defined elements

Format: `sa = str2sa(str,dell);`

Input: `str` string
`dell` character/string array defining what characters should delimit elements in row (first element) and what characters should delimit rows (second element) If dell contains one element, the output has one column

Output: `sa` string array

Remark: this is the inverse of SA2STR

Source: `Util.src`

SummaryStatistics

- Purpose:** To compute (and print) summary statistics
- Format:** `out = SummaryStatistics(dataset,vars,cvars,e);`
- Input:**
- dataset** string, name of Gauss data set, or data matrix
 - vars** column indices or variable names defining what variables to compute statistics for
 - cvars** column nos. or variable names of control variables, defining groups within which computations are done. Negative numbers for character columns
 - e** vector defining what percentiles to compute
e = 0 means none, e = .5 means median only etc
- Output:** `out` output with defined statistics.
- Globals:** `--output` controls output
- Options:** `LOG` with this option geometric means and CV will be computed instead of arithmetic means and SD
- Source:** `basstat.src`

SUMSOFSQUARES

Purpose: To compute sums of squares matrix for ANOVA models

Format: `{nm,ss,df,m} = SumsOfSquares(q,j);`

Input: `j` the row number in ANOVA output whose SS is to be analysed

Output:

- `nm` name of factor
- `ss` sums of squares matrix
- `df` degrees of freedom for ss
- `m` `inv(sserr)*ss`

Remark: `m` can be used for computation of various test statistics after having computed its eigenvalues

Source: `lmutil.src`

SurvivalFunction

Purpose: To compute NonParametric Estimators of Counting Process data. It estimates the cumulative hazard function as well as the survival function.

Format: `{out, means} = SurvivalFunction(dataset,dep,keys);`

Input:

dataset string, defining Gauss data set, or data matrix

dep variables for
1 = observational start time (optional)
2(1) = observational end time
3(2) = censor variable
The censor variable should take integer values. Only one value means no censoring. More than two values means that a competing risk situation is encountered:
THEN 0 MUST MEAN CENSORED
in other cases the lowest value is taken to indicate censor

keys variables giving key variables (or 0) use negative numbers for character columns

continued on next page

Output:

out matrix with the following columns:
1:n gives keys, n=rows(keys)
n+1 jump times
n+2 no at risk at jump time
n+3 no of jumps at jump time
n+4 Nelson-Aalen estimator
n+5 s.d. of Nelson-Aalen estimator
n+6,n+7 confidence limits for Nelson-Aalen estimator
n+8 Survival Function estimator
n+9 s.d. of Survival Function estimator
n+10,n+11 confidence limits for Survival estimator

means matrix with following columns:
1:n gives key values as above
n+1 estimated mean survival time
n+2 estimated s.d of survival time

Options:

ciplain plain scale for confidence limits
ciloglog loglog-survival scale for confidence limits
Peto the lower conf. limit is computed using Petos method Fleming-Harrington - the survival function estimator with this name
Efron Correction for ties in the computation of the Nelson-Aalen estim.

continued on next page

Globals:

<code>_rmn_conflevel</code>	confidence degree (default is 0.95)
<code>_psym</code>	output global, can be used directly to plot censored observ. in a Kaplan-Meier plot.
<code>_plmaxtime</code>	contains key variables (including risk value if relevant) followed by the maximal time point for this group. This global is used by PlotCurves when plotting survival function plots to extend the left part of the curve.

Source: `Countpr.src`

SurvivalPercentiles

Purpose: to estimate percentiles for survival function based on Kaplan-Meier estimate

Format: `out = SurvivalPercentiles(data,keys,p);`

Input:

- `data` first output from SurvivalFunction
- `keys` key variables in data (characters negative)
- `p` list of percentiles to estimate. If 0, only the median is estimated (as if $p=0.5$)

Output: `out` matrix with key-combination (col 1:n), percentile key followed by estimate of percentile and confidence limits for percentile

Source: `Countpr.src`

SurvivalTest

Purpose: Hypothesis Testing of Survival data, comparing groups. Stratification is allowed

Format: `{z,cov,table} = SurvivalTest(data,times,group,rho);`

Input:

- `data` data matrix
- `times` vector containing column nos. in data containing
 - 1 - start of observational period (optional)
 - 2(1) - observational end time
 - 3(2) - censoring (optional) (integers: smallest=censored)
- `group` column no. giving group variable, negative if character
- `rho` parameter in Fleming-Harringtons family of tests
 - rho = 0 gives log-rank test
 - rho = 1 gives is an approximation of the Peto-Peto modification of the Wilcoxon test
 - rho = "Wilcoxon" means the Gehan-Wilcoxon test

continued on next page

Output:

- `z` deviance (observed-expected) for each group
- `cov` covariance matrix of test statistic `z`
- `table` A summary table of result with columns:
 1. group value
 2. N in group
 3. Observed
 4. Expected
 5. $(O-E)^2/E$

Globals:

- `_rmn_strata` Lx2 matrix containing index of column in data that contains stratification variables, and names for this variable (for table output) `_rmn_strata = 0` means no stratification
- `--output` controls printed output

Source: `Countpr.src`

SymmetricStableRegression

Purpose: does a stable symmetric regression, i.e. the error structure distribution has a log characteristic function given by

$$\ln E[\exp(itX)] = |ct|^a$$

where $a=2$ is the standard Gaussian distribution and $a=1$ the Cauchy distribution. α is constrained to lie in $(0.84,2)$.

Format: `q = SymmetricStableRegression(dataset,dep,class,indep,cfg);`

Input:

- dataset** either string with name of dataset on disk or name of matrix already in memory
- dep** dependent variables. If dataset is string, then string with names of columns, if dataset is matrix, then vector with column numbers
- class** class variables (factors). Format as for dep
- indep** independent variables (covariates). Format as for dep
- cfg** string giving model format. If dataset is string, then cfg is in terms of names of class and indep; if dataset is matrix, then use x1 to denote column

Output: `q` a packed vector output

continued on next page

Globals:

<code>_rmn_conflevel</code>	confidence level, default = 95%
<code>__output</code>	0 if no output, default 2
<code>__weight</code>	string, name of weight variable. By default, non-weighted. If weighted then the weights are contained in - dataset variable with name <code>__weight</code> if reading from disk - dataset in column no. <code>__weight</code> if reading from memory.
<code>_FactorNames</code>	names for class variables when reading from memory
<code>_RegNames</code>	names for indep variables when reading from memory

Source: `stabreg.src`

Table

Purpose: Procedure which can print Tables in LaTeX and HTML format. Fuller description in Riemann Manual.

Format: `stats = Table(data,dec,tabhd,cs,head);`

Input:

- `data` data matrix or string array containing data to be printed
- `dec` list of decimal places to be used for each column. Use -1 for character elements. Will be expanded with the last element if too short. Set to -1 for string array.
- `tabhd` string array defining column headers. Lines are separated by `\and` and cells within a line with `~`. To let an item span `n` lines, start with `@n` (ending with one space!).
- `cs` string, column alignment specification.
- `head` header text. If 0, there will be no table environment in LaTeX mode.

Output: `stats` if column statistics is to be computed and added on last rows, this is outputted.

continued on next page

Options:	LONGTABLE	Use the TeX-macro LongTable
	STAT	Include descriptive statistics at bottom of numeric columns
	GEOM	Use geometric mean and coefficient of variation instead of arithmetic mean and standard deviation in descriptive statistics
	BOX	Box the table
	NOBOX	No box of the table
	NOCENTER	Center the table horisontally
	LANDSCAPE	surround the table with a landscape environment

Globals:	<code>_tab_Replace</code>	a 3-column string array (or character matrix) which can be used to modify entries in columns.
	<code>_tab_ctl</code>	Defines intercolumn space in LaTeX in 'em's
	<code>_tab_charpos</code>	
	<code>_tab_pre</code>	Character matrix of the same size as data. Is added to the left to data. Default is 0.
	<code>_tab_post</code>	Character matrix of the same size as data. Is added to the right to data. Default is 0.
	<code>_tab_u_line</code>	Nx3 matrix with row number, start column and end column to underline in the table header
	<code>_tab_row_ul</code>	Either Nx1 vector with row numbers in main table to underline, or Nx3 matrix with row number, start column and end column to underline

continued on next page

<code>_tab_label</code>	string, defining a table label
<code>_tab_prec</code>	positive number, to be used if all data in table should be given with a fixed precision
<code>_tab_message</code>	string, defining a footnote to the table (is in fact last row in table)
<code>_tab_statistics</code>	list of which (if subset is wanted) statistics to print. Only if option STAT is in effect.
<code>_tab_cmp</code>	Data from which descriptive statistics should be computed, if not data in matrix. Could e.g. have substitutes for values jLOQ. Only if option STAT is in effect.
<code>_tab_nostat</code>	list of column numbers that we do not want descriptive statistics on. Only effective if option STAT is in effect.

Source: `Table.src`

TherapeuticIndex

Purpose: To estimate the relative therapeutic index from efficacy and side-effect mean values

Format: {tr,eta} = TherapeuticIndex(beta,cov,df,d1,d2,bmat1,bmat2,name)

Input:

- beta** parameter estimates (vector!)
- cov** covariance matrix of parameter estimates
- df** residual degrees of freedom
- d1** contrast matrix such that $d1 \cdot \text{beta}$ are the mean values to which the efficacy model is to be fitted
- d2** contrast matrix such that $d2 \cdot \text{beta}$ are the mean values to which the side-effect model is to be fitted
- bmat1** design matrix for efficacy problem. $T \times 3$ -matrix for T treatments. First two columns define drug belonging whereas doses are given in the third.
- bmat2** design matrix for side-effect problem. $P \times 3$ -matrix for P treatments.
- name** name of parameter to be estimated (for output only)

Output:

- tr** row vector with result for relative therapeutic index
- eta** 3×2 matrix with dose-response lines parameters. First column corresponds to efficacy, the second to side-effect.

Source: `lmutil.src`

Time2FirstEvent

Purpose: To trim data that contain multiple events down to first event-data only

Format: `data = Time2FirstEvent(data,id,tid);`

Input:

- `data` data matrix (dataset in future)
- `id` column number of cluster identifier
- `tid` 2x1 matrix with time and event variable

Output: `data` data set of the same form as input-data, except that only first event is included. Event variable is 0 for censored and 1 for event

Source: `Coxreg.src`

UniqueData

Purpose: to get all key-combinations of its entry

Format: `groups = UniqueData(data,cvars);`

Input:

- `data` string defining Gauss dataset, a data matrix, or a string array
- `cvars` indices or names of variables in dataset for which unique key-combinations are to be found positive and negative column numbers depending on numeric or character sorting.

Output: `groups` the existing value combinations of `cvars`

Source: `Util.src`

WilcoxonProb

Purpose: To compute $P(X1|X2)$ with confidence limits for two independent group;

Format: `out=wilcoxonprob(data,dep,indep);`

Input:

- `data` data matrix
- `dep` column no. of dependent variable
- `indep` column no. of group variable

Globals: `_rmn_conflevel` - defines confidence level (default = .95)

Output: `out` 1x3-vector containing estimate, lower/upper confidence limits of $P(X1|X2)$ of two groups, otherwise Gx5-matrix where the first two columns define which groups are compared Reference: Halperin et al, Distribution-free Confidence Intervals for $Pr(X1|X2)$, Biometrics 43, 71-80

Source: `nonpar.src`

WilkShapiro

Purpose: To compute Wilk-Shapiros test for univariate normality and plot residuals vs ordered normal variates

Format: `out = WilkShapiro(x);`

Input: `x` NxP-matrix whose columns will be tested for normality

Output: `out` Px2-matrix : row `i` gives `W` and `P`-value of col `i` of `x`

Options: `NOPLOT` suppresses plotting

Remark: This is an adaptation of the algorithm given by Royston (1982) (Appl. Statist. 31 pp 115-124, 176-180). It differs in that the mean values of the ordered normal variate is computed from Bloms approximation (see `linrank`). Remark: This proc first removes all lines in `x` containing a missing. If you do not want that, you will have to call the proc `column` for `column`.

Source: `diagnos.src`

WNLREG

Purpose: To do a nonlinear regression, weighted by a matrix. Require data to be a data matrix.

Format: `q = WNLREG(dataset,dep,indep,fct,wt,x0,df);`

Input:

<code>dataset</code>	matrix in memory
<code>dep</code>	column number of dependent variable in dataset
<code>indep</code>	column numbers independent variables in dataset
<code>fct</code>	Pointer to a proc that computes the regression function for individual data. It should take the form $f(b,data)$ where b is the regression parameter and data the matrix of independent data.
<code>wt</code>	Weight matrix. If $N=\text{rows}(\text{dataset})$, wt is $N \times N$. Can also be a pointer to a proc which computes the Weight matrix. This should have the form $wt(m,x,phi)$; where $m = f(b,x)$, x the independent data and phi auxillary parameters, controlled by the global <code>_rmn_phi</code> .
<code>x0</code>	start value for regression parameters
<code>df</code>	degrees of freedom. If $df=0$ it is computed by the proc. If $df \neq 0$, weighting is assumed to be by the covariance matrix and dispersion parameter is set to 1.

Source: `nlreg.src`

WorldPlot

Purpose: provides data for plot of (part of) the world. Based on code in MATHEMATICA.

Format: `{x,y} = WorldPlot(countries);`

Input: `countries` string containing names of world parts or countries to plot. 0=global map

Output: `x` longitude data
`y` latitude data

Source: `worldmap.src`