Package ‘utilities’

October 12, 2022

Type Package

Title Data Utility Functions

Version 0.6.1

Date 2022-06-30

Author Ben O'Neill [aut, cre]

Maintainer Ben O'Neill <ben.oneill@hotmail.com>

URL https://github.com/ben-oneill/utilities/

Description Data utility functions for use in probability and statistics. Includes functions for computing higher-moments for samples and their decompositions.
Also includes utilities to examine functional mappings between factor variables and other variables in a data set.

License MIT + file LICENSE

Encoding UTF-8

Imports stats

Suggests ggplot2, ggdag (>= 0.2.4), gmp, gridExtra, matrixStats, scales, VGAM

RoxygenNote 7.2.0

NeedsCompilation no

Repository CRAN

Date/Publication 2022-07-01 06:20:02 UTC

R topics documented:

`.print.data.frame` *Print a Data-Frame (allowing column/row separators)*

Description

Custom print method for objects of type `data.frame`. This function prints the data-frame in the same way as the default print.data.frame in the base package, except that it allows the user to add textual column/row separators in the print output in specified positions. To do this the user adds row/column values for the inputs `row.separator` and `col.separator` indicating that separators should be added after those rows/columns. The user can also set `sep.extend` to `TRUE` to extend the separators into the row/column-names.

Usage

```
.print.data.frame(
  x,
  ...,
  row.separator = NULL,
  col.separator = NULL,
  sep.extend = FALSE,
  print.gap = 1,
  digits = NULL,
  quote = FALSE,
  right = TRUE,
  row.names = TRUE,
  max = NULL
)
```

Arguments

`x`	A data-frame (object of class `data.frame`)
`...`	optional arguments to `print` or `plot` methods
`row.separator`	A vector of values of rows (adds separators after those rows)

`col.separator`	A vector of values of columns (adds separators after those columns)
`sep.extend`	Logical value; if `TRUE` the separators are extended into the row/column-names
`print.gap`	A non-negative integer specifyig the number of spaces between columns
`digits`	the minimum number of significant digits to be used: see `print.default`
`quote`	Logical value; if `TRUE` entries are printed with surrounding quotes
`right`	Logical value; if `TRUE` strings are right-aligned
`row.names`	Logical value or character vector; indicating whether (or what) row names are printed
`max`	numeric or `NULL`, specifying the maximal number of entries to be printed. By default, when `NULL`, `getOption("max.print")` used

Value

Prints the data frame with the specified column/row separators

`datasets.str` *Structure of Available Datasets*

Description

`datasets.str` returns the structure of available datasets

Usage

```
datasets.str(package = NULL)
```

Arguments

`package`	The package/packages containing the datasets of interest

Details

Datasets are often available in packages loaded into `R` and it is useful to know the structure of these datasets. This function shows the user the strucure of all available datasets in a specified package or packages. (If the user does not specify a `package`) then the function searches over all available packages.

Value

A data frame listing available data sets, invisibly

Examples

```
datasets.str("datasets")
```

KDE *Kernel Density Estimator*

Description

KDE returns the probability function for the kernel density estimator

Usage

```
KDE(
  data,
  weights = NULL,
  bandwidth = NULL,
  df = Inf,
  density.name = "kde",
  value.name = "Value",
  discrete = FALSE,
  discrete.warn = TRUE,
  to.environment = FALSE,
  envir = .GlobalEnv
)
```

Arguments

data	Input data for the kernel density estimator (a numeric vector)
weights	Weights for the kernel density estimator (a numeric vector with the same length as the data)
bandwidth	Bandwidth for the KDE; if NULL it is estimated
df	Degrees-of-freedom for the T-distribution
density.name	Name of the KDE distribution; used for naming of the probability functions (a character string)
value.name	Name of the values in the data; used for naming the plot of the KDE
discrete	Logical; if TRUE the function produces a discrete KDE over the integers
discrete.warn	Logical; if TRUE the function gives a warning if non-discrete data is used to produce a discrete KDE
to.environment	Logical; if TRUE the probability functions are attached to the global environment
envir	The environment where the probability functions are loaded (if to.environment is TRUE)

Details

The kernel density estimator for a set of input data is obtained by taking a mixture distribution consisting of a (possibly weighted) combination of kernels. In this function we compute the KDE using the kernel of the T-distribution; the function can also estimate a discretised version of the KDE (taken over the integers) if required. The degrees-of-freedom and the bandwidth for the KDE can be specified in the inputs; if the bandwidth is not specified then it are estimated using the methods set out in Sheather and Jones (1991) used in the `stats::density` function. The output of the function is a list of class `kde` that contains the probability functions for the KDE and associated information. The output object can be plotted to show the density function for the KDE.

Note: The function has an option `to.environment` to allow the user to load the probability functions to the global environment or another specified environment. If this is set to `TRUE` then the probability functions are loaded to the specified environment in addition to appearing as elements of the output; there is a message informing the user if existing objects in the global environment were overwritten. If the functions are not loaded to the environment then the user can use the function `KDE.load` to load them later from the produced object.

Value

A `kde` object containing the probability functions for the kernel density estimator

Examples

```
k <- KDE(rnorm(500))
print(k)
plot(k)
KDE.load(k, environment()); ls()
```

`KDE_utils` *Utilities for KDE fits*

Description

Utilities for KDE fits

`KDE.load` copies KDE distribution functions from a KDE object to a target environment.

`print.kde` prints the KDE object and returns it invisibly.

`plot.kde` draws a plot of the KDE.

Usage

```
KDE.load(object, envir = NULL, overwrite = TRUE)

## S3 method for class 'kde'
print(x, digits = 6, ...)

## S3 method for class 'kde'
plot(
  x,
  digits = 6,
  n = 512,
  cut = 4,
  fill.colour = "dodgerblue",
  fill.color = fill.colour,
  ...
)
```

Arguments

`object, x`	A KDE object
`envir`	The target environment
`overwrite`	If FALSE, aborts if the function names are already present in the target environment
`digits`	Number of digits to print
`...`	unused
`n`	number of bins
`cut`	cutoffs for xaxis (in steps of bw)
`fill.colour, fill.color`	fill color of bars

Value

KDE.load returns `envir`

`print.kde` returns `x`, invisibly

`plot.kde` returns the plot as recorded by `recordPlot`

`kurtosis` *Sample Kurtosis*

Description

`kurtosis` returns the sample kurtosis of a data vector/matrix

Usage

```
kurtosis(x, kurt.type = NULL, kurt.excess = FALSE, na.rm = FALSE)
```

Arguments

`x`	A data vector/matrix
`kurt.type`	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
`kurt.excess`	Logical value; if `TRUE` the function gives the excess kurtosis (instead of raw kurtosis)
`na.rm`	Logical value; if `TRUE` the function removes `NA` values

Details

This function computes the sample kurtosis for a data vector or matrix. For a vector input the function returns a single value for the sample kurtosis of the data. For a matrix input the function treats each column as a data vector and returns a vector of values for the sample kurtosis of each of these datasets. The function can compute different types of kurtosis statistics using the `kurt.type` input.

Value

The sample kurtosis of the data vector/matrix

Examples

```
kurtosis(rnorm(1000))
kurtosis(rexp(1000))
```

`log` *Logarithm Function*

Description

`log` returns the logarithm of the input

Usage

```
log(x, base = exp(1), gradient = FALSE, hessian = FALSE)

log2(x, gradient = FALSE, hessian = FALSE)

log10(x, gradient = FALSE, hessian = FALSE)
```

Arguments

`x`	An input value (numeric/complex scalar or vector)
`base`	The base for the logarithm (a positive scalar value)
`gradient`	Logical; if `TRUE` the output will include a `'gradient'` attribute
`hessian`	Logical; if `TRUE` the output will include a `'hessian'` attribute

Details

The logarithm function in base `R` accomodates complex numbers but it does not accomodate negative values (which is strange). The pressent version of the logarithm function allows both numeric and complex inputs, including negative numeric values. For negative inputs this function gives the princpal complex logarithm of the input value. If the output of the logarithm has no complex part then the output is given as a numeric value. This function also allows the user to generate the gradient and Hessian.

Value

The logarithm of the input

Examples

```
log(1)
log(-1)
log10(-10, TRUE, TRUE)
```

`mappings` *Examine mappings between factor variables in a data-frame*

Description

`mappings` determines the mappings between factor variables in a data-frame

Usage

```
mappings(data, na.rm = TRUE, all.vars = FALSE, plot = TRUE)
```

Arguments

`data`	A data-frame (or an object coercible to a data-frame)
`na.rm`	Logical value; if `TRUE` the function removes `NA` values from consideration
`all.vars`	Logical value; if `TRUE` the function only examines factor variables in the data-frame; if `FALSE` the function examines all variables in the data-frame (caution is required in interpretation of output)
`plot`	Logical value; if `TRUE` the function plots the DAG for the mappings (requires `ggplot2` and ggdag to work)

Details

In preliminary data analysis prior to statistical modelling, it is often useful to investigate whether there are mappings between factor variables in a data-frame in order to see if any of these factor variables are redundant (i.e., fully determined by other factor variables). This function takes an input data-frame `data` and examines whether there are any mappings between the factor variables. (Note that the function will interpret all character variables as factors but will not interpret numeric or logical variables as factors.) The output is a list showing the uniqueness of the binary relations between the factor variables (a logical matrix showing left-uniqueness in the binary relations), the mappings between factor variables, the redundant and non-redundant factor variables, and the directed acyclic graph (DAG) of these mappings (the last element requires the user to have the `ggdag` package installed; it is omitted if the package is not installed). If `plot = TRUE` the function also returns a plot of the DAG (if `ggdag` and `ggplot2` packages are installed).

Note that the function also allows the user to examine mappings between all variables in the data-frame (i.e., not just the factor variables) by setting `all.vars = TRUE`. The output from this analysis should be interpreted with caution; one-to-one mappings between non-factor variables are common (e.g., when two variables are continuous it is almost certain that they will be in a one-to-one mapping), and so the existence of a mapping may not be indicative of variable redundancy.

Note on operation: If `na.rm = FALSE` then the function analyses the mappings between the factors/variables without removing NA values. In this case an `NA` value is treated as a missing value that could be any outcome. Consequently, for purposes of determining whether there is a mapping between the variables, an `NA` value is treated as if it were every possible value. The mapping is falsified if there are at least two identical values in the domain (which may include one or more `NA` values) that map to different values in the codomain (which may include one or more `NA` values).

Value

A list object of class 'mappings' giving information on the mappings between the variables

Examples

```
DATA <- data.frame(
  VAR1 = c(0,1,2,2,0,1,2,0,0,1),
  VAR2 = c('A','B','B','B','A','B','B','A','A','B'),
  VAR3 = 1:10,
  VAR4 = c('A','B','C','D','A','B','D','A','A','B'),
  VAR5 = c(1:5,1:5)
)

# Apply mappings
mappings(DATA, all.vars = TRUE, plot = FALSE)
```

`moments` *Sample Moments*

Description

`moments` returns the sample moments of a data vector/matrix

Usage

```
moments(
  x,
  skew.type = NULL,
  kurt.type = NULL,
  kurt.excess = FALSE,
  na.rm = TRUE,
  include.sd = FALSE
)
```

Arguments

`x`	A data vector/matrix/list
`skew.type`	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
`kurt.type`	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
`kurt.excess`	Logical value; if `TRUE` the function gives the excess kurtosis (instead of raw kurtosis)
`na.rm`	Logical value; if `TRUE` the function removes `NA` values
`include.sd`	Logical value; if `TRUE` the output includes a column for the sample standard deviation (if needed)

Details

This function computes the sample moments for a data vector, matrix or list (sample mean, sample variance, sample skewness and sample kurtosis). For a vector input the function returns a single value for each sample moment of the data. For a matrix or list input the function treats each column/element as a data vector and returns a matrix of values for the sample moments of each of these datasets. The function can compute different types of skewness and kurtosis statistics using the `skew.type`, `kurt.type` and `kurt.excess` inputs. (For details on the different types of skewness and kurtosis statistics, see Joanes and Gill 1998.)

Value

A data frame containing the sample moments of the data vector/matrix

Examples

```
#Create some subgroups of mock data and a pooled dataset
set.seed(1)
N    <- c(28, 44, 51)
SUB1 <- rnorm(N[1])
SUB2 <- rnorm(N[2])
SUB3 <- rnorm(N[3])
DATA <- list(Subgroup1 = SUB1, Subgroup2 = SUB2, Subgroup3 = SUB3)
POOL <- c(SUB1, SUB2, SUB3)

#Compute sample moments for subgroups and pooled data
MOMENTS <- moments(DATA)
POOLMOM <- moments(POOL)

#Compute pooled moments via sample decomposition
sample.decomp(moments = MOMENTS)
```

nlm.prob — *Nonlinear minimisation/maximisation allowing probability vectors as inputs*

Description

`nlm.prob` minimises/maximises a function allowing probability vectors as inputs

Usage

```
nlm.prob(
  f,
  p,
  prob.vectors = list(1:length(p)),
  ...,
  lambda = 1,
  eta0max = 1e+10,
  maximise = FALSE,
  maximize = maximise,
  hessian = FALSE,
  typsize = rep(1, length(p)),
  fscale = 1,
  print.level = 0,
  ndigit = 12,
  gradtol = 1e-06,
  stepmax = max(1000 * sqrt(sum((p/typsize)^2)), 1000),
  steptol = 1e-06,
  iterlim = 100,
  check.analyticals = TRUE
)
```

Arguments

`f`	The objective function to be minimised; output should be a single numeric value.
`p`	Starting argument values for the minimisation.
`prob.vectors`	A list specifying which sets of elements are constrained to be a probability vector (each element in the list should be a vector specifying indices in the argument vector; elements cannot overlap into multiple probability vectors).
`...`	Additional arguments to be passed to `f` via `nlm`
`lambda`	The tuning parameter used in the softmax transformation for the optimisation (a single positive numeric value).
`eta0max`	The maximum absolute value for the elements of eta0 (the starting value in the unconstrained optimisation problem).
`maximise, maximize`	Logical value; if `TRUE` the function maximises the objective function instead of mimimising.

`hessian`	Logical; if `TRUE` then the output of the function includes the Hessian of `f` at the minimising point.
`typsize`	An estimate of the size of each parameter at the minimum.
`fscale`	An estimate of the size of `f` at the minimum.
`print.level`	This argument determines the level of printing which is done during the minimisation process. The default value of `0` means that no printing occurs, a value of `1` means that initial and final details are printed and a value of `2` means that full tracing information is printed.
`ndigit`	The number of significant digits in the function `f`.
`gradtol`	A positive scalar giving the tolerance at which the scaled gradient is considered close enough to zero to terminate the algorithm. The scaled gradient is a measure of the relative change in `f` in each direction `p[i]` divided by the relative change in `p[i]`.
`stepmax`	A positive scalar which gives the maximum allowable scaled step length. `stepmax` is used to prevent steps which would cause the optimisation function to overflow, to prevent the algorithm from leaving the area of interest in parameter space, or to detect divergence in the algorithm. `stepmax` would be chosen small enough to prevent the first two of these occurrences, but should be larger than any anticipated reasonable step.
`steptol`	A positive scalar providing the minimum allowable relative step length.
`iterlim`	A positive integer specifying the maximum number of iterations to be performed before the routine is terminated.
`check.analyticals`	Logical; if `TRUE` then the analytic gradients and Hessians (if supplied) are checked against numerical derivatives at the initial parameter values. This can help detect incorrectly formulated gradients or Hessians.

Details

This is a variation of the `stats::nlm` function for nonlinear minimisation. The present function is designed to minimise an objective function with one or more arguments that are probability vectors. (The objective function may also have other arguments that are not probability vectors.) The function uses the same inputs as the `stats::nlm` function, except that the user can use the input `prob.vectors` to specify which inputs are constrained to be probability vectors. This input is a list where each element in the list specifies a set of indices for the argument of the objective function; the specified set of indices is constrained to be a probability vector (i.e., each corresponding argument is non-negative and the set of these arguments must sum to one). The input `prob.vectors` may list one or more probability vectors, but they must use disjoint elements of the argument (i.e., a variable in the argument cannot appear in more than one probability vector).

Optimisation is performed by first converting the objective function into unconstrained form using the softmax transformation and its inverse to convert from unconstrained space to probability space and back. Optimisation is done on the unconstrained objective function and the results are converted back to probability space to solve the constrained optimisation problem. For purposes of conversion, this function allows specification of a tuning parameter `lambda` for the softmax and inverse- softmax transformations. (This input can either be a single tuning value used for all conversions, or a vector of values for the respective probability vectors; if the latter, there must be one value for each element of the `prob.vector` input.)

Most of the input descriptions below are adapted from the corresponding descriptions in `stat::nlm`, since our function is a wrapper to that function. The additional inputs for this function are `prob.vectors`, `lambda` and `eta0max`. The function also adds an option `maximise` to conduct maximisation instead of minimisation.

Value

A list showing the computed minimising point and minimum of `f` and other related information.

Examples

```
x <- rbinom(100, 1, .2)
nlm.prob(function(p) sum(dbinom(x,1,p[2],log=TRUE)), c(.5, .5), maximise = TRUE)
```

`PFD` *Prime Factor Decomposition (PFD)*

Description

`PFD` converts a positive integer to its prime-factor decomposition or *vice versa*

Usage

```
PFD(x)

## S3 method for class 'prime.factor.decomposition'
print(x, quote = FALSE, ...)
```

Arguments

`x`	An input vector/matrix/array (can be a vector of integers/bigz or PFDs)
`quote`	logical, indicating whether or not strings should be printed with surrounding quotes.
`...`	further arguments passed to or from other methods.

Details

This function converts a vector of integers to a corresponding character vector giving the prime-factor decomposition in a condensed form. The input can be a vector of integers or a 'bigz' vector containing large integers. In either case the function returns the corresponding vector of the prime-factor decomposition (PFD) values, written in a condensed character form. The function also converts back from the PFD form to an integer/bigz vector.

This function depends on the gmp package.

Value

If the input is integer/bigz then the output is the PFD; if the input is PFD then the output is integer/bigz

Examples

```
PFD(1:10)
stopifnot(all.equal(1:100, PFD(PFD(1:100))))
```

`plot.data.mappings` *Plot components from data mapping*

Description

This needs `ggplot2` and ggdag to function correctly.

Usage

```
## S3 method for class 'data.mappings'
plot(x, node.size = 1, text.size = 1, line.width = 1, ...)
```

Arguments

`x`	a data mapping
`node.size`	node size
`text.size`	label size for a node
`line.width`	line width
`...`	not used

Value

nothing

queue — *Generate queuing information from arrival and use times*

Description

queue returns queuing information for users and service facilities.

Usage

```
queue(
  n,
  arrive,
  use.full,
  wait.max = NULL,
  revive = 0,
  close.arrive = Inf,
  close.service = Inf,
  close.full = Inf
)

## S3 method for class 'queue'
print(x, ...)

## S3 method for class 'queue'
plot(
  x,
  print = TRUE,
  gap = NULL,
  line.width = 2,
  line.colors = NULL,
  line.colours = line.colors,
  ...
)

## S3 method for class 'queue'
summary(object, probs = NULL, probs.decimal.places = 2, ...)

## S3 method for class 'summary.queue'
print(x, ...)

## S3 method for class 'summary.queue'
plot(
  x,
  print = TRUE,
  count = FALSE,
  bar.colors = NULL,
  bar.colours = bar.colors,
    ...
  )
```

Arguments

`n`	Number of service facilities at the amenity (positive integer)
`arrive`	Vector of arrival-times for the users (non-negative numeric values)
`use.full`	Vector of (intended) use-times for the users (non-negative numeric values)
`wait.max`	Vector of maximum-waiting-times for the users (non-negative numeric values)
`revive`	Revival-time for service facilities
`close.arrive`	Closure-time for new arrivals (no new arrivals allowed)
`close.service`	Closure-time for new services (no new services allowed)
`close.full`	Closure-time for all services (all existing services are terminated)
`x, object`	a `queue` object
`...`	further arguments passed to or from other methods.
`print, gap, line.width, line.colors, line.colours`	plotting paramaters
`probs`	summary quantiles to be included in output.
`probs.decimal.places`	rounds the output to specified number of decimal places.
`count`	absolute or relative frequencies
`bar.colors, bar.colours`	plotting parameters

Details

This function computes takes inputs giving the arrival times and (intended) use times for a set of users at an amenity, plus the number of service facilities at the amenity. The function computes full information on the use of the facilities by the users, including their waiting time, actual use time, leaving time, and the facility that was used by each user.

In addition to the required inputs, the function also accepts inputs for a maximum-waiting time for each user; if the user waits up to this time then the user will leave without service. The user can also impose closure times on new arrivals, new services, or termination of services.

Note: Service facilities are assumed to be allocated to users on a "first-come, first-served" basis; in the event that more than one service facility is available for a user then the user is allocated to facilities first-to-last based on the facility number (i.e., the allocation favours the earlier facilities and it is not exchangeable with respect to the facility number).

Value

If all inputs are correctly specified then the function will return a list of class `queue` containing queuing information for the users and service facilities

Examples

```
q <- queue(2, 4:6, 7:9)
summary(q)
plot(q)
plot(summary(q))
```

`rm.attr` *Remove (non-protected) attributes from an object*

Description

`rm.attr` removes (non-protected) attributes from an object

Usage

```
rm.attr(
  object,
  list.levels = Inf,
  protected = c("class", "dim", "names", "dimnames", "rownames", "colnames")
)
```

Arguments

`object`	An object to operate on attributes from the object
`list.levels`	A non-negative integer specifying the number of levels of lists to apply the removal to
`protected`	A character vector containing the names of protected attributes (not to be removed)

Details

This function removes non-protected attributes from an `R` object. If the object is a list then the function will remove attributes within elements of the list down to the level specified by the `list.levels` input. (By default the function removes attributes from all levels of lists.) If you do not want to remove attributes from elements of a list (but still remove attributes from the outer level) you can set `list.levels = 0` to do this..

Value

The object is returned with non-protected attributes removed

Examples

```
a <- structure(list(structure(1, x=2, names=3),
              list(0, structure(3, x=4, names=5))),
              x=3, names = 4)
str(rm.attr(a, 1))
```

`sample.all` *All Sampling Variations/Permutations*

Description

`sample.all` returns a matrix of all sampling variations/permutations from a set of integers

Usage

```
sample.all(n, size = n, replace = FALSE, prob = NULL)
```

Arguments

`n`	Number of integers to sample from
`size`	Length of the sample vectors
`replace`	Logical value; if `FALSE` the sampling is without replacement; if `TRUE` the sampling is with replacement
`prob`	Probability vector giving the sampling probability for each element (must be a probability vector with length `n`)

Details

This function computes all sample vectors of size `size` composed of the elements `1, ..., n`, either with or without replacement of elements. If `size` = `n` and `replace` = `TRUE` then the list of all sample vectors corresponds to a list of all permutations of the integers `1, ..., n`.

Value

A matrix of all permutations of the elements `1, ..., n` (rows of the matrix give the permutations)

Examples

```
sample.all(n = 4, replace = FALSE)
```

`sample.decomp` *Sample decomposition*

Description

`sample.decomp` returns the data-frame of sample statistics for sample groups and their pooled sample

Usage

```
sample.decomp(
  moments = NULL,
  n = NULL,
  sample.mean = NULL,
  sample.sd = NULL,
  sample.var = NULL,
  sample.skew = NULL,
  sample.kurt = NULL,
  names = NULL,
  pooled = NULL,
  skew.type = NULL,
  kurt.type = NULL,
  kurt.excess = NULL,
  include.sd = FALSE
)
```

Arguments

`moments`	A data-frame of moments (an object of class 'moments')
`n`	A vector of sample sizes
`sample.mean`	A vector of sample means
`sample.sd`	A vector of sample standard deviations
`sample.var`	A vector of sample variances
`sample.skew`	A vector of sample skewness
`sample.kurt`	A vector of sample kurotsis
`names`	A vector of names for the sample groups
`pooled`	The number of the pooled group (if the pooled group is already present)
`skew.type`	The type of skewness statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
`kurt.type`	The type of kurtosis statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
`kurt.excess`	Logical value; if `TRUE` the sample kurtosis is the excess kurtosis (instead of the raw kurtosis)
`include.sd`	Logical value; if `TRUE` the output includes a column for the sample standard deviation (if needed)

Details

It is often useful to take a set of sample groups with known sample statistics and aggregate these into a single pooled sample and find the sample statistics of the pooled sample. Likewise, it is sometimes useful to take a set of sample groups and a pooled group with known sample statistics and determine the statistics of the other group required to complete the pooled sample. Both of these tasks can be accomplished using decomposition formulae for the sample size, sample mean and sample variance (or sample standard deviation). This function implements either of these two decomposition methods to find the sample statistics of the pooled sample or the other group remaining to obtain the pooled sample. The user inputs vectors for the sample size, sample mean and sample variance (or sample standard deviation). By default the groups are taken to be separate groups and the function computes the sample statistics for the pooled sample However, the user can input the number pooled sample as the input `pooled`; in this case that group is treated as the pooled sample and the function computes the other sample group required to obtain this pooled sample. The function returns a data-frame showing the sample statistics for all the groups including the pooled sample.

Value

A data-frame of all groups showing their sample sizes and sample moments

See Also

`moments`

`skewness` *Sample Skewness*

Description

`skewness` returns the sample skewness of a data vector/matrix

Usage

```
skewness(x, skew.type = NULL, na.rm = FALSE)
```

Arguments

`x`	A data vector/matrix
`skew.type`	The type of skewness statistic used ('Moment', 'Fisher Pearson' or 'Adjusted Fisher Pearson')
`na.rm`	Logical value; if `TRUE` the function removes `NA` values

Details

This function computes the sample skewness for a data vector or matrix. For a vector input the function returns a single value for the sample skewness of the data. For a matrix input the function treats each column as a data vector and returns a vector of values for the sample skewness of each of these datasets. The function can compute different types of skewness statistics using the `skew.type` input.

Value

The sample skewness of the data vector/matrix

Examples

```
skewness(rnorm(1000))
skewness(rexp(1000))
```

`softmax` *Softmax and logsoftmax functions and their inverse functions*

Description

`softmax` returns the value of the softmax function `softmaxinv` returns the value of the inverse-softmax function `logsoftmax` returns the value of the logsoftmax function `logsoftmaxinv` returns the value of the inverse-logsoftmax function

Usage

```
softmax(eta, lambda = 1, gradient = FALSE, hessian = FALSE)

softmaxinv(p, lambda = 1, gradient = FALSE, hessian = FALSE)

logsoftmax(eta, lambda = 1, gradient = FALSE, hessian = FALSE)

logsoftmaxinv(l, lambda = 1, gradient = FALSE, hessian = FALSE)
```

Arguments

`eta`	A numeric vector input
`lambda`	Tuning parameter (a single positive value)
`gradient`	Logical
`hessian`	Logical
`p`	A probability vector (i.e., numeric vector of non-negative values that sum to one)
`l`	A log-probability vector (i.e., numeric vector of non-positive values that logsum to zero)

Details

The softmax function is a bijective function that maps a real vector with length `m-1` to a probability vector with length `m` with all non-zero probabilities. The softmax function is useful in a wide range of probability and statistical applications. The present functions define the softmax function and its inverse, both with a tuning parameter. It also defines the log-softmax function and its inverse, both with a tuning parameter.

Value

Value of the softmax function or its inverse (or their log). If `gradient` or `hessian` is `TRUE`, it will be included as an attribute.

Examples

```
softmax(5:7)
softmaxinv(softmax(5:7))
logsoftmax(5:7)
logsoftmaxinv(logsoftmax(5:7))
```

`tailplot` *Generate tail plots for a data vector*

Description

`tailplot` generates the tail plot and Hill plot for the input data

Usage

```
tailplot(x, tail.prop = 0.05, left = TRUE, right = TRUE,
       show.lines = TRUE, lines = 16, line.order = 3,
       tail.plot = TRUE, hill.plot = FALSE, dsm.plot = FALSE,
       point.size = 3, point.alpha = 0.4,
       point.color = NULL, point.colour = point.color,
       line.color  = NULL, line.colour  = line.color,
       ytop.mult.left = 3, ytop.mult.right = 3)
```

Arguments

`x`	Data vector (numeric)
`tail.prop`	The proportion of values to use in each tail
`left`	Logical; if `TRUE` the tail plot includes a plot for the left tail
`right`	Logical; if `TRUE` the tail plot includes a plot for the upper tail
`show.lines`	Logical; if `TRUE` the plots include lines for fixed logarithmic decay
`lines`	Number of lines to include in the plot (included if `show.lines` is `TRUE`)
`line.order`	Order of the lines in the plot (included if `show.lines` is `TRUE`)
`tail.plot`	Logical; if `TRUE` the output includes the tail plot
`hill.plot`	Logical; if `TRUE` the output includes the Hill plot
`dsm.plot`	Logical; if `TRUE` the output includes the DSM plot
`point.size`	Size of the points in the plots
`point.alpha`	Alpha-transparency of the points in the plots
`point.color`, `point.colour`	Colour of the points in the plots (default is blue)

`line.color, line.colour`	Colour of the lines in the plots (default is darkred)
`ytop.mult.left`	Multiplier used to determine the height of axis for left Hill/DSM plots
`ytop.mult.right`	Multiplier used to determine the height of axis for right Hill/DSM plots

Details

The tail plot for a dataset shows the rate of decay of the tails in the data. It is used to diagnose whether certain moments of the underlying distribution exist (e.g., the variance), which is used in turn to determine whether certain statistical laws apply to the distribution (e.g., the central limit theorem). The Hill plot shows the adjusted Hill estimator for the tail index of the distribution. The DSM (De-Sousa-Michailidis) plot shows the adjusted DSM estimator for the tail index of the distribution. These latter estimators are very similar; more details are found in the reference below. Our adjusted versions of these estimators are computed using data for the left and right deviations from the extreme values of the dataset; this is done to ensure that the estimators are shift-invariant (the standard Hill and DSM estimators are not).

The present function produces tail plots and Hill/DSM plots for the input data vector to show the rate of decay in the tails. By default, both the tails are shown, but the user can show the plots only for one tail if preferred. The user can turn any of the plots on or off using the inputs to the function.

By default, the tail plot includes lines showing cubic decay in the tails — if the tails of the distribution decay faster than cubic decay then the variance of the distribution exists. The user can change the order of the lines in the plot to show other decay orders; this can be used to diagnose the existence of moments of other orders.

De Sousa, B. and Michailidis, G. (2004) A Diagnostic Plot for Estimating the Tail Index of a Distribution. Journal of Computational and Graphical Statistics 13(4), pp. 1-22.

Value

Tail plots for the input data (and Hill plots or DSM plots if requested)

Examples

```
try(tailplot(rnorm(500)))
```

`zipfplot` *Generate Zipf plot*

Description

`zipfplot` generates the Zipf plot for the input data

Usage

```
zipfplot(
  x,
  relative.freq = TRUE,
  smooth.line = TRUE,
  smooth.conf = TRUE,
  conf.level = 0.99,
  separate.plots = FALSE,
  data.name = FALSE,
  point.size = 3,
  point.alpha = 0.4
)
```

Arguments

`x`	Data vector, matrix or data-frame
`relative.freq`	Logical; if `TRUE` the plot shows the relative frequency on vertical axis
`smooth.line`	Logical; if `TRUE` the plot shows a smoothed line through the data using LOESS method
`smooth.conf`	Logical; if `TRUE` the plot shows confidence bands on the smoothed line (only shown if smoothed line is shown)
`conf.level`	The confidence level for the confidence bands on the smoothed line
`separate.plots`	Logical; if `TRUE` the plot shows
`data.name`	Logical; if `TRUE` the subtitle will state the name of the input data
`point.size`	Size of the points in the plot
`point.alpha`	Alpha-transparency of the points in the plot

Details

The Zipf plot for a dataset shows the ranks of outcomes versus their frequency on a log-log scale. It is used to determine how closely a dataset follows "Zipf's law". The present function takes in a vector of values and produces the Zipf plot. The data input can be either a vector, a matrix or a data frame. If the input data is a vector then the output will be a Zipf plot for that data vector. If the input data is a matrix or data frame then each column will be treated as a separate variable and the output will be a single Zipf plot showing each of the variables. The user can control whether the variables are shown on a single plot or separate plots.

Value

Zipf plot for the input data

Examples

```
try(zipfplot(sample(LETTERS, 300, replace = TRUE)))
```

Index