Package 'doBy'

June 21, 2024

Version 4.6.22

Title Groupwise Statistics, LSmeans, Linear Estimates, Utilities

Author Søren Højsgaard <sorenh@math.aau.dk> and Ulrich Halekoh <uhalekoh@health.sdu.dk>

Maintainer Søren Højsgaard <sorenh@math.aau.dk>

Description Utility package containing:
1) Facilities for working with grouped data: 'do' something to data stratified 'by' some variables.
2) LSmeans (least-squares means), general linear estimates.
3) Restrict functions to a smaller domain.
4) Miscellaneous other utilities.

Encoding UTF-8

VignetteBuilder knitr

LazyData true

LazyDataCompression xz

License GPL (>= 2)

Depends R (>= 4.2.0), methods

Imports boot, broom, cowplot, Deriv, dplyr, ggplot2, MASS, Matrix, modelr, microbenchmark, rlang, tibble, tidyr,

Suggests geepack, knitr, lme4, markdown, multcomp, pbkrtest (>= 0.4-8.1), survival, testthat (>= 2.1.0)

RoxygenNote 7.3.1

NeedsCompilation no

Repository CRAN

Date/Publication 2024-06-21 08:10:05 UTC

Contents

`.rhsf2list` *Convert right hand sided formula to a list*

Description

Convert right hand sided formula to a list

Usage

```
.rhsf2list(f)
```

Arguments

`f`	A right hand sided formula

beets *beets data*

Description

Yield and sugar percentage in sugar beets from a split plot experiment. Data is obtained from a split plot experiment. There are 3 blocks and in each of these the harvest time defines the "whole plot" and the sowing time defines the "split plot". Each plot was 25 square meters and the yield is recorded in kg. See 'details' for the experimental layout.

Usage

```
beets
```

Format

The format is: chr "beets"

Details

```
Experimental plan
Sowing times            1        4. april
                        2       12. april
                        3       21. april
                        4       29. april
                        5       18. may
Harvest times           1        2. october
                        2       21. october
Plot allocation:
              Block 1     Block 2     Block 3
            +-----------|-----------|-----------+
      Plot  | 1 1 1 1 1 | 2 2 2 2 2 | 1 1 1 1 1 | Harvest time
       1-15 | 3 4 5 2 1 | 3 2 4 5 1 | 5 2 3 4 1 | Sowing time
            |-----------|-----------|-----------|
      Plot  | 2 2 2 2 2 | 1 1 1 1 1 | 2 2 2 2 2 | Harvest time
      16-30 | 2 1 5 4 3 | 4 1 3 2 5 | 1 4 3 2 5 | Sowing time
            +-----------|-----------|-----------+
```

References

Ulrich Halekoh, Søren Højsgaard (2014)., A Kenward-Roger Approximation and Parametric Bootstrap Methods for Tests in Linear Mixed Models - The R Package pbkrtest., Journal of Statistical Software, 58(10), 1-30., https://www.jstatsoft.org/v59/i09/

Examples

```
data(beets)

beets$bh <- with(beets, interaction(block, harvest))
summary(aov(yield ~ block + sow + harvest + Error(bh), beets))
summary(aov(sugpct ~ block + sow + harvest + Error(bh), beets))
```

---

binomial_to_bernoulli_data

Convert binomial data to bernoulli data

Description

Convert binomial data to bernoulli data by expanding dataset.

Usage

```
binomial_to_bernoulli_data(
  data.,
  y,
  size,
  type = c("rest", "total"),
  response_name = "response",
  rest_name = NULL
)
```

Arguments

data.	A dataframe
y	Column with 'successes' in binomial distribution y~bin(size, p)
size	Column with 'failures', i.e. size-y or 'total', i.e. size.
type	Whether size is rest (i.e. 'failures') or 'total'
response_name	Name of response variable in output dataset.
rest_name	Name of 'failures' in column response_name.

Examples

```
dat <- budworm
dat <- dat[dat$dose %in% c(1,2), ]
dat$ntotal <- 5
dat
dat.a <- dat |>
  binomial_to_bernoulli_data(ndead, ntotal, type="total")
dat.b <- dat |>
  dplyr::mutate(nalive=ntotal-ndead) |> dplyr::select(-ntotal) |>
  binomial_to_bernoulli_data(ndead, nalive, type="rest")
```

```
m0 <- glm(cbind(ndead, ntotal-ndead) ~ dose + sex, data=dat, family=binomial())
m1 <- glm(ndead / ntotal ~ dose + sex, data=dat, weight=ntotal, family=binomial())
ma <- glm(response ~ dose + sex, data=dat.a, family=binomial())
mb <- glm(response ~ dose + sex, data=dat.b, family=binomial())

dat.a$response
dat.b$response ## Not same and therefore the following do not match

all.equal(coef(m0), coef(ma))
all.equal(coef(m0), coef(mb))
all.equal(coef(m1), coef(ma))
all.equal(coef(m1), coef(mb))
```

`bquote_fun_list` *Backquote a list of functions*

Description

Backquote a list of functions

Usage

```
bquote_fun_list(fun_list)
```

Arguments

`fun_list`	List of functions

See Also

`base::bquote()`, `set_default()`, `section_fun()`

Examples

```
## Evaluate a list of functions
f1 <- function(x){x + 1}
f2 <- function(x){x + 8}

f1_ <- set_default(f1, list(x=10))
f2_ <- set_default(f2, list(x=10))

f1_(); f2_()

fn_list  <- list(f1_, f2_)
fn_list_ <- bquote_fun_list(fn_list)

eval(fn_list[[1]])     ## No
sapply(fn_list, eval)  ## No
```

```
eval(fn_list_[[1]])    ## Yes
sapply(fn_list_, eval) ## Yes
```

---

by-lapply *Formula based version of lapply and sapply*

Description

This function is a wrapper for calling lapply on the list resulting from first calling splitBy.

Usage

```
lapply_by(data, formula, FUN, ...)

lapplyBy(formula, data = parent.frame(), FUN, ...)

sapply_by(data, formula, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

sapplyBy(
  formula,
  data = parent.frame(),
  FUN,
  ...,
  simplify = TRUE,
  USE.NAMES = TRUE
)
```

Arguments

data	A dataframe.
formula	A formula describing how data should be split.
FUN	A function to be applied to each element in the split list, see 'Examples' below.
...	optional arguments to FUN.
simplify	Same as for sapply
USE.NAMES	Same as for sapply

Value

A list.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

splitBy, split_by

Examples

```
fun <- function(x) range(x$uptake)
lapplyBy(~Treatment + Type, data=CO2, FUN=fun)
sapplyBy(~Treatment + Type, data=CO2, FUN=fun)

# Same as
lapply(splitBy(~Treatment + Type, data=CO2), FUN=fun)
```

---

by-lmby *List of lm objects with a common model*

Description

The data is split into strata according to the levels of the grouping factors and individual lm fits are obtained for each stratum.

Usage

```
lm_by(data, formula, id = NULL, ...)

lmBy(formula, data, id = NULL, ...)
```

Arguments

data	A dataframe
formula	A linear model formula object of the form y ~ x1 + ... + xn | g1 + ... + gm. In the formula object, y represents the response, x1, ..., xn the covariates, and the grouping factors specifying the partitioning of the data according to which different lm fits should be performed.
id	A formula describing variables from data which are to be available also in the output.
...	Additional arguments passed on to lm().

Value

A list of lm fits.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
bb <- lmBy(1 / uptake ~ log(conc) | Treatment, data=CO2)
coef(bb)

fitted(bb)
residuals(bb)

summary(bb)
coef(summary(bb))
coef(summary(bb), simplify=TRUE)
```

---

by-order *Ordering (sorting) rows of a data frame*

Description

Ordering (sorting) rows of a data frame by the certain variables in the data frame. This function is essentially a wrapper for the `order()` function - the important difference being that variables to order by can be given by a model formula.

Usage

```
order_by(data, formula)

orderBy(formula, data)
```

Arguments

`data`	A dataframe
`formula`	The right hand side of a formula

Details

The sign of the terms in the formula determines whether sorting should be ascending or decreasing; see examples below

Value

The ordered data frame

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk> and Kevin Wright

See Also

`transformBy`, `transform_by`, `splitBy`, `split_by`

Examples

```
orderBy(~ conc + Treatment, CO2)
## Sort decreasingly by conc
orderBy(~ - conc + Treatment, CO2)
```

---

by-sample *Sampling from a data frame*

Description

A data frame is split according to some variables in a formula, and a sample of a certain fraction of each is drawn.

Usage

```
sample_by(data, formula, frac = 0.1, replace = FALSE, systematic = FALSE)

sampleBy(
  formula,
  frac = 0.1,
  replace = FALSE,
  data = parent.frame(),
  systematic = FALSE
)
```

Arguments

data	A data frame.
formula	A formula defining the grouping of the data frame.
frac	The part of data to be sampled.
replace	Is the sampling with replacement.
systematic	Should sampling be systematic.

Details

If systematic=FALSE (default) then frac gives the fraction of data sampled. If systematic=TRUE and frac=.2 then every 1/.2 i.e. every 5th observation is taken out.

Value

A dataframe.

See Also

orderBy, order_by, splitBy, split_by, summaryBy, summary_by, transformBy, transform_by

Examples

```
data(dietox)
sampleBy(formula = ~ Evit + Cu, frac=.1, data = dietox)
```

---

by-split *Split a data frame*

Description

Split a dataframe according to the levels of variables in the dataframe. The variables to split by can be given as a formula or as a character vector.

Usage

```
split_by(data, formula, drop = TRUE)

splitBy(formula, data = parent.frame(), drop = TRUE)

## S3 method for class 'splitByData'
head(x, n = 6L, ...)

## S3 method for class 'splitByData'
tail(x, n = 6L, ...)
```

Arguments

data	A data frame
formula	Variables to split data frame by, as as.quoted variables, a formula or character vector.
drop	Logical indicating if levels that do not occur should be dropped. Deprecated; levels that do not occur are ignored.
x	An object.
n	A single integer. If positive or zero, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the "n" last/first number of elements of "x".
...	Arguments to be passed to or from other methods.

Value

A list of dataframes.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

orderBy, order_by, summaryBy, summary_by, transformBy, transform_by

Examples

```
data(dietox, package="doBy")
splitBy(formula = ~Evit + Cu, data = dietox)
splitBy(formula = c("Evit", "Cu"), data = dietox)

splitBy(~Treatment + Type, data=CO2)
splitBy(c("Treatment", "Type"), data=CO2)

x <- splitBy(~Treatment, data=CO2)
head(x)
tail(x)
```

---

by-subset *Finds subsets of a dataframe which is split by variables in a formula.*

Description

A data frame is split by a formula into groups. Then subsets are found within each group, and the result is collected into a data frame.

Usage

```
subset_by(data, formula, subset, select, drop = FALSE, join = TRUE, ...)

subsetBy(
  formula,
  subset,
  data = parent.frame(),
  select,
  drop = FALSE,
  join = TRUE,
  ...
)
```

Arguments

data	A data frame.
formula	A right hand sided formula or a character vector of variables to split by.
subset	logical expression indicating elements or rows to keep: missing values are taken as false.
select	expression, indicating columns to select from a data frame.
drop	passed on to [indexing operator.

join	If FALSE the result is a list of data frames (as defined by 'formula'); if TRUE one data frame is returned.
...	further arguments to be passed to or from other methods.

Value

A data frame.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

splitBy, split_by

Examples

```
data(dietox)
subsetBy(~Evit, Weight < mean(Weight), data=dietox)
```

---

by-summary *Function to calculate groupwise summary statistics*

Description

Function to calculate groupwise summary statistics, much like the summary procedure of SAS

Usage

```
summary_by(
  data,
  formula,
  id = NULL,
  FUN = mean,
  keep.names = FALSE,
  p2d = FALSE,
  order = TRUE,
  full.dimension = FALSE,
  var.names = NULL,
  fun.names = NULL,
  ...
)

summaryBy(
  formula,
  data = parent.frame(),
  id = NULL,
```

```
  FUN = mean,
  keep.names = FALSE,
  p2d = FALSE,
  order = TRUE,
  full.dimension = FALSE,
  var.names = NULL,
  fun.names = NULL,
  ...
)
```

Arguments

data	A data frame.
formula	A formula object, see examples below.
id	A formula specifying variables which data are not grouped by but which should appear in the output. See examples below.
FUN	A list of functions to be applied, see examples below.
keep.names	If TRUE and if there is only ONE function in FUN, then the variables in the output will have the same name as the variables in the input, see 'examples'.
p2d	Should parentheses in output variable names be replaced by dots?
order	Should the resulting dataframe be ordered according to the variables on the right hand side of the formula? (using orderBy
full.dimension	If TRUE then rows of summary statistics are repeated such that the result will have the same number of rows as the input dataset.
var.names	Option for user to specify the names of the variables on the left hand side.
fun.names	Option for user to specify function names to apply to the variables on the left hand side.
...	Additional arguments to FUN. This could for example be NA actions.

Details

Extra arguments (...) are passed onto the functions in FUN. Hence care must be taken that all functions in FUN accept these arguments - OR one can explicitly write a functions which get around this. This can particularly be an issue in connection with handling NAs. See examples below. Some code for this function has been suggested by Jim Robison-Cox. Thanks.

Value

A dataframe.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ave, descStat, orderBy, order_by, splitBy, split_by, transformBy, transform_by

Examples

```
data(dietox)
dietox12    <- subset(dietox,Time==12)

fun <- function(x){
  c(m=mean(x), v=var(x), n=length(x))
}

summaryBy(cbind(Weight, Feed) ~ Evit + Cu, data=dietox12,
          FUN=fun)

summaryBy(list(c("Weight", "Feed"), c("Evit", "Cu")), data=dietox12,
          FUN=fun)

## Computations on several variables is done using cbind( )
summaryBy(cbind(Weight, Feed) ~ Evit + Cu, data=subset(dietox, Time > 1),
   FUN=fun)

## Calculations on transformed data is possible using cbind( ), but
# the transformed variables must be named

summaryBy(cbind(lw=log(Weight), Feed) ~ Evit + Cu, data=dietox12, FUN=mean)

## There are missing values in the 'airquality' data, so we remove these
## before calculating mean and variance with 'na.rm=TRUE'. However the
## length function does not accept any such argument. Hence we get
## around this by defining our own summary function in which length is
## not supplied with this argument while mean and var are:

sumfun <- function(x, ...){
  c(m=mean(x, na.rm=TRUE, ...), v=var(x, na.rm=TRUE, ...), l=length(x))
}
summaryBy(cbind(Ozone, Solar.R) ~ Month, data=airquality, FUN=sumfun)
## Compare with
aggregate(cbind(Ozone, Solar.R) ~ Month, data=airquality, FUN=sumfun)

## Using '.' on the right hand side of a formula means to stratify by
## all variables not used elsewhere:

data(warpbreaks)
summaryBy(breaks ~ wool + tension, warpbreaks, FUN=mean)
summaryBy(breaks ~ ., warpbreaks, FUN=mean)
summaryBy(. ~ wool + tension, warpbreaks, FUN=mean)

summaryBy(. ~ wool + tension, warpbreaks, FUN=mean)
```

`by-transform` *Function to make groupwise transformations*

Description

Function to make groupwise transformations of data by applying the transform function to subsets of data.

Usage

```
transform_by(data, formula, ...)

transformBy(formula, data, ...)
```

Arguments

data	A data frame
formula	A formula with only a right hand side, see examples below
...	Further arguments of the form tag=value

Details

The ... arguments are tagged vector expressions, which are evaluated in the data frame data. The tags are matched against names(data), and for those that match, the value replace the corresponding variable in data, and the others are appended to data.

Value

The modified value of the dataframe data.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

orderBy, order_by, summaryBy, summary_by, splitBy, split_by

Examples

```
data(dietox)
transformBy(~Pig, data=dietox, minW=min(Weight), maxW=max(Weight),
    gain=diff(range(Weight)))
```

by_scale *Scale a dataframe or matrix*

Description

Split a dataframe into a list according to the levels of variables in the dataframe and scale the numeric variables in each dataframe in the list.

Usage

```
scaleBy(formula, data = parent.frame(), center = TRUE, scale = TRUE)

scale_by(data, formula, center = TRUE, scale = TRUE)
```

Arguments

formula	Variables to split data frame by, as as.quoted variables, a formula or character vector.
data	A dataframe or matrix
center	Logical, should data be centered.
scale	Logical, should data be scaled.

Value

A list of objects of same class as x

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

orderBy, order_by, summaryBy, summary_by, transformBy, transform_by

Examples

```
scaleBy(~Species, data=iris, center=TRUE, scale=FALSE)
scaleBy(~1, data=iris, center=TRUE, scale=FALSE)

scale_by(iris, ~Species)
scale_by(iris, ~1)

## Not combine list of dataframes to one dataframe e.g. as:
a <- scale_by(iris, ~Species)
d <- do.call(rbind, a)
```

carcass *Lean meat contents of 344 pig carcasses*

Description

Measurement of lean meat percentage of 344 pig carcasses together with auxiliary information collected at three Danish slaughter houses

Usage

```
carcass
```

Format

carcassall: A data frame with 344 observations on the following 17 variables.

`weight` Weight of carcass

`lengthc` Length of carcass from back toe to head (when the carcass hangs in the back legs)

`lengthf` Length of carcass from back toe to front leg (that is, to the shoulder)

`lengthp` Length of carcass from back toe to the pelvic bone

`Fat02`, `Fat03`, `Fat11`, `Fat12`, `Fat13`, `Fat14`, `Fat16` Thickness of fat layer at different locations on the back of the carcass (FatXX refers to thickness at (or rather next to) rib no. XX. Notice that 02 is closest to the head

`Meat11`, `Meat12`, `Meat13` Thickness of meat layer at different locations on the back of the carcass, see description above

`LeanMeat` Lean meat percentage determined by dissection

`slhouse` Slaughter house; a factor with levels `slh1` and `slh2`.

`sex` Sex of the pig; a factor with levels `castrate` and `female`.

`size` Size of the carcass; a factor with levels `normal` and `large`. Here, `normal` refers to carcass weight under 80 kg; `large` refers to carcass weights between 80 and 110 kg.

Details

: Notice that there were slaughtered large pigs only at one slaughter house.

Note

carcass: Contains only the variables Fat11, Fat12, Fat13, Meat11, Meat12, Meat13, LeanMeat

Source

Busk, H., Olsen, E. V., Brøndum, J. (1999) Determination of lean meat in pig carcasses with the Autofom classification system, Meat Science, 52, 307-314

Examples

```
data(carcass)
head(carcass)
```

`chr_to_matrix` *Character vector to matrix*

Description

Character vector to matrix

Usage

```
chr_to_matrix(x, value = 0)
```

Arguments

`x`	character vector
`value`	value in matrix

Details

creates square matrix with `x` as row and column names and `val` as values

Examples

```
d1 <- letters[1:3]
chr_to_matrix(d1, 3:5)
```

`codstom` *Diet of Atlantic cod in the Gulf of St. Lawrence (Canada)*

Description

Stomach content data for Atlantic cod (*Gadus morhua*) in the Gulf of St.Lawrence, Eastern Canada. Note: many prey items were of no interest for this analysis and were regrouped into the "Other" category.

Usage

```
codstom
```

Format

A data frame with 10000 observations on the following 10 variables.

`region` a factor with levels `SGSL` `NGSL` representing the southern and northern Gulf of St. Lawrence, respectively

`ship.type` a factor with levels `2` `3` `31` `34` `90` `99`

`ship.id` a factor with levels `11558` `11712` `136148` `136885` `136902` `137325` `151225` `151935` `99433`

`trip` a factor with levels `10` `11` `12` `179` `1999` `2` `2001` `20020808` `3` `4` `5` `6` `7` `8` `88` `9` `95`

`set` a numeric vector

`fish.id` a numeric vector

`fish.length` a numeric vector, length in mm

`prey.mass` a numeric vector, mass of item in stomach, in g

`prey.type` a factor with levels `Ammodytes_sp` `Argis_dent` `Chion_opil` `Detritus` `Empty` `Eualus_fab` `Eualus_mac` `Gadus_mor` `Hyas_aran` `Hyas_coar` `Lebbeus_gro` `Lebbeus_pol` `Leptocl_mac` `Mallot_vil` `Megan_norv` `Ophiuroidea` `Other` `Paguridae` `Pandal_bor` `Pandal_mon` `Pasiph_mult` `Sabin_sept` `Sebastes_sp` `Them_abys` `Them_comp` `Them_lib`

Details

Cod are collected either by contracted commerical fishing vessels (`ship.type` 90 or 99) or by research vessels. Commercial vessels are identified by a unique `ship.id`.

Either one research vessel or several commercial vessels conduct a survey (`trip`), during which a trawl, gillnets or hooked lines are set several times. Most trips are random stratified surveys (depth-based stratification).

Each trip takes place within one of the `region`s. The `trip` label is only guaranteed to be unique within a region and the `set` label is only guaranteed to be unique within a `trip`.

For each fish caught, the `fish.length` is recorded and the fish is allocated a `fish.id`, but the `fish.id` is only guaranteed to be unique within a `set`. A subset of the fish caught are selected for stomach analysis (stratified random selection according to fish length; unit of stratification is the set for research surveys, the combination ship.id and stratum for surveys conducted by commercial vessels, although strata are not shown in codstom).

The basic experimental unit in this data set is a cod stomach (one stomach per fish). Each stomach is uniquely identified by a combination of `region`, `ship.type`, `ship.id`, `trip`, `set`, and `fish.id`.

For each prey item found in a stomach, the species and mass of the prey item are recorded, so there can be multiple observations per stomach. There may also be several prey items with the same `prey.type` in the one stomach (for example many `prey.types` have been recoded `Other`, which produced many instances of `Other` in the same stomach).

If a stomach is empty, a single observation is recorded with `prey.type` `Empty` and a `prey.mass` of zero.

Source

Small subset from a larger dataset (more stomachs, more variables, more `prey.types`) collected by D. Chabot and M. Hanson, Fisheries & Oceans Canada <chabotd@dfo-mpo.gc.ca>.

Examples

```
data(codstom)
str(codstom)
# removes multiple occurences of same prey.type in stomachs
codstom1 <- summaryBy(prey.mass ~
                      region + ship.type + ship.id + trip + set + fish.id + prey.type,
                      data = codstom,
                      FUN = sum)

# keeps a single line per stomach with the total mass of stomach content
codstom2 <- summaryBy(prey.mass ~ region + ship.type + ship.id + trip + set + fish.id,
                      data = codstom,
                      FUN = sum)

# mean prey mass per stomach for each trip
codstom3 <- summaryBy(prey.mass.sum ~ region + ship.type + ship.id + trip,
                      data = codstom2, FUN = mean)

## Not run:
# wide version, one line per stomach, one column per prey type
library(reshape)
codstom4 <- melt(codstom, id = c(1:7, 9))
codstom5 <- cast(codstom4,
                 region + ship.type + ship.id + trip + set + fish.id + fish.length ~
                 prey.type, sum)
k <- length(names(codstom5))
prey_col <- 8:k
out <- codstom5[,prey_col]
out[is.na(out)] <- 0
codstom5[,prey_col] <- out
codstom5$total.content <- rowSums(codstom5[, prey_col])

## End(Not run)
```

`crickets` *crickets data*

Description

name crickets

Usage

```
crickets
```

Format

This data frame contains:

species: Species, see details

temp: temperature

pps: pulse per second

Details

Walker (1962) studied the mating songs of male tree crickets. Each wingstroke by a cricket produces a pulse of song, and females may use the number of pulses per second to identify males of the correct species. Walker (1962) wanted to know whether the chirps of the crickets Oecanthus exclamationis and Oecanthus niveus had different pulse rates. See http://www.biostathandbook.com/ancova.html for details. He measured the pulse rate of the crickets (variable pps) at a variety of temperatures (temp):

Examples

```
data(crickets)
coplot(pps ~ temp | species, data=crickets)
```

`crimeRate` *crimeRate*

Description

Crime rates per 100,000 inhabitants in states of the USA for different crime types in 1977.

Usage

```
crimeRate
```

Format

This data frame contains:

state: State of the USA

murder: crime of murder

rape:

robbery:

assault:

burglary: residential theft

larceny: unlawful taking of personal property (pocket picking)

autotheft:

Examples

```
data(crimeRate)
```

crime_rate *crimeRate*

Description

Crime rates per 100,000 inhabitants in states of the USA for different crime types in 1977.

Usage

```
crime_rate
```

Format

This data frame contains:

murder: crime of murder

rape:

robbery:

assault:

burglary: residential theft

larceny: unlawful taking of personal property (pocket picking)

autotheft:

Examples

```
data(crime_rate)
```

cropyield *Yield from Danish agricultural production of grain and root crop.*

Description

Yield from Danish agricultural production of grain and root crop.

Usage

```
cropyield
```

Format

A dataframe with 97 rows and 7 columns.

`year` From 1901 to 1997.

`precip` Milimeter precipitation.

`yield` Million feed units (see details).

`area` Area in 1000 ha for grains and root crop.

`fertil` 1000 tons fertilizer.

`avgtmp1` Average temperature April-June (3 months).

`avgtmp2` Average temperature July-Octobre (4 months).

Details

A feed unit is the amount of energy in a kg of barley.

References

Danmarks statistik (Statistics Denmark).

`cv_glm_fitlist` *Cross-validation for list of glm objects*

Description

Cross-validation for list of glm objects

Usage

```
cv_glm_fitlist(data., fit_list, K = 5)
```

Arguments

`data.`	A data frame
`fit_list`	A list of glm objects
`K`	Number of folds

`data_breastcancer` *Gene expression signatures for p53 mutation status in 250 breast cancer samples*

Description

Perturbations of the p53 pathway are associated with more aggressive and therapeutically refractory tumours. We preprocessed the data using Robust Multichip Analysis (RMA). Dataset has been truncated to the 1000 most informative genes (as selected by Wilcoxon test statistics) to simplify computation. The genes have been standardized to have zero mean and unit variance (i.e. z-scored).

Usage

```
breastcancer
```

Format

A data frame with 250 observations on 1001 variables. The first 1000 columns are numerical variables; the last column (named `code`) is a factor with levels `case` and `control`.

Details

The factor `code` defines whether there was a mutation in the p53 sequence (code=case) or not (code=control).

Source

Chris Holmes, `<c.holmes@stats.ox.ac.uk>`

References

Miller et al (2005, PubMed ID:16141321)

Examples

```
data(breastcancer)
bc <- breastcancer
pairs(bc[,1:5], col=bc$code)

train <- sample(1:nrow(bc), 50)
table(bc$code[train])
## Not run:
library(MASS)
z <- lda(code ~ ., data=bc, prior = c(1, 1) / 2, subset = train)
pc <- predict(z, bc[-train, ])$class
pc
bc[-train, "code"]
table(pc, bc[-train, "code"])
```

```
## End(Not run)
```

`data_budworm` *Budworm data*

Description

Experiment on the toxicity to the tobacco budworm Heliothis virescens of doses of the pyrethroid trans-cypermethrin to which the moths were beginning to show resistance. Batches of 20 moths of each sex were exposed for three days to the pyrethroid and the number in each batch that were dead or knocked down was recorded. Data is reported in Collett (1991, p. 75).

Usage

```
budworm
```

Format

This data frame contains 12 rows and 4 columns:

sex: sex of the budworm.

dose: dose of the insecticide trans-cypermethrin (in micro grams).

ndead: budworms killed in a trial.

ntotal: total number of budworms exposed per trial.

Source

Collett, D. (1991) Modelling Binary Data, Chapman & Hall, London, Example 3.7

References

Venables, W.N; Ripley, B.D.(1999) Modern Applied Statistics with S-Plus, Heidelberg, Springer, 3rd edition, chapter 7.2

Examples

```
data(budworm)

## function to caclulate the empirical logits
empirical.logit<- function(nevent,ntotal) {
   y <- log((nevent + 0.5) / (ntotal - nevent + 0.5))
   y
}


# plot the empirical logits against log-dose
```

```
log.dose <- log(budworm$dose)
emp.logit <- empirical.logit(budworm$ndead, budworm$ntotal)
plot(log.dose, emp.logit, type='n', xlab='log-dose',ylab='emprirical logit')
title('budworm: emprirical logits of probability to die ')
male <- budworm$sex=='male'
female <- budworm$sex=='female'
lines(log.dose[male], emp.logit[male], type='b', lty=1, col=1)
lines(log.dose[female], emp.logit[female], type='b', lty=2, col=2)
legend(0.5, 2, legend=c('male', 'female'), lty=c(1,2), col=c(1,2))

## Not run:
* SAS example;
data budworm;
infile 'budworm.txt' firstobs=2;
input sex dose ndead ntotal;
run;

## End(Not run)
```

`data_cad` *Coronary artery disease data*

Description

A cross classified table with observational data from a Danish heart clinic. The response variable is CAD (coronary artery disease, some times called heart attack).

Usage

```
data(cad1)
```

Format

A data frame with 236 observations on the following 14 variables.

`Sex` Sex; a factor with levels `Female` `Male`

`AngPec` Angina pectoris (chest pain attacks); a factor with levels `Atypical` `None` `Typical`

`AMI` Acute myocardic infarct; a factor with levels `Definite` `NotCertain`

`QWave` A reading from an electrocardiogram; a factor with levels `No` `Yes`; Yes means pathological and is a sign of previous myocardial infarction.

`QWavecode` a factor with levels `Nonusable` `Usable`. An assesment of whether QWave is reliable.

`STcode` a factor with levels `Nonusable` `Usable`. An assesment of whether STchange is reliable.

`STchange` A reading from an electrocardiogram; a factor with levels `No` `Yes`. An STchange indicates a blockage of the coronary artery.

`SuffHeartF` Sufficient heart frequency; a factor with levels `No`, `Yes`

`Hypertrophi` a factor with levels `No`, `Yes`. Hypertrophy refers to an increased size of the heart muscle due to exercise.

`Hyperchol` a factor with levels `No` `Yes`. Hypercholesterolemia, also called high cholesterol, is the presence of high levels of cholesterol in the blood.

`Smoker` Is the patient a smoker; a factor with levels `No`, `Yes`.

`Inherit` Hereditary predispositions for CAD; a factor with levels `No`, `Yes`.

`Heartfail` Previous heart failures; a factor with levels `No` `Yes`

`CAD` Coronary Artery Disease; a factor with levels `No` `Yes`. CAD refers to a reduction of blood flow to the heart muscle (commonly known as a heart attack). The diagnosis made from biopsies.

Details

Notice that data are collected at a heart clinic, so data do not represent the population, but are conditional on patients having ended up at the clinic.

- cad1: Complete dataset, 236 cases.
- cad2: Incomplete dataset, 67 cases. Information on (some of) the variables 'Hyperchol', 'Smoker' and 'Inherit' is missing.

References

Hansen, J. F. (1980). The clinical diagnosis of ischaemic heart disease due to coronary artery disease. Danish Medical Bulletin

Højsgaard, Søren and Thiesson, Bo (1995). BIFROST - Block recursive models Induced From Relevant knowledge, Observations and Statistical Techniques. Computational Statistics and Data Analysis, vol. 19, p. 155-175

Examples

```
data(cad1)
## maybe str(cad1) ; plot(cad1) ...
```

---

data_mathmark *Mathematics marks for students*

Description

The `mathmark` data frame has 88 rows and 5 columns.

Usage

```
data(mathmark)
```

Format

This data frame contains the following columns: mechanics, vectors, algebra, analysis, statistics.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

David Edwards, An Introduction to Graphical Modelling, Second Edition, Springer Verlag, 2000

Examples

```
data(mathmark)
```

---

data_personality *Personality traits*

Description

The peronality dataframe has 240 rows and 32 columns

Usage

```
data(personality)
```

Format

This dataframe has recordings on the following 32 variables: distant, talkatv, carelss, hardwrk, anxious, agreebl, tense, kind, opposng, relaxed, disorgn, outgoin, approvn, shy, discipl, harsh, persevr, friendl, worryin, respnsi, contrar, sociabl, lazy, coopera, quiet, organiz, criticl, lax, laidbck, withdrw, givinup, easygon

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Origin unclear

Examples

```
data(personality)
str(personality)
```

descStat *Computing simple descriptive statistics of a numeric vector.*

Description

Computing simple descriptive statistics of a numeric vector - not unlike what proc means of SAS does

Usage

```
descStat(x, na.rm = TRUE)
```

Arguments

x	A numeric vector
na.rm	Should missing values be removed

Value

A vector with named elements.

Author(s)

Gregor Gorjanc; <gregor.gorjanc@bf.uni-lj.si>

See Also

summaryBy, summary_by

Examples

```
x <- c(1, 2, 3, 4, NA, NaN)
descStat(x)
```

dietox *Growth curves of pigs in a 3x3 factorial experiment*

Description

The dietox data frame has 861 rows and 7 columns.

Usage

```
dietox
```

Format

This data frame contains the following columns:

Weight Weight in Kg

Feed Cumulated feed intake in Kg

Time Time (in weeks) in the experiment

Pig Factor; id of each pig

Evit Factor; vitamin E dose; see 'details'.

Cu Factor, copper dose; see 'details'

Start Start weight in experiment, i.e. weight at week 1.

Litter Factor, id of litter of each pig

Details

Data contains weight of slaughter pigs measured weekly for 12 weeks. Data also contains the start weight (i.e. the weight at week 1). The treatments are 3 different levels of Evit = vitamin E (dose: 0, 100, 200 mg dl-alpha-tocopheryl acetat /kg feed) in combination with 3 different levels of Cu=copper (dose: 0, 35, 175 mg/kg feed) in the feed. The cumulated feed intake is also recorded. The pigs are litter mates.

Source

Lauridsen, C., Højsgaard, S.,Sørensen, M.T. C. (1999) Influence of Dietary Rapeseed Oli, Vitamin E, and Copper on Performance and Antioxidant and Oxidative Status of Pigs. J. Anim. Sci.77:906-916

Examples

```
data(dietox)
head(dietox)
coplot(Weight ~ Time | Evit * Cu, data=dietox)
```

---

esticon *Contrasts for lm, glm, lme, and geeglm objects*

Description

Computes linear functions (i.e. weighted sums) of the estimated regression parameters. Can also test the hypothesis, that such a function is equal to a specific value.

Usage

```
esticon(obj, L, beta0, conf.int = TRUE, level = 0.95, joint.test = FALSE, ...)

## S3 method for class 'esticon_class'
coef(object, ...)

## S3 method for class 'esticon_class'
summary(object, ...)

## S3 method for class 'esticon_class'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'esticon_class'
vcov(object, ...)
```

Arguments

obj	Regression object (of type lm, glm, lme, geeglm).
L	Matrix (or vector) specifying linear functions of the regression parameters (one linear function per row). The number of columns must match the number of fitted regression parameters in the model. See 'details' below.
beta0	A vector of numbers
conf.int	TRUE
level	The confidence level
joint.test	Logical value. If TRUE a 'joint' Wald test for the hypothesis L beta = beta0 is made. Default is that the 'row-wise' tests are made, i.e. (L beta)i=beta0i. If joint.test is TRUE, then no confidence interval etc. is calculated.
...	Additional arguments; currently not used.
object	An esticon_class object.
parm	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.

Details

Let the estimated parameters of the model be

$$\beta_1, \beta_2, \ldots, \beta_p$$

A linear function of the estimates is of the form

$$l = \lambda_1\beta_1 + \lambda_2\beta_2 + \cdots + \lambda_p\beta_p$$

where $\lambda_1, \lambda_2, \ldots, \lambda_p$ is specified by the user.

The esticon function calculates l, its standard error and by default also a 95 pct confidence interval. It is sometimes of interest to test the hypothesis $H_0 : l = \beta_0$ for some value β_0 given by the user. A test is provided for the hypothesis $H_0 : l = 0$ but other values of β_0 can be specified.

In general, one can specify r such linear functions at one time by specifying L to be an $r \times p$ matrix where each row consists of p numbers $\lambda_1, \lambda_2, \ldots, \lambda_p$. Default is then that β_0 is a p vector of 0s but other values can be given.

It is possible to test simultaneously that all specified linear functions are equal to the corresponding values in β_0.

For computing contrasts among levels of a single factor, 'contrast.lm' may be more convenient.

Value

Returns a matrix with one row per linear function. Columns contain estimated coefficients, standard errors, t values, degrees of freedom, two-sided p-values, and the lower and upper endpoints of the 1-alpha confidence intervals.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
data(iris)
lm1  <- lm(Sepal.Length ~ Sepal.Width + Species + Sepal.Width : Species, data=iris)
## Note that the setosa parameters are set to zero
coef(lm1)

## Estimate the intercept for versicolor
lambda1 <- c(1, 0, 1, 0, 0, 0)
esticon(lm1, L=lambda1)

## Estimate the difference between versicolor and virgica intercept
## and test if the difference is 1
lambda2 <- c(0, 1, -1, 0, 0, 0)
esticon(lm1, L=lambda2, beta0=1)

## Do both estimates at one time
esticon(lm1, L=rbind(lambda1, lambda2), beta0=c(0, 1))

## Make a combined test for that the difference between versicolor and virgica intercept
## and difference between versicolor and virginica slope is zero:
lambda3 <- c(0, 0, 0, 0, 1, -1)
esticon(lm1, L=rbind(lambda2, lambda3), joint.test=TRUE)

# Example using esticon on coxph objects (thanks to Alessandro A. Leidi).
# Using dataset 'veteran' in the survival package
# from the Veterans' Administration Lung Cancer study

if (require(survival)){
data(veteran)
sapply(veteran, class)
levels(veteran$celltype)
attach(veteran)
veteran.s <- Surv(time, status)
```

```
coxmod <- coxph(veteran.s ~ age + celltype + trt, method='breslow')
summary(coxmod)

# compare a subject 50 years old with celltype 1
# to a subject 70 years old with celltype 2
# both subjects on the same treatment
AvB <- c(-20, -1, 0, 0, 0)

# compare a subject 40 years old with celltype 2 on treat=0
# to a subject 35 years old with celltype 3 on treat=1
CvB <- c(5, 1, -1, 0, -1)

est <- esticon(coxmod, L=rbind(AvB, CvB))
est
##exp(est[, c(2, 7, 8)])
}
```

`expr_to_fun` *Convert expression into function object.*

Description

Convert expression into function object.

Usage

```
expr_to_fun(expr_, order = NULL, vec_arg = FALSE)
```

Arguments

`expr_`	R expression.
`order`	desired order of function argument.
`vec_arg`	should the function take vector valued argument.

Examples

```
ee <- expression(b1 + (b0 - b1)*exp(-k*x) + b2*x)
ff <- expr_to_fun(ee)

ee <- expression(matrix(c(x1+x2, x1-x2, x1^2+x2^2, x1^3+x2^3), nrow=2))
ff <- expr_to_fun(ee)

ee <- expression(
  matrix(
    c(8 * x1 * (4 * x1^2 - 625 * x2^2 - 2 * x2
    - 1) + 9 * x1 - 20 * x2 * (x3 + 0.473809523809524 + exp(-x1 *
    x2)/20) * exp(-x1 * x2) - 3 * cos(x2 * x3) - 4.5,
    -20 * x1 * (x3 + 0.473809523809524 + exp(-x1 * x2)/20) * exp(-x1 * x2) + 3 * x3 *
    (x1 - cos(x2 * x3)/3 - 0.5) * sin(x2 * x3) + (-1250 * x2 - 2) * (4
```

```
    * x1^2 - 625 * x2^2 - 2 * x2 - 1),
    3 * x2 * (x1 - cos(x2 * x3)/3 -
    0.5) * sin(x2 * x3) + 400 * x3 + 189.52380952381 + 20 * exp(-x1 *
    x2)), nrow = 3))
f1 <- expr_to_fun(ee)
f2 <- expr_to_fun(ee, vec_arg=TRUE)
## Note: how long should parm be in f2?
formals(f2)$length_parm
```

`fatacid` *Fish oil in pig food*

Description

Fish oil in pig food

Usage

```
fatacid
```

Format

A dataframe.

Details

A fish oil fatty acid `X14` has been added in different concentrations to the food for pigs in a study. Interest is in studying how much of the fatty acid can be found in the tissue. The concentrations of `x14` in the food are `verb+dose+={0.0, 4.4, 6.2, 9.3}`.

The pigs are fed with this food until their weight is 60 kg. From thereof and until they are slaughtered at 100kg, their food does not contain the fish oil. At 60kg (sample=1) and 100kg (sample=2) muscle biopsies are made and the concentration of x14 is determined. Measurements on the same pig are correlated, and pigs are additionally related through litters.

References

Data courtesy of Charlotte Lauridsen, Department of Animal Science, Aarhus University, Denmark.

fev *Forced expiratory volume in children*

Description

Dataset to examine if respiratory function in children was influenced by smoking.

Usage

```
fev
```

Format

A data frame with 654 observations on the following 5 variables.

`Age` Age in years.

`FEV` Forced expiratory volume in liters per second.

`Ht` Height in centimeters.

`Gender` Gender.

`Smoke` Smoking status.

References

I. Tager and S. Weiss and B. Rosner and F. Speizer (1979). Effect of Parental Cigarette Smoking on the Pulmonary Function of Children. American Journal of Epidemiology. 110:15-26

Examples

```
data(fev)
summary(fev)
```

firstlastobs *Locate the index of the first/last unique value*

Description

Locate the index of the first/last unique value in i) a vector or of a variable in a data frame.

Usage

```
lastobs(x, ...)

firstobs(x, ...)

## Default S3 method:
lastobs(x, ...)

## Default S3 method:
firstobs(x, ...)

## S3 method for class 'formula'
lastobs(formula, data = parent.frame(), ...)

## S3 method for class 'formula'
firstobs(formula, data = parent.frame(), ...)
```

Arguments

x	A vector
...	Currently not used
formula	A formula (only the first term is used, see 'details').
data	A data frame

Details

If writing ~a + b + c as formula, then only a is considered.

Value

A vector.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
x <- c(rep(1, 5), rep(2, 3), rep(3, 7), rep(1, 4))
firstobs(x)
lastobs(x)
data(dietox)
firstobs(~Pig, data=dietox)
lastobs(~Pig, data=dietox)
```

formula_ops *Formula operations and coercion.*

Description

Formula operations and coercion as a supplement to `update.formula()`

Usage

```
formula_add_str(frm1, terms, op = "+")

formula_add(frm1, frm2)

formula_poly(chr1, n, noint = FALSE, y = NULL)

formula_nth(frm1, n)

formula_to_interaction_matrix(frm1)

formula_chr_to_form(rhs, lhs = character(0))

to_str(chr1, collapse = "+")

terms_labels(frm1)

simplify_rhs(object)

## S3 method for class 'formula'
simplify_rhs(object)

## S3 method for class 'character'
simplify_rhs(object)

as_rhs_frm(object)

as_lhs_frm(object)

as_rhs_chr(object, string = FALSE)

as_lhs_chr(object, string = FALSE)

unique_formula(list_of_formulas)
```

Arguments

`frm1`, `frm2`	Formulas to be coerced to character vectors.
`terms`	Character string.

op	Either "+" (default) or "-".
chr1	Character vector to be coerced to formulas.
n	Positive integer.
noint	Boolean.
y	Response
rhs, lhs	right-hand-side and left-hand-side for formula (as characters)
collapse	Character to use as separator.
object	Character vector or formula.
string	Boolean.
list_of_formulas	list of formulas

Examples

```
formula_poly("z", 2)
formula_poly("z", 2, noint=TRUE)

as_rhs_chr(c("a", "b", "z"))
as_rhs_chr(c("a*b", "z"))

as_rhs_chr(y~a+b+z)
as_rhs_chr(y~a+b+z, string=TRUE)
as_rhs_chr(y~a+b+z)
as_rhs_chr(y~a*b+z)
as_rhs_chr(y~a*b+z, string=TRUE)

as_lhs_chr(y~a*b+z)
as_lhs_chr(log(y)~a*b+z)      ## Not what one might expect
as_lhs_chr(cbind(y, u)~a*b+z) ## Not what one might expect

formula_chr_to_form(c("a*b", "z"))
formula_chr_to_form(c("a*b", "z"), "y")
formula_chr_to_form(c("a*b", "z"), "log(y)")

formula_add(y~a*b+z, ~-1)
formula_add(y~a*b+z, ~a:b)

formula_add_str(y~x1 + x2, "x3")
formula_add_str(y~x1 + x2, "x1")
formula_add_str(y~x1 + x2, "x1", op="-")
```

---

generate_data_list *Generate data list*

Description

Generate data list

Usage

```
generate_data_list(data., K, method = c("subgroups", "resample"))
```

Arguments

data.	A data frame
K	Number of folds
method	Method for generating data

`get_formulas` *Get formulas from model_stability_glm_class object*

Description

Get formulas from model_stability_glm_class object

Usage

```
get_formulas(object, unique = TRUE, text = FALSE)
```

Arguments

object	A model_stability_glm_class object
unique	If TRUE, return unique models
text	If TRUE, return text (rather than formula).

`haldCement` *Heat development in cement under hardening.*

Description

Heat development in cement under hardening related to the chemical composition.

Usage

```
haldCement
```

Format

A data frame with 13 observations on the following 5 variables.

`x1` Percentage (weight) of `[3Ca0][Al2O3]`

`x2` Percentage (weight) of `[3Cao][SiO2]`

`x3` Percentage (weight) of `[4Ca0][Al2O3][Fe03]`

`x4` Percentage (weight) of `[2Cao][SiO2]`

`y` Heat development measured in calories per gram cement after 180 days

References

Anders Hald (1949); Statistiske Metoder; Akademisk Forlag (in Danish), page 509.

Examples

```
data(haldCement)

if( interactive() ){
pairs( haldCement )
}
m <- lm(y ~ x1 + x2 + x3 + x4, data=haldCement)
summary(m)

# Notice: The model explains practically all variation in data;
# yet none of the explanatory variables appear to be statistically
# significant.
```

---

interaction-plot *Two-way interaction plot*

Description

Plots the mean of the response for two-way combinations of factors, thereby illustrating possible interactions.

Usage

```
interaction_plot(.data, .formula, interval = "conf.int")
```

Arguments

`.data`	A data frame
`.formula`	A formula of the form `y ~ x1 + x2`
`interval`	Either `conf.int`, `boxplot` or `none`

Note

This is a recent addition to the package and is subject to change.

Examples

```
ToothGrowth |> interaction_plot(len ~ dose + supp)
ToothGrowth |> interaction_plot(len ~ dose + supp, interval="conf.int")
ToothGrowth |> interaction_plot(len ~ dose + supp, interval="boxplot")
ToothGrowth |> interaction_plot(len ~ dose + supp, interval="none")
```

---

is-estimable *Determines if contrasts are estimable.*

Description

Determines if contrasts are estimable, that is, if the contrasts can be written as a linear function of the data.

Usage

```
is_estimable(K, null.basis)
```

Arguments

K	A matrix.
null.basis	A basis for a null space (can be found with null_basis()).

Details

Consider the setting $E(Y) = Xb$. A linear function of b, say $l'b$ is estimable if and only if there exists an r such that $r'X = l'$ or equivalently $l = X'r$. Hence l must be in the column space of X', i.e. in the orthogonal complement of the null space of X. Hence, with a basis B for the null space, is_estimable() checks if each row l of the matrix K is perpendicular to each column basis vector in B.

Value

A logical vector.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

http://web.mit.edu/18.06/www/Essays/newpaper_ver3.pdf

See Also

null_basis

Examples

```
## TO BE WRITTEN
```

---

linest *Compute linear estimates*

Description

Compute linear estimates, i.e. `L %*% beta` for a range of models. One example of linear estimates is population means (also known as LSMEANS).

Usage

```
linest(object, L = NULL, ...)

## S3 method for class 'linest_class'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'linest_class'
coef(object, ...)

## S3 method for class 'linest_class'
summary(object, ...)
```

Arguments

`object`	Model object
`L`	Either `NULL` or a matrix with p columns where p is the number of parameters in the systematic effects in the model. If `NULL` then `L` is taken to be the p times p identity matrix
`...`	Additional arguments; currently not used.
`parm`	Specification of the parameters estimates for which confidence intervals are to be calculated.
`level`	The level of the (asymptotic) confidence interval.
`confint`	Should confidence interval appear in output.

Value

A dataframe with results from computing the contrasts.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

LSmeans, LE_matrix

Examples

```
## Make balanced dataset
dat.bal <- expand.grid(list(AA=factor(1:2), BB=factor(1:3), CC=factor(1:3)))
dat.bal$y <- rnorm(nrow(dat.bal))

## Make unbalanced dataset
#   'BB' is nested within 'CC' so BB=1 is only found when CC=1
#   and BB=2,3 are found in each CC=2,3,4
dat.nst <- dat.bal
dat.nst$CC <-factor(c(1,1,2,2,2,2,1,1,3,3,3,3,1,1,4,4,4,4))

mod.bal  <- lm(y ~ AA + BB * CC, data=dat.bal)
mod.nst  <- lm(y ~ AA + BB : CC, data=dat.nst)

L <- LE_matrix(mod.nst, effect=c("BB", "CC"))
linest( mod.nst, L )
```

---

linest-get *Auxillary functions for computing lsmeans, contrasts etc*

Description

Auxillary functions for computing lsmeans, contrasts etc.

Usage

```
get_xlevels(obj)

## Default S3 method:
get_xlevels(obj)

## S3 method for class 'mer'
get_xlevels(obj)

## S3 method for class 'merMod'
get_xlevels(obj)

get_contrasts(obj)
```

```
## Default S3 method:
get_contrasts(obj)

## S3 method for class 'merMod'
get_contrasts(obj)

set_xlevels(xlev, at)

get_vartypes(obj)

set_covariate_val(xlev, covariateVal)

get_X(obj, newdata, at = NULL)

## Default S3 method:
get_X(obj, newdata, at = NULL)

## S3 method for class 'merMod'
get_X(obj, newdata, at = NULL)
```

Arguments

obj	An R object
xlev	FIXME: to be described
at	FIXME: to be described
covariateVal	FIXME: to be described
newdata	FIXME: to be described

`linest-matrix` *Linear estimates matrix*

Description

Generate matrix specifying linear estimate.

Usage

```
LE_matrix(object, effect = NULL, at = NULL)

## Default S3 method:
LE_matrix(object, effect = NULL, at = NULL)

aggregate_linest_list(linest_list)

get_linest_list(object, effect = NULL, at = NULL)
```

Arguments

object	Model object
effect	A vector of variables. For each configuration of these the estimate will be calculated.
at	Either NULL, a list or a dataframe. 1) If a list, then the list must consist of covariates (including levels of some factors) to be used in the calculations. 2) If a dataframe, the dataframe is split rowwise and the function is invoked on each row.
linest_list	Linear estimate list (as generated by get_linest_list).

Details

Check this

See Also

LSmeans, linest

Examples

```
## Two way anova:

data(warpbreaks)

## An additive model
m0 <- lm(breaks ~ wool + tension, data=warpbreaks)

## Estimate mean for each wool type, for tension="M":
K <- LE_matrix(m0, at=list(wool=c("A", "B"), tension="M"))
K

## Vanilla computation:
K %*% coef(m0)

## Alternative; also providing standard errors etc:
linest(m0, K)
esticon(m0, K)

## Estimate mean for each wool type when averaging over tension;
# two ways of doing this
K <- LE_matrix(m0, at=list(wool=c("A", "B")))
K
K <- LE_matrix(m0, effect="wool")
K
linest(m0, K)

## The linear estimate is sometimes called to "least squares mean"
# (LSmeans) or popupulation means.
# Same as
LSmeans(m0, effect="wool")
```

```
## Without mentioning 'effect' or 'at' an average across all
#predictors are calculated:
K <- LE_matrix(m0)
K
linest(m0, K)

## Because the design is balanced (9 observations per combination
#of wool and tension) this is the same as computing the average. If
#the design is not balanced, the two quantities are in general not
#the same.
mean(warpbreaks$breaks)

## Same as
LSmeans(m0)

## An interaction model
m1 <- lm(breaks ~ wool * tension, data=warpbreaks)

K <- LE_matrix(m1, at=list(wool=c("A", "B"), tension="M"))
K
linest(m1, K)
K <- LE_matrix(m1, at=list(wool=c("A", "B")))
K
linest(m1, K)
K <- LE_matrix(m1, effect="wool")
K
linest(m1, K)
LSmeans(m1, effect="wool")

K <- LE_matrix(m1)
K
linest(m1, K)
LSmeans(m1)
```

`ls-means` *Compute LS-means (aka population means or marginal means)*

Description

LS-means (least squares means, also known as population means and as marginal means) for a range of model types.

Usage

```
LSmeans(object, effect = NULL, at = NULL, level = 0.95, ...)

## Default S3 method:
LSmeans(object, effect = NULL, at = NULL, level = 0.95, ...)
```

```
## S3 method for class 'lmerMod'
LSmeans(object, effect = NULL, at = NULL, level = 0.95, adjust.df = TRUE, ...)

popMeans(object, effect = NULL, at = NULL, level = 0.95, ...)

## Default S3 method:
popMeans(object, effect = NULL, at = NULL, level = 0.95, ...)

## S3 method for class 'lmerMod'
popMeans(object, effect = NULL, at = NULL, level = 0.95, adjust.df = TRUE, ...)
```

Arguments

object	Model object
effect	A vector of variables. For each configuration of these the estimate will be calculated.
at	A list of values of covariates (including levels of some factors) to be used in the calculations
level	The level of the (asymptotic) confidence interval.
...	Additional arguments; currently not used.
adjust.df	Should denominator degrees of freedom be adjusted?

Details

There are restrictions on the formulas allowed in the model object. For example having `y ~ log(x)` will cause an error. Instead one must define the variable `logx = log(x)` and do `y ~ logx`.

Value

A dataframe with results from computing the contrasts.

Warning

Notice that `LSmeans` and `LE_matrix` fails if the model formula contains an offset (as one would have in connection with e.g. Poisson regression.

Note

`LSmeans` and `popMeans` are synonymous. Some of the code has been inspired by the **lsmeans** package.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

LE_matrix, linest

Examples

```
## Two way anova:

data(warpbreaks)

m0 <- lm(breaks ~ wool + tension, data=warpbreaks)
m1 <- lm(breaks ~ wool * tension, data=warpbreaks)
LSmeans(m0)
LSmeans(m1)

## same as:
K <- LE_matrix(m0);K
linest(m0, K)
K <- LE_matrix(m1);K
linest(m1, K)

LE_matrix(m0, effect="wool")
LSmeans(m0, effect="wool")

LE_matrix(m1, effect="wool")
LSmeans(m1, effect="wool")

LE_matrix(m0, effect=c("wool", "tension"))
LSmeans(m0, effect=c("wool", "tension"))

LE_matrix(m1, effect=c("wool", "tension"))
LSmeans(m1, effect=c("wool", "tension"))


## Regression; two parallel regression lines:

data(Puromycin)

m0 <- lm(rate ~ state + log(conc), data=Puromycin)
## Can not use LSmeans / LE_matrix here because of
## the log-transformation. Instead we must do:
Puromycin$lconc <- log( Puromycin$conc )
m1 <- lm(rate ~ state + lconc, data=Puromycin)

LE_matrix(m1)
LSmeans(m1)

LE_matrix(m1, effect="state")
LSmeans(m1, effect="state")

LE_matrix(m1, effect="state", at=list(lconc=3))
LSmeans(m1, effect="state", at=list(lconc=3))

## Non estimable contrasts

## ## Make balanced dataset
dat.bal <- expand.grid(list(AA=factor(1:2), BB=factor(1:3),
```

```
                            CC=factor(1:3)))
dat.bal$y <- rnorm(nrow(dat.bal))

## ## Make unbalanced dataset
#      'BB' is nested within 'CC' so BB=1 is only found when CC=1
#       and BB=2,3 are found in each CC=2,3,4
dat.nst <- dat.bal
dat.nst$CC <-factor(c(1, 1, 2, 2, 2, 2, 1, 1, 3, 3,
                      3, 3, 1, 1, 4, 4, 4, 4))

mod.bal  <- lm(y ~ AA + BB * CC,    data=dat.bal)
mod.nst  <- lm(y ~ AA + BB : CC, data=dat.nst)

LSmeans(mod.bal, effect=c("BB", "CC"))
LSmeans(mod.nst, effect=c("BB", "CC"))
LSmeans(mod.nst, at=list(BB=1, CC=1))

LSmeans(mod.nst, at=list(BB=1, CC=2))
## Above: NA's are correct; not an estimable function

if( require( lme4 )){
 warp.mm <- lmer(breaks ~ -1 + tension + (1|wool), data=warpbreaks)
 LSmeans(warp.mm, effect="tension")
 class(warp.mm)
 fixef(warp.mm)
 coef(summary(warp.mm))
 vcov(warp.mm)
 if (require(pbkrtest))
   vcovAdj(warp.mm)
}

LSmeans(warp.mm, effect="tension")
```

`matrix_op` *Matrix operations based on matching dimensions*

Description

Matrix operations based on matching dimensions

Usage

```
matrix_op(m1, m2, op = `+`)
```

Arguments

`m1, m2`	matrices with dimnames
`op`	the operation to be performed

Examples

```
nms1 <- letters[1:4]
mat1 <- matrix(1:8, nrow=4, ncol=4, dimnames=list(nms1, nms1))

nms2 <- letters[2:3]
mat2 <- matrix(11:18, nrow=2, ncol=4, dimnames=list(nms2, nms1))

matrix_op(mat1, mat2)
matrix_op(mat1, mat2, `*`)
```

`mb_summary` *Fast summary of microbenchmark object*

Description

Fast summary of microbenchmark object. The default summary method from the microbenchmark package is fairly slow in producing a summary (due to a call to a function from the multcomp package.)

Usage

```
mb_summary(object, unit, add.unit = TRUE, ...)

summary_mb(object, unit, add.unit = TRUE, ...)
```

Arguments

`object`	A microbenchmark object
`unit`	The time unit to be used
`add.unit`	Should time unit be added as column to resulting dataframe.
`...`	Additional arguments; currently not used.

`milkman` *Milk yield data for manually milked cows.*

Description

Milk yield data for cows milked manually twice a day (morning and evening).

Usage

```
milkman
```

Format

A data frame with 161836 observations on the following 12 variables.

`cowno` a numeric vector; cow identification

`lactno` a numeric vector; lactation number

`ampm` a numeric vector; milking time: 1: morning; 2: evening

`dfc` a numeric vector; days from calving

`my` a numeric vector; milk yield (kg)

`fatpct` a numeric vector; fat percentage

`protpct` a numeric vector; protein percentage

`lactpct` a numeric vector; lactose percentage

`scc` a numeric vector; somatic cell counts

`race` a factor with levels `RDM` `Holstein` `Jersey`

`ecmy` a numeric vector; energy corrected milk

`cowlact` Combination of cowno and lactno; necessary because the same cow may appear more than once in the dataset (in different lactations)

Details

There are data for 222 cows. Some cows appear more than once in the dataset (in different lactations) and there are 288 different lactations.

References

Friggens, N. C.; Ridder, C. and Løvendahl, P. (2007). On the Use of Milk Composition Measures to Predict the Energy Balance of Dairy Cows. J. Dairy Sci. 90:5453–5467 doi:10.3168/jds.2006-821.

This study was part of the Biosens project used data from the "Malkekoens energibalance og mobilisering" project; both were funded by the Danish Ministry of Food, Agriculture and Fisheries and the Danish Cattle Association.

Examples

```
data(milkman)
```

model_stability_glm *Model stability for glm objects*

Description

Model stability for glm objects

Usage

```
model_stability_glm(
  data.,
  model,
  n.searches = 10,
  method = c("subgroups", "resample"),
  ...
)
```

Arguments

data.	A data frame
model	A glm object
n.searches	Number of searches
method	Method for generating data
...	Additional arguments to be passed to step

NIRmilk *NIRmilk*

Description

Near infra red light (NIR) measurements are made at 152 wavelengths on 17 milk samples. While milk runs through a glass tube, infra red light is sent through the tube and the amount of light passing though the tube is measured at different wavelengths. Each milk sample was additionally analysed for fat, lactose, protein and dry matter.

Usage

```
NIRmilk
```

Format

This data frame contains 17 rows and 158 columns. The first column is the sample number. The columns Xklm contains the transmittance (fraction of electromagnetic power) transmittance through the sample at wavelength klm. The response variables are fat, protein, lactose and dm (dry matter).

Examples

```
data(NIRmilk)
```

`nir_milk` *nir_milk*

Description

Near infra red light (NIR) measurements are made at 152 wavelengths on 17 milk samples. While milk runs through a glass tube, infra red light is sent through the tube and the amount of light passing though the tube is measured at different wavelengths. Each milk sample was additionally analysed for fat, lactose, protein and dry matter.

Usage

```
nir_milk
```

Format

A list with two components x Datafrane with infra red light amount at different wavelengths (column names are the wavelengths; just remove the leading X). y Datafrane with response variables fat, protein, lactose and dm (drymatter)

See Also

NIRmilk

Examples

```
data(nir_milk)
```

`null-basis` *Finds the basis of the (right) null space.*

Description

Finds the basis of the (right) null space of a matrix, a vector (a 1-column matrix) or a model object for which a model matrix can be extracted. I.e. finds basis for the (right) null space x : Mx = 0.

Usage

```
null_basis(object)
```

Arguments

object	A matrix, a vector (a 1-column matrix) or a model object for which a model matrix can be extracted (using model.matrix).

Value

A matrix (possibly with zero columns if the null space consists only of the zero vector).

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

Null

Examples

```
M <- matrix(c(1,1,1,1,1,1,0,0,0,0,1,1), nrow=4)
null_basis(M)
MASS::Null(t(M))

M <- c(1,1,1,1)
null_basis(M)
MASS::Null(t(M))

m0 <- lm(breaks ~ wool + tension, data=warpbreaks)
null_basis(m0)
MASS::Null(t(model.matrix(m0)))

## Make balanced dataset
dat.bal   <- expand.grid(list(A=factor(1:2), B=factor(1:3), C=factor(1:3)))
dat.bal$y <- rnorm(nrow(dat.bal))

## Make unbalanced dataset: 'B' is nested within 'C' so B=1 is only
## found when C=1 and B=2,3 are found in each C=2,3,4
dat.nst <- dat.bal
dat.nst$C <-factor(c(1,1,2,2,2,2,1,1,3,3,3,3,1,1,4,4,4,4))
xtabs(y ~ C+B+A , data=dat.nst)

mod.bal  <- lm(y ~ A + B*C, data=dat.bal)
mod.nst  <- lm(y ~ A + B*C, data=dat.nst)

null_basis( mod.bal )
null_basis( mod.nst )

null_basis( model.matrix(mod.bal) )
null_basis( model.matrix(mod.nst) )

MASS::Null( t(model.matrix(mod.bal)) )
MASS::Null( t(model.matrix(mod.nst)) )
```

parseGroupFormula *Extract components from a formula with "conditioning bar"*

Description

Extract components from a formula with the form y ~ x1 + ... + xn | g1 + ... + gm

Usage

```
parseGroupFormula(form)
```

Arguments

form	A formula of the form y ~ x1 + ... + xn | g1 + ... + gm

Value

If the formula is y ~ x1 + x2 | g1 + g2 the result is

model	y ~ x1 + x2
groups	g1 + g2
groupFormula	~ g1 + g2

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
gf <- parseGroupFormula(y ~ x1 + x2 | g1 + g2)
gf
```

plot_lm *Plot linear model object*

Description

Plot linear model object

Usage

```
plot_lm(lm_fit, format = "2x2")
```

Arguments

lm_fit	An object of class 'lm'
format	The format of the plot (or a list of plots if format is "list")

Examples

```
m1 <- lm(speed ~ dist, data=cars)
plot_lm(m1)
plot_lm(m1, "2x2")
plot_lm(m1, "1x4")
plot_lm(m1, "4x1")
plot_lm(m1, "list")
```

---

potatoes *Weight and size of 20 potatoes*

Description

Weight and size of 20 potatoes. Weight in grams; size in millimeter. There are two sizes: `length` is the longest length and `width` is the shortest length across a potato.

Usage

```
potatoes
```

Format

A data frame with 20 observations on the following 3 variables.

`weight` a numeric vector

`length` a numeric vector

`width` a numeric vector

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Source

My own garden; autumn 2015.

Examples

```
data(potatoes)
plot(potatoes)
```

prostate *Prostate Tumor Gene Expression Dataset*

Description

This is the Prostate Tumor Gene Expression dataset used in Chung and Keles (2010).

Usage

```
data(prostate)
```

Format

A list with two components:

x Gene expression data. A matrix with 102 rows and 6033 columns.

y Class index. A vector with 102 elements.

Details

The prostate dataset consists of 52 prostate tumor and 50 normal samples. Normal and tumor classes are coded in 0 and 1, respectively, in y vector. Matrix x is gene expression data and arrays were normalized, log transformed, and standardized to zero mean and unit variance across genes as described in Dettling (2004) and Dettling and Beuhlmann (2002). See Chung and Keles (2010) for more details.

Source

Singh D, Febbo P, Ross K, Jackson D, Manola J, Ladd C, Tamayo P, Renshaw A, DAmico A, Richie J, Lander E, Loda M, Kantoff P, Golub T, and Sellers W (2002), "Gene expression correlates of clinical prostate cancer behavior", *Cancer Cell*, Vol. 1, pp. 203–209.

References

Chung D and Keles S (2010), "Sparse partial least squares classification for high dimensional data", *Statistical Applications in Genetics and Molecular Biology*, Vol. 9, Article 17.

Dettling M (2004), "BagBoosting for tumor classification with gene expression data", *Bioinformatics*, Vol. 20, pp. 3583–3593.

Dettling M and Beuhlmann P (2002), "Supervised clustering of genes", *Genome Biology*, Vol. 3, pp. research0069.1–0069.15.

Examples

```
data(prostate)
prostate$x[1:5,1:5]
prostate$y
```

recodeVar *Recode values of a vector*

Description

Recodes a vector with values, say 1,2 to a variable with values, say 'a', 'b'

Usage

```
recodeVar(x, src, tgt, default = NULL, keep.na = TRUE)

recode_var(x, src, tgt, default = NULL, keep.na = TRUE)
```

Arguments

x	A vector; the variable to be recoded.
src	The source values: a subset of the present values of x
tgt	The target values: the corresponding new values of x
default	Default target value for those values of x not listed in src. When default=NULL, values of x which are not given in src will be kept in the output.
keep.na	If TRUE then NA's in x will be retained in the output

Value

A vector

Warning

Care should be taken if x is a factor. A safe approach may be to convert x to a character vector using as.character.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

cut, factor, recodeVar

Examples

```
x <- c("dec", "jan", "feb", "mar", "apr", "may")
src1 <- list(c("dec", "jan", "feb"), c("mar", "apr", "may"))
tgt1 <- list("winter", "spring")
recodeVar(x, src=src1, tgt=tgt1)
#[1] "winter" "winter" "winter" "spring" "spring" "spring"
```

```
x <- c(rep(1:3, 3))
#[1] 1 2 3 1 2 3 1 2 3

## Simple usage:
recodeVar(x, src=c(1, 2), tgt=c("A", "B"))
#[1] "A" "B" NA  "A" "B" NA  "A" "B" NA

## Here we need to use lists
recodeVar(x, src=list(c(1, 2)), tgt=list("A"))
#[1] "A" "A" NA  "A" "A" NA  "A" "A" NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"), default="L")
#[1] "A" "A" "L" "A" "A" "L" "A" "A" "L"
recodeVar(x, src=list(c(1, 2), 3), tgt=list("A", "B"), default="L")
#[1] "A" "A" "B" "A" "A" "B" "A" "A" "B"

## Dealing with NA's in x
x<-c(NA,rep(1:3, 3),NA)
#[1] NA  1  2  3  1  2  3  1  2  3 NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"))
#[1] NA  "A" "A" NA  "A" "A" NA  "A" "A" NA  NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"), default="L")
#[1] NA  "A" "A" "L" "A" "A" "L" "A" "A" "L" NA
recodeVar(x, src=list(c(1, 2)), tgt=list("A"), default="L", keep.na=FALSE)
#[1] "L" "A" "A" "L" "A" "A" "L" "A" "A" "L" "L"

x <- c("no", "yes", "not registered", "no", "yes", "no answer")
recodeVar(x, src = c("no", "yes"), tgt = c("0", "1"), default = NA)
```

`recover_pca_data` *Recover data from principal component analysis*

Description

Recover data from principal component analysis based on the first (typically few) components.

Usage

```
recover_pca_data(object, comp = 1)
```

Arguments

`object`	An object of class `prcomp`.
`comp`	The number of components to be used. Must be smaller than the number of variables.

Value

A dataframe

Examples

```
crime <- doBy::crimeRate
rownames(crime) <- crime$state
crime$state <- NULL

o <- order(apply(scale(crime), 1, sum))
dat <- crime[o,]
head(dat)
tail(dat)
matplot(scale(dat), type="l")

pc1 <- prcomp(dat, scale. = TRUE)
summary(pc1)
rec2 <- recover_pca_data(pc1, 2)

pairs(rec2)

par(mfrow=c(1,2))
matplot(scale(dat), type="l")
matplot(scale(rec2), type="l")

j <- merge(dat, rec2, by=0)
pairs(j[,-1])
```

---

renameCol *Rename columns in a matrix or a dataframe.*

Description

Rename columns in a matrix or a dataframe.

Usage

```
renameCol(indata, src, tgt)
```

Arguments

indata	A dataframe or a matrix
src	Source: Vector of names of columns in indata to be renamed. Can also be a vector of column numbers.
tgt	Target: Vector with corresponding new names in the output.

Value

A dataframe if indata is a dataframe; a matrix in indata is a matrix.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
renameCol(CO2, 1:2, c("kk", "ll"))
renameCol(CO2, c("Plant", "Type"), c("kk", "ll"))

# These fail - as they should:
# renameCol(CO2, c("Plant", "Type", "conc"), c("kk", "ll"))
# renameCol(CO2, c("Plant", "Type", "Plant"), c("kk", "ll"))
```

---

scale_df *Scaling numerical values*

Description

Similar to 'base::scale' but applies to scales / centers only numeric values in data.

Usage

```
scale_df(x, center = TRUE, scale = TRUE)

scale2(x, center = TRUE, scale = TRUE)
```

Arguments

x	dataframe or matrix
center	Logical, should data be centered.
scale	Logical, should data be scaled.

Details

- If x is not a dataframe, then base::scale is invoked on x.
- Suppose x is a dataframe. Then base::scale is invoked on all columns that are numeric, integer or logical.

Value

An object of same class as x

Examples

```
scale2(iris)
```

`section_fun` *Section a function and set default values in function*

Description

Section a functions domain by fixing certain arguments of a function call.

Usage

```
set_default(fun, nms, vls = NULL)

section_fun(fun, nms, vls = NULL, method = "args")

section_fun_sub(fun, nms, vls = NULL, envir = parent.frame())

section_fun_env(fun, nms, vls = NULL)

get_section(object)

get_fun(object)
```

Arguments

`fun`	Function to be sectioned
`nms`	Either a named list of the form name=value where each name is the name of an argument of the function (in which case `vls` is ignored) or a character vector of names of arguments.
`vls`	A vector or list of values of the arguments
`method`	One of the following: 1) "args" (default); based on substituting fixed values into the function argument list as default values). For backward compatibility can also be "def". 2) "body" for substituting fixed values into the function body. For backward compatibility can also be "sub". 3) "env": (for environment); using an auxillary argument for storing sectioned values.
`envir`	Environment
`object`	An object from section_fun (a scaffold object).

Details

Let E be a subset of the cartesian product X x Y where X and Y are some sets. Consider a function f(x,y) defined on E. Then for any x in X, the section of E defined by x (denoted Ex) is the set of ys in Y such that (x, y) is in E. Correspondingly, the section of f(x,y) defined by x is the function f_x defined on Ex given by $f_x(y)=f(x,y)$.

`section_fun` is a wrapper for calling `set_default` (default method), `section_fun_env` or `section_fun_sub`. Notice that creating a sectioned function with `section_fun_sub` can be time consuming.

Value

A new function: The input function `fun` but with certain arguments fixed at specific values.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk> based on code adapted from the curry package.

See Also

`bquote_fun_list()`

Examples

```
f  <- function(x, y){x + y}

f_ <- section_fun(f, list(y = 10),    method="args") ## "def"" is default
f_ <- section_fun(f, nms="y", vls=10, method="args") ## SAME AS ABOVE
f_
f_(x=1)

f_ <- section_fun(f, list(y = 10),    method="body") ##
f_ <- section_fun(f, nms="y", vls=10, method="body") ## SAME AS ABOVE
f_
f_(x=1)

f_ <- section_fun(f, list(y = 10),    method="env")
f_ <- section_fun(f, nms="y", vls=10, method="env") ## SAME AS ABOVE
f_
f_(x=1)
get_section(f_)
get_fun(f_)


## With more complicated values:
g <- function(A, B) {
  A + B
}
g_ <- section_fun(g, list(A = matrix(1:4, nrow=2)))
g_ <- section_fun(g, "A", list(matrix(1:4, nrow=2)))
g_(diag(1, 2))

g_ <- section_fun(g, list(A = matrix(1:4, nrow=2)))

## Using built in function
set.seed(123)
rnorm5 <- section_fun(rnorm, list(n=5))
rnorm5(0, 1)

set.seed(123)
rnorm(5)
```

set_list_set_matrix *Matrix representatation of list of vectors and vice versa*

Description

Matrix representatation of list of vectors and vice versa

Usage

```
set_list2matrix(set_list, aggregate = FALSE)

matrix2set_list(set_matrix)
```

Arguments

set_list	list of vectors
aggregate	should the vectors be aggregated
set_matrix	matrix representatation

Examples

```
l <- list(c(1,2,3), c(3,2,4), c(3,2,4))
m1 <- set_list2matrix(l)
m1
matrix2set_list(m1)

m2 <- set_list2matrix(l, aggregate=TRUE)
m2
matrix2set_list(m2)
```

split_byrow_bycol *Split matrix or dataframe into list*

Description

Split matrix or dataframe into list by columns or by rows

Usage

```
split_bycol(x, idx = NULL, as.list = FALSE)

split_byrow(x, idx = NULL)
```

Arguments

x	Matrix or dataframe.
idx	Index to split by. If NULL, split by columns or rows.
as.list	If TRUE, return list of dataframes. If FALSE, return list of matrices.

Examples

```
x <- mtcars[1:3, 1:6]
x  |> split_bycol()
x  |> split_bycol(as.list=TRUE)
x  |> split_bycol(as.list=FALSE)
x  |> split_bycol(idx=c(1,1,1,2,2,3,3,3))
## x  |> split_bycol(idx=c(1,1,7,2,2,3,3,3)) ## Gives error

x <- mtcars[1:6, 1:6]
x  |> split_byrow()
x  |> split_byrow(idx=c(1,1,2,2))

m <- as.matrix(x)
u <- x |> split_byrow(idx=c(1,1,2,2))
y <- m |> split_byrow(idx=c(1,1,2,2))
```

---

sub_seq *Find sub-sequences of identical elements in a vector.*

Description

Find sub-sequences of identical elements in a vector.

Usage

```
subSeq(x, item = NULL)

sub_seq(x, item = NULL)

is_grouped(x)

rle2(x)
```

Arguments

x	An atomic vector or a factor.
item	Optionally a specific value to look for in x.

Details

- sub_seq is synonymous with subSeq
- rle2 is identical to rle (from base) but rle2 works on factors as input (a factor is coerced to character).
- is_grouped checks if the values in x are clustered into the smallest number of clusters.

Value

A dataframe.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

rle

Examples

```
x <- c(1, 1, 1, 0, 0, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 3)
(ans <- subSeq(x))
ans$value

## Notice: Same results below
subSeq(x, item=1)
subSeq(x, item="1")

xc <- as.character(x)
(ans<-subSeq(xc))
ans$value

## Notice: Same results below
subSeq(xc, item="1")
subSeq(xc, item=1)

is_grouped(x)
is_grouped(sort(x))
is_grouped(xc)
is_grouped(sort(xc))
```

---

taylor *Taylor expansion (one dimension)*

Description

Returns Taylor polynomial approximating a function fn(x)

Usage

```
taylor(fn, x0, ord = 1)
```

Arguments

fn	A function of one variable and that variable must be named 'x'.
x0	The point in which to to the Taylor expansion.
ord	The order of the Taylor expansion.

Value

function.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
fn <- function(x) log(x)
ord <- 2
x0 <- 2

xv  <- seq(.2, 5, .1)
plot(xv, fn(xv), type="l")
lines(xv, taylor(fn, x0=x0, ord=ord)(xv), lty=2)
abline(v=x0)

fn <- function(x)sin(x)
ord <- 4
x0 <- 0
xv <- seq(-2*pi, 2*pi, 0.1)
plot(xv, fn(xv), type="l")
lines(xv, taylor(fn, x0=x0, ord=ord)(xv), lty=2)
abline(v=x0)
```

---

tidy-esticon *Tidy an esticon object*

Description

Tidy summarizes information about the components of the object.

Usage

```
## S3 method for class 'esticon_class'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

Arguments

x	A 'esticon_class' object (produced by esticon methods).
conf.int	Should confidence intervals be added.
conf.level	Desired confidence level.
...	Additional arguments; currently not used.

---

tidy-linest *Tidy a linest object*

Description

Tidy summarizes information about the components of the object.

Usage

```
## S3 method for class 'linest_class'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

Arguments

x	A 'linest_class' object (produced by linest methods).
conf.int	Should confidence intervals be added.
conf.level	Desired confidence level.
...	Additional arguments; currently not used.

---

timeSinceEvent *Calculate "time since event" in a vector.*

Description

Calculate "time since event" in a vector.

Usage

```
timeSinceEvent(yvar, tvar = seq_along(yvar))
```

Arguments

yvar	A numerical or logical vector specifying the events
tvar	An optional vector specifying time

Details

Events are coded as 1 in numeric vector (and non-events are coded with values different from 1). `timeSinceEvent` will give the time since event (with and without sign). In a logical vector, events are coded as TRUE and all non-events as FALSE.

Value

A dataframe with columns 'yvar', 'tvar', 'abs.tse' (absolute time since nearest event), 'sign.tse' (signed time since nearest event) and 'run' (indicator of the time window around each event).

Note

NA's in yvar are converted to zeros.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

subSeq, rle

Examples

```
## Events:
yvar <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
          0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0)

## Plot results:
tse <- timeSinceEvent(yvar)
plot(sign.tse ~ tvar, data=tse, type="b")
grid()
rug(tse$tvar[tse$yvar==1], col=4, lwd=4)
points(scale(tse$run), col=tse$run, lwd=2)
lines(abs.tse + .2 ~ tvar, data=tse, type="b", col=3)

## Find times for which time since an event is at most 1:
tse$tvar[tse$abs <= 1]

yvar <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
)

tvar <- c(207, 208, 208, 208, 209, 209, 209, 209, 210, 210, 211, 211,
211, 212, 213, 213, 214, 214, 215, 216, 216, 216, 216, 217, 217,
217, 218, 218, 219, 219, 219, 219, 220, 220, 221, 221, 221, 221,
222, 222, 222)

timeSinceEvent(yvar, tvar)
```

`truncate0` *Truncate values in a matrix / vector to zero if they are below a certain threshold.*

Description

Truncate values in a matrix / vector to zero if they are below a certain threshold.

Usage

```
truncate0(x, tol = 0.6, sparse = TRUE)
```

Arguments

`x`	matrix / vector
`tol`	threshold
`sparse`	logical; if TRUE and `x` is a matrix, return a sparse matrix

`which.maxn` *Where are the n largest or n smallest elements in a numeric vector ?*

Description

Determines the locations, i.e., indices of the n largest or n smallest elements of a numeric vector.

Usage

```
which.maxn(x, n = 1)
```

Arguments

`x`	numeric vector
`n`	integer >= 1

Value

A vector of length at most n with the indices of the n largest / smaller elements. NAs are discarded and that can cause the vector to be smaller than n.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`which.max`, `which.min`

Examples

```
x <- c(1:4, 0:5, 11, NA, NA)
ii <- which.minn(x, 5)

x <- c(1, rep(NA,10), 2)
ii <- which.minn(x, 5)
```

Index