Package 'dvmisc'

July 22, 2025

Type Package

Title Convenience Functions, Moving Window Statistics, and Graphics

Version 1.1.4

Date 2019-12-15

Author Dane R. Van Domelen

Maintainer Dane R. Van Domelen <vandomed@gmail.com>

Description Contains functions that do something convenient (e.g. create BMI categories), functions for calculating moving-window statistics efficiently, and functions for generating various figures (e.g. histograms with fitted probability mass/density function).

License GPL-3

Depends rbenchmark, dplyr

Imports cubature, data.table, ggplot2, graphics, MASS, mvtnorm, pracma, Rcpp (>= 0.12.15), stats, survey, tab, utils

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 6.1.1

Suggests knitr, microbenchmark, printr, rmarkdown, RcppRoll

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-12-16 20:10:09 UTC

Contents

bmi3 3
bmi4 3
cleancut 4
clean_glm 4
create_qgroups 5
create_qgroups_svy 6

cut_decreasing . . . 7
dots_bars . . . 8
dvmisc . . . 10
expand_grid . . . 11
gammareg . . . 11
get_mse . . . 12
headtail . . . 13
histo . . . 14
inside . . . 16
interval_groups . . . 17
iterate . . . 18
list_override . . . 19
logit_prob . . . 20
lognormalreg . . . 20
logodds_graph . . . 21
max_n . . . 22
means_graph . . . 22
mean_i . . . 23
min_n . . . 24
mle_gamma . . . 25
mle_gamma_lnorm . . . 26
mle_lnorm . . . 27
mle_lnorm_lnorm . . . 28
moving_mean . . . 29
n_2t_equal . . . 29
n_2t_unequal . . . 30
odds_prob . . . 31
plot_ll . . . 31
pooled_var . . . 32
power_2t_equal . . . 33
power_2t_unequal . . . 33
prob_logit . . . 34
prob_odds . . . 34
quant_groups . . . 35
quant_groups_svy . . . 35
reverse_cut . . . 36
sliding_cor . . . 37
sliding_cov . . . 37
sumsim . . . 38
sum_i . . . 39
trim . . . 40
truerange . . . 41
which.max2 . . . 41
which.min2 . . . 42
which_max_im . . . 43
which_max_iv . . . 44
which_max_nm . . . 45
which_max_nv . . . 46

which_min_im 47
which_min_iv 48
which_min_nm 49
which_min_nv 50

Index **51**

`bmi3` *Convert Continuous BMI Values into 3-Level Factor*

Description

Converts a continuous BMI variable into a 3-level factor variable: Normal weight if `[-Inf, 25)`, Overweight if `[25, 30)`, and Obese if `[30, Inf)`.

Usage

```
bmi3(x, labels = TRUE)
```

Arguments

`x`	Numeric vector of BMI values.
`labels`	If `TRUE`, factor levels are labeled `"Normal weight"`, `"Overweight"`, and `"Obese"`; if `FALSE`, factor levels are `[-Inf, 25)`, `[25, 30)`, and `[30, Inf)`.

Value

Factor variable with 3 levels.

`bmi4` *Convert Continuous BMI Values into 4-Level Factor*

Description

Converts a continuous BMI variable into a 4-level factor variable: Underweight if `[-Inf, 18.5)`, Normal weight if `[18.5, 25)`, Overweight if `[25, 30)`, and Obese if `[30, Inf)`.

Usage

```
bmi4(x, labels = TRUE)
```

Arguments

`x`	Numeric vector of BMI values.
`labels`	If `TRUE`, factor levels are labeled `"Underweight"`, `"Normal weight"`, `"Overweight"`, and `"Obese"`; if `FALSE`, factor levels are `[-Inf, 18.5)`, `[18.5, 25)`, `[25, 30)`, and `[30, Inf)`.

Value

Factor variable with 4 levels.

---

cleancut *Convert Numeric to Factor with Convenient Interface*

Description

So you can stop guess-and-checking with `cut`.

Usage

```
cleancut(x, breaks, labels = NULL)
```

Arguments

`x`	Numeric vector.
`breaks`	Character string, e.g. `"[-Inf, 0), [0, 10], (10, Inf)"`.
`labels`	Character vector.

Value

Factor or integer vector.

Examples

```
x <- rnorm(100)
y <- cleancut(x, "(-Inf, -1), [-1, 1], (1, Inf)")
tapply(x, y, range)

y <- cleancut(x, "(-Inf, -1), [-1, 1], (1, Inf)", c("<-1", "-1 to 1", ">1"))
tapply(x, y, range)
```

---

clean_glm *Create a Clean Summary Table from a glm Object*

Description

Formats a `glm` object for printing to console or inputting to `kable`.

Usage

```
clean_glm(fit, columns = NULL, expand_factors = TRUE,
  variable_labels = NULL, prep_kable = FALSE, decimals = 2,
  formatp_list = NULL)
```

Arguments

fit	Object returned from glm.
columns	Character vector specifying what columns to include. Choices for each element are "beta", "se", "betaci", "beta_se", "beta_ci" "or", "orci", "or_ci", "hr", "hrci", "hr_ci"), "z", "t", and "p".
expand_factors	Logical value for whether to include two blank rows for factor variables (name of variable and reference group).
variable_labels	Character vector in case you want labels other than the variable names.
prep_kable	Logical value for whether to prepare for printing via kable. Right now, it just adds forward slashes so factor levels are indented, which only applies if there are factor variables and expand_factors = TRUE.
decimals	Numeric value of vector specifying number of decimal places for each column.
formatp_list	Arguments to pass to formatp.

Value

Data frame.

Examples

```
fit <- glm(mpg ~ wt + as.factor(cyl) + hp, data = mtcars)
clean_glm(fit)
fit %>% clean_glm(prep_kable = TRUE) %>% knitr::kable()
```

---

create_qgroups *Create Quantile Groups*

Description

Combines quantile and cut into a single function, with strata-specific quantiles possible. For example, you could create sex-specific height tertiles with create_qgroups(height, groups = 3, strata = sex). Compatible with **dplyr** functions like mutate and transmute.

Usage

```
create_qgroups(x, groups = 4, probs = seq(1/groups, 1 - 1/groups,
  1/groups), strata = NULL, quantile_list = list(na.rm = TRUE),
  cut_list = list(include.lowest = TRUE))
```

Arguments

x	Numeric vector.
groups	Numeric value, e.g. 3 for tertiles, 4 for quartiles, etc.
probs	Numeric vector.
strata	Factor specifying subgroups to calculate quantiles within. For multivariable subgroups, you can use interaction.
quantile_list	Arguments to pass to quantile.
cut_list	Arguments to pass to cut.

Value

Factor variable.

Examples

```
# In mtcars dataset, create tertiles for mpg
mtcars$mpg_tertiles <- create_qgroups(mtcars$mpg, groups = 3)
table(mtcars$mpg_tertiles)

# Define tertile cutpoints separately for 4-, 6-, and 8-cylinder vehicles
mtcars$mpg_tertiles <- create_qgroups(mtcars$mpg, groups = 3, strata = mtcars$cyl)
table(mtcars$mpg_tertiles)

# Works with dplyr functions like mutate
mtcars <- mtcars %>%
  dplyr::mutate(mpg_tertiles = create_qgroups(mpg, groups = 3, strata = cyl))
table(mtcars$mpg_tertiles)

# Can embed in lm, glm, etc.
summary(lm(mpg ~ create_qgroups(wt), data = mtcars))
```

---

create_qgroups_svy *Create Quantile Groups (Complex Survey Data)*

Description

Complex survey version of create_qgroups. Relies heavily on the **survey** package [1,2].

Usage

```
create_qgroups_svy(x, groups = 4, probs = seq(1/groups, 1 - 1/groups,
  1/groups), strata = NULL, design, svyquantile_list = list(na.rm =
  TRUE), cut_list = list(include.lowest = TRUE))
```

Arguments

x	Numeric vector.
groups	Numeric value, e.g. 3 for tertiles, 4 for quartiles, etc.
probs	Numeric vector.
strata	Factor specifying subgroups to calculate quantiles within. For multivariable subgroups, you can use interaction.
design	Survey design object.
svyquantile_list	Arguments to pass to svyquantile.
cut_list	Arguments to pass to cut.

Value

Factor variable.

References

1. Therneau, T. (2015). A Package for Survival Analysis in S. R package version 2.38. https://cran.r-project.org/package=survival.

2. Therneau, T.M. and Grambsch, P.M. (2000). Modeling Survival Data: Extending the Cox Model. Springer, New York. ISBN 0-387-98784-3.

---

cut_decreasing *Cut with Decreasing Factor Levels*

Description

Convenience function to get decreasing factor levels from cut. Currently requires specifying breaks as vector of cutpoints rather than number of desired intervals.

Usage

```
cut_decreasing(x, breaks, include.lowest = FALSE, right = TRUE, ...)
```

Arguments

x, breaks, include.lowest, right	See cut. specifying number of intervals is not currently supported).
...	Arguments to pass to cut.

Value

Factor variable.

Examples

```
# In mtcars dataset, create 3 mpg groups
table(cut(mtcars$mpg, breaks = c(-Inf, 15, 20, Inf)))

# Repeat with cut_decreasing to get factor levels ordered from high to low.
# To match cut here, need to specify right = FALSE
table(cut_decreasing(mtcars$mpg, breaks = c(Inf, 20, 15, -Inf), right = FALSE))

# You can specify breaks from low to high, but then include.lowest and right
# arguments get confusing
table(cut_decreasing(mtcars$mpg, breaks = c(-Inf, 15, 20, Inf), right = TRUE))
```

---

dots_bars *Plot Points +/- Error Bars*

Description

Creates plot showing user-specified points (e.g. means, medians, regression coefficients) along with user-specified error bars (e.g. standard deviations, min/max, 95% confidence intervals).

Usage

```
dots_bars(y = NULL, bars = NULL, bars.lower = y - bars,
  bars.upper = y + bars, truth = NULL, group.labels = NULL,
  group.dividers = TRUE, subgroup.spacing = 1,
  subgroup.labels = NULL, subgroup.pch = NULL, subgroup.col = NULL,
  points.list = NULL, arrows.list = NULL, xaxis.list = NULL,
  yaxis.list = xaxis.list, abline.dividers.list = NULL,
  abline.truth.list = NULL, legend.list = NULL, ...)
```

Arguments

y	Numeric vector of y-values for different groups, or numeric matrix where each column contains y-values for clustered subgroups within a group.
bars	Numeric vector or matrix (matching whichever type y is) specifying the length of the error bar for each group/subgroup (i.e. distance from point to one end of error bar).
bars.lower	Numeric vector or matrix (matching whichever type y is) specifying the position of the lower end of the error bar for each group/subgroup.
bars.upper	Numeric vector or matrix (matching whichever type y is) specifying the position of the upper end of the error bar for each group/subgroup.
truth	Numeric value specifying true value of parameter being estimated. If specified, a horizontal reference line is added to the plot.
group.labels	Character vector of labels for the groups.
group.dividers	Logical value for whether to add vertical lines distinguishing the groups.

`subgroup.spacing`
: Numeric value controlling the amount of spacing between subgroups, with values > 1 corresponding to more spacing.

`subgroup.labels`
: Character vector giving labels for the subgroups.

`subgroup.pch`
: Plotting symbol for different subgroups within each group.

`subgroup.col`
: Plotting color for different subgroups within each group.

`points.list`
: Optional list of inputs to pass to `points`.

`arrows.list`
: Optional list of inputs to pass to `arrows`.

`xaxis.list`
: Optional list of inputs to pass to `axis` for x-axis.

`yaxis.list`
: Optional list of inputs to pass to `axis` for y-axis.

`abline.dividers.list`
: Optional list of inputs to pass to `abline` for group dividers. Only used if `group.dividers = TRUE`.

`abline.truth.list`
: Optional list of inputs to pass to `abline` for horizontal line at true value of parameter. Only used if `truth` is specified.

`legend.list`
: Optional list of inputs to pass to `legend`.

`...`
: Additional arguments to pass to `plot` function.

Value

Plot showing points +/- error bars across groups/subgroups.

Examples

```
# Generate 100 values from normal distributions with different means, and
# graph mean +/- standard deviation across groups
dat <- cbind(rnorm(100, 2), rnorm(100, 2.5), rnorm(100, 1.75))
means <- apply(dat, 2, mean)
sds <- apply(dat, 2, sd)
fig1 <- dots_bars(y = means, bars = sds, main = "Mean +/- SD by Group",
                  ylab = "Mean +/- SD")

# Simulate BMI values for males and females in 3 different age groups, and
# graph mean +/- 95% CI
sex <- as.factor(c(rep("Male", 300), rep("Female", 300)))
age <- as.factor(rep(c("Young", "Middle", "Old"), 2))
bmi <- c(rnorm(100, 25, 4), rnorm(100, 26, 4.25), rnorm(100, 27, 4.5),
         rnorm(100, 26.5, 4.5), rnorm(100, 27.25, 4.75), rnorm(100, 28, 5))
dat <- data.frame(sex = sex, age = age, bmi = bmi)
means <- tapply(dat$bmi, dat[, c("sex", "age")], mean)
ci.lower <- tapply(dat$bmi, dat[, c("sex", "age")],
                   function(x) t.test(x)$conf.int[1])
ci.upper <- tapply(dat$bmi, dat[, c("sex", "age")],
                   function(x) t.test(x)$conf.int[2])
fig2 <- dots_bars(y = means, bars.lower = ci.lower, bars.upper = ci.upper,
                  main = "BMI by Sex and Age",
```

```
        ylab = "BMI (mean +/- CI)",
        xlab = "Age group")
```

---

dvmisc *Convenience Functions, Moving Window Statistics, and Graphics*

Description

Contains functions that do something convenient (e.g. create BMI categories), functions for calculating moving-window statistics efficiently, and functions for generating various figures (e.g. histograms with fitted probability mass/density function).

Details

Package:	dvmisc
Type:	Package
Version:	1.1.4
Date:	2019-12-15
License:	GPL-3

See CRAN documentation for full list of functions.

Author(s)

Dane R. Van Domelen
<vandomed@gmail.com>

References

Eddelbuettel, D. and Francois, R. (2011) Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18. http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, D. (2013) Seamless R and C++ Integration with Rcpp. Springer, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Balamuta, J.J. (2017). Extending R with C++: A Brief Introduction to Rcpp. PeerJ Preprints 5:e3188v1. https://doi.org/10.7287/peerj.preprints.3188v1.

Acknowledgment: This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0940903.

expand_grid *Similar to expand.grid but with Sequences Reversed and Ability to Treat Variables as Sets*

Description

Loops over the last argument, then the second-last, and so on. It should be faster than `expand.grid`.

Usage

```
expand_grid(..., together = NULL)
```

Arguments

`...`	Vectors you want all combinations of.
`together`	Data frame of vectors, where each row is a set of parameter values that are always kept together.

Value

Data table.

Examples

```
# Simple example of expand.grid vs. expand_grid
expand.grid(x = c("a", "b", "c"), y = c(1, 2), z = c(TRUE, FALSE))
expand_grid(x = c("a", "b", "c"), y = c(1, 2), z = c(TRUE, FALSE))

# How to keep certain variables together
expand_grid(x = 1: 5,
            together = data.frame(y = c("1a", "2a"), z = c("1b", "2b")))
```

gammareg *Constant-Scale Gamma Model for Y vs. Covariates with Y Potentially Subject to Multiplicative Lognormal Errors*

Description

Uses maximum likelihood to fit Y|**X** ~ Gamma(exp(beta_0 + **beta_x**^T **X**), b), with the shape-scale (as opposed to shape-rate) parameterization described in `GammaDist`. Y can be precisely measured or subject to multiplicative mean-1 lognormal errors, in which case replicates can be incorporated by specifying `y` as a list.

Usage

```
gammareg(y, x = NULL, merror = FALSE, integrate_tol = 1e-08,
  integrate_tol_hessian = integrate_tol, estimate_var = TRUE,
  fix_posdef = FALSE, ...)
```

Arguments

y	Numeric vector.
x	Numeric vector or matrix. If NULL, model reduces to marginal Gamma model Y ~ Gamma(exp(beta_0), b).
merror	Logical value for whether to model multiplicative lognormal measurement errors in Y.
integrate_tol	Numeric value specifying the tol input to hcubature. Only used if merror = TRUE.
integrate_tol_hessian	Same as integrate_tol, but for use when estimating the Hessian matrix only. Sometimes more precise integration (i.e. smaller tolerance) than used for maximizing the likelihood helps prevent cases where the inverse Hessian is not positive definite.
estimate_var	Logical value for whether to return Hessian-based variance-covariance matrix.
fix_posdef	Logical value for whether to repeatedly reduce integrate_tol_hessian by factor of 5 and re-estimate Hessian to try to avoid non-positive definite variance-covariance matrix.
...	Additional arguments to pass to nlminb.

Value

List containing:

1. Numeric vector of parameter estimates.
2. Variance-covariance matrix (if estimate_var = TRUE).
3. Returned nlminb object from maximizing the log-likelihood function.
4. Akaike information criterion (AIC).

---

get_mse *Extract Mean Squared Error (MSE) from Fitted Regression Model*

Description

The MSE, defined as the sum of the squared residuals divided by n-p (n = number of observations, p = number of regression coefficients), is an unbiased estimator for the error variance in a linear regression model. This is a convenience function that extracts the MSE from a fitted lm or glm object. The code is rev(anova(model.fit)$"Mean Sq")[1] if model.fit is a lm object and sum(model.fit$residuals^2) / model.fit$df.residual if model.fit is a glm object.

Usage

```
get_mse(model.fit, var.estimate = FALSE)
```

Arguments

model.fit	Fitted regression model returned from lm or glm.
var.estimate	If TRUE, function returns a variance estimate for the error variance, defined as 2 * MSE^2 / (n - p).

Value

If var.estimate = FALSE, numeric value indicating the MSE; if var.estimate = TRUE, named numeric vector indicating both the MSE and a variance estimate for the error variance.

Examples

```
# Generate 100 values: Y = 0.5 + 1.25 X + e, e ~ N(0, 1)
set.seed(123)
x <- rnorm(100)
y <- 0.5 + 1.25 * x + rnorm(100, sd = 1)

# Fit regression model using lm and using glm
lm.fit <- lm(y ~ x)
glm.fit <- glm(y ~ x)

# Extract MSE from lm.fit and glm.fit
get_mse(lm.fit)
get_mse(glm.fit)
```

---

headtail *Return the First and Last Part of an Object*

Description

Simply head and tail combined.

Usage

```
headtail(x, ...)
```

Arguments

x	Input object.
...	Additional arguments to pass to head and tail functions.

Value

Same class as x.

Examples

```
# Generate data from N(0, 1), sort, and look at smallest and largest 3 values
x <- rnorm(1000)
x.sorted <- sort(x)
headtail(x.sorted, 3)
```

`histo` *Histogram with Added Options*

Description

Similar to base R function `hist`, but with two added features: (1) Can overlay one or more fitted probability density/mass functions (PDFs/PMFs) for any univariate distribution supported in R (see `Distributions`); and (2) Can generate more of a barplot type histogram, where each possible value gets its own bin centered over its value (useful for discrete variables with not too many possible values).

Usage

```
histo(x, dis = "none", dis_shift = NULL, integer_breaks = NULL,
  colors = rep("black", length(dis)), lty = 1:length(dis),
  legend_form = ifelse(length(dis) == 1, 0, 1), aic_decimals = 1,
  points_list = NULL, axis_list = NULL, legend_list = NULL, ...)
```

Arguments

`x`	Numeric vector of values.

`dis`	Character vector indicating which distributions should be used to add fitted PDF/PMF to the histogram. If not `"none"`, choices for each element are:
	`"beta"`
	`"binom"` (must specify `size`)
	`"cauchy"`
	`"chisq"`
	`"exp"`
	`"f"`
	`"gamma"`
	`"geom"`
	`"hyper"` (must specify total number of balls in urn, `N`, and number of balls drawn each time, `k`)
	`"lnorm"`
	`"nbinom"` (must specify `size`)
	`"norm"`
	`"pois"`,

	"t" "unif" "weibull"
dis_shift	Numeric value for shifting the fitted PDF/PMF along the x-axis of the histogram.
integer_breaks	If TRUE, integers covering the range of x are used for breaks, so there is one bin for each integer. Useful for discrete distributions that don't take on too many unique values.
colors	Character vector of colors for each PDF/PMF.
lty	Integer vector specifying line types for each curve.
legend_form	Integer value controlling what type of legend to include. Choices are 0 for no legend, 1 for legend naming each distribution, and 2 for legend naming each distribution and the corresponding AIC.
aic_decimals	Integer value for number of decimals for AIC.
points_list	Optional list of inputs to pass to points function, which is used to add the fitted PDF/PMF.
axis_list	Optional list of inputs to pass to axis.
legend_list	Optional list of inputs to pass to legend.
...	May include arguments to pass to hist and/or parameter values needed for certain distributions (size if dis = "binom" or dis = "nbinom", N and k if dis = "hyper").

Details

When x takes on whole numbers, you typically want to set dis_shift = -0.5 if right = TRUE (hist's default) and dis_shift = 0.5 if right = FALSE. The function will do this internally by default.

To illustrate, suppose a particular bin represents (7, 10]. Its midpoint will be at x = 8.5 on the graph. But if input values are whole numbers, this bin really only includes values of 8, 9, and 10, which have a mean of 9. So you really want f(9) to appear at x = 8.5. This requires shifting the curve to the left 0.5 units, i.e. setting dis_shift = -0.5.

When x takes on whole numbers with not too many unique values, you may want the histogram to show one bin for each integer. You can do this by setting integer_breaks = TRUE. By default, the function sets integer_breaks = TRUE if x contains whole numbers with 10 or fewer unique values.

Value

Histogram with fitted PDFs/PMFs if requested.

Examples

```
# Sample 10,000 Poisson(2) values and commpare default hist vs. histo
set.seed(123)
x <- rpois(n = 10000, lambda = 2)
par(mfrow = c(1, 2))
hist(x, main = "hist function")
histo(x, main = "histo function")
```

```
# Sample 10,000 lognormal(0, 0.35) values. Create histogram with curves
# showing fitted lognormal, normal, and Gamma PDFs.
set.seed(123)
x <- rlnorm(n = 10000, meanlog = 0, sdlog = 0.35)
par(mfrow = c(1, 1))
histo(x, c("lnorm", "norm", "gamma"), main = "X ~ Lognormal(0, 0.35)")

# Generate 10,000 Binomial(8, 0.25) values. Create histogram, specifying
# size = 5, with blue line/points showing fitted PMF.
set.seed(123)
x <- rbinom(n = 10000, size = 5, prob = 0.25)
par(mfrow = c(1, 1))
histo(x, dis = "binom", size = 5, colors = "blue",
      points_list = list(type = "b"))
```

`inside` *Check Whether Numeric Value Falls Inside Two Other Numeric Values*

Description

Returns `TRUE` if `x` falls inside range defined by `ends` and `FALSE` otherwise. Also works for multiple sets of values and/or endpoints.

Usage

```
inside(x, ends, inclusive = TRUE)
```

Arguments

`x`	Numeric value or vector of numeric values.
`ends`	Numeric vector of length 2 specifying the endpoints for the interval, or a 2-column numeric matrix where each row specifies a pair of endpoints.
`inclusive`	Logical value indicating whether endpoints should be included.

Value

Logical value or vector.

Examples

```
# Check whether 2 is inside [0, 2.5]
inside(1, c(0, 2.5))

# Check whether 2 and 3 are inside (0, 3)
inside(c(2, 3), c(0, 3), inclusive = FALSE)

# Check whether 1 is inside [1, 2] and [3, 4]
```

```
inside(1, rbind(c(1, 2), c(3, 4)))
```

`interval_groups` *Split Continuous Variable into Equal-Width Groups*

Description

Splits a continuous variable into equal-width groups. Useful for assessing linearity in regression models.

Usage

```
interval_groups(x, groups = 5, ...)
```

Arguments

`x`	Numeric vector.
`groups`	Numeric value specifying number of groups to create.
`...`	Arguments to pass to `cut`.

Value

Factor variable.

See Also

`cut`

Examples

```
# Convert values from N(0, 1) into 6 equal-width groups
x <- rnorm(1000)
groups <- interval_groups(x, 6)
table(groups)

# Use interval_groups to detect non-linearity
set.seed(123)
x <- rnorm(1000)
y <- 1.5 + 1.25 * x + 0.25 * x^2 + rnorm(1000)
plot(tapply(y, interval_groups(x), mean))
```

`iterate` *Iterate Function Over All Combinations of User-Specified Inputs, Potentially Multiple Times*

Description

Same idea as **purrr**::*pmap*, but with some different functionality. It can runs all combinations of vector-valued arguments in `...` or the 1st set, 2nd set, and so forth, and multiple trials can be run for each scenario, which can be useful for simulations.

Usage

```
iterate(f, ..., all_combinations = TRUE, fix = NULL, trials = 1,
  varnames = NULL)
```

Arguments

`f`	A function.
`...`	Arguments to `f`, any of which can be vector-valued.
`all_combinations`	Logical value for whether to iterate over all combinations of arguments in `...`, or just use the first set of elements, then the second, and so on.
`fix`	List of arguments to `f` to hold fixed rather than loop over.
`trials`	Numeric value.
`varnames`	Character vector of names for values that `f` returns, to avoid generic labels (V1, V2, ...).

Value

Data frame.

Examples

```
# Define function to generate data from N(mu, sigsq) and perform t-test.
f <- function(n = 100, mu = 0, sigsq = 1, alpha = 0.05) {
  x <- rnorm(n = n, mean = mu, sd = sqrt(sigsq))
  fit <- t.test(x = x, alpha = alpha)
  return(list(t = fit$statistic, p = fit$p.value))
}

# Call f once for various sample sizes and means
f %>% iterate(n = c(100, 500), mu = c(0.1, 0.25))

# Run 100 trials for each scenario and calculate empirical power
f %>% iterate(n = c(100, 500), mu = c(0.1, 0.25), trials = 100) %>%
  group_by(n, mu) %>%
  summarise(mean(p < 0.05))
```

`list_override` *Add Elements of Second List to First List, Replacing Elements with Same Name*

Description

Adds each element of `list2` to `list1`, overriding any elements of the same name. Similar to `modifyList` function in **utils** package, but either list can be `NULL`. Useful for `do.call` statements, when you want to combine a list of default inputs with a list of user-specified inputs.

Usage

```
list_override(list1, list2)
```

Arguments

`list1`	Initial list that has some number of named elements. Can be `NULL` or an empty list.
`list2`	List with named elements that will be added to `list1`, replacing any elements with the same name. Can be `NULL` or an empty list.

Value

List containing the named elements initially in `list1` and not in `list2`, any additional named elements in `list2`, and any named elements in `list1` that were replaced by elements of the same name in `list2`.

Examples

```
# Create list that has default inputs to the plot function
list.defaults <- list(x = 1: 5, y = 1: 5, type = "l", lty = 1)

# Create list of user-specified inputs to the plot function
list.user <- list(main = "A Straight Line", lty = 2, lwd = 1.25)

# Combine the two lists into one, giving priority to list.user
list.combined <- list_override(list.defaults, list.user)

# Plot data using do.call
do.call(plot, list.combined)
```

logit_prob *Convert Logit to Probability*

Description

Defined as: `exp_x <- exp(x); out <- exp_x / (1 + exp_x)`. This 2-step approach is faster than `exp(x) / (1 + exp(x))` because the exponentials only have to be calculated once.

Usage

```
logit_prob(x)
```

Arguments

x	Numeric vector.

Value

Numeric vector.

lognormalreg *Linear Regression of log(Y) vs. Covariates with Y Potentially Subject to Multiplicative Lognormal Errors*

Description

Uses maximum likelihood to fit Y|**X** ~ Lognormal(beta_0 + **beta_x**^T **X**, sigsq). Y can be precisely measured or subject to multiplicative mean-1 lognormal errors, in which case replicates can be incorporated by specifying `y` as a list).

Usage

```
lognormalreg(y, x = NULL, merror = FALSE, estimate_var = TRUE,
  fix_posdef = FALSE, ...)
```

Arguments

y	Numeric vector or list.
x	Numeric vector or matrix. If `NULL`, model reduces to marginal lognormal model Y ~ Lognormal(beta_0, sigsq).
merror	Logical value for whether to model multiplicative lognormal measurement errors in Y.
estimate_var	Logical value for whether to return Hessian-based variance-covariance matrix.
fix_posdef	Logical value for whether to repeatedly reduce `integrate_tol_hessian` by factor of 5 and re-estimate Hessian to try to avoid non-positive definite variance-covariance matrix.
...	Additional arguments to pass to `nlminb`.

Value

List containing:

1. Numeric vector of parameter estimates.
2. Variance-covariance matrix (if `estimate_var = TRUE`).
3. Returned `nlminb` object from maximizing the log-likelihood function.
4. Akaike information criterion (AIC).

`logodds_graph` *Graph Log-Odds of Binary Variable Across A Grouping Variable*

Description

Creates plot showing sample log-odds of binary Y variable across levels of a grouping variable, with customizable error bars. Observations with missing values for `y` and/or `group` are dropped.

Usage

```
logodds_graph(y, group, error.bars = "none", alpha = 0.05,
  p.legend = "chi", plot.list = NULL, lines.list = NULL,
  axis.list = NULL, legend.list = NULL, ...)
```

Arguments

`y`	Vector of values for binary response variable. Must take on 2 values, but can be any type (e.g. numeric, character, factor, logical). Function plots log-odds of second value returned by `table(y)`.
`group`	Vector of values indicating what group each `y` observation belongs to. Function plots group levels across x-axis in same order as `table(group)`.
`error.bars`	Character string indicating what the error bars should represent. Possible values are `"exact.ci"` for exact 95% confidence interval based on binomial distribution, `"z.ci"` for approximate 95% confidence interval based on Z distribution, and `"none"` for no error bars.
`alpha`	Numeric value indicating what alpha should be set to for confidence intervals. Only used if `error.bars` is `"exact.ci"` or `"z.ci"`.
`p.legend`	Character string controlling what p-value is printed in a legend. Possible values are `"chi"` for Chi-square test of association, `"fisher"` for Fisher's exact test, and `"none"` for no legend at all.
`plot.list`	Optional list of inputs to pass to `plot` function.
`lines.list`	Optional list of inputs to pass to `lines` function.
`axis.list`	Optional list of inputs to pass to `axis` function.
`legend.list`	Optional list of inputs to pass to `legend` function.
`...`	Additional arguments to pass to `chisq.test` or `fisher.test` functions.

Value

Plot showing log-odds of y across levels of group.

---

max_n *Maximum of Numeric Values*

Description

Written in C++, this function tends to run faster than max for large numeric vectors/matrices.

Usage

```
max_n(x)
```

Arguments

x	Numeric vector.

Value

Numeric value.

Examples

```
# For large objects, max_n is faster than max
x <- rnorm(100000)
max(x) == max_n(x)
benchmark(max(x), max_n(x), replications = 1000)

# For smaller objects, max_n is slower than max
x <- rnorm(100)
max(x) == max_n(x)
benchmark(max(x), max_n(x), replications = 1000)
```

---

means_graph *Graph Means Across a Grouping Variable*

Description

Creates plot showing mean of Y variable across levels of a grouping variable, with customizable error bars. Observations with missing values for y and/or group are dropped.

Usage

```
means_graph(y, group, error.bars = "t.ci", alpha = 0.05,
  p.legend = TRUE, plot.list = NULL, lines.list = NULL,
  axis.list = NULL, legend.list = NULL, ...)
```

Arguments

y	Numeric vector of values for the continuous variable.
group	Vector of values indicating what group each y observation belongs to. Function plots group levels across x-axis in same order as table(group).
error.bars	Character string indicating what the error bars should represent. Possible values are "sd" for +/- one standard deviation, "se" for +/- one standard error, "t.ci" for 95% confidence interval based on t distribution, "z.ci" for 95% confidence interval based on Z distribution, and "none" for no error bars.
alpha	Numeric value indicating what alpha should be set to for confidence intervals. Only used if error.bars is "t.ci" or "z.ci".
p.legend	If TRUE, p-value (from t.test function if group has 2 levels, otherwise aov function) is printed in a legend.
plot.list	Optional list of inputs to pass to plot function.
lines.list	Optional list of inputs to pass to lines function.
axis.list	Optional list of inputs to pass to axis function.
legend.list	Optional list of inputs to pass to legend function.
...	Additional arguments to pass to t.test or aov.

Value

Plot showing mean of y across levels of group.

---

mean_i *Mean of Integer Values*

Description

Written in C++, this function runs faster than mean for large integer vectors/matrices.

Usage

```
mean_i(x)
```

Arguments

x	Integer vector or matrix.

Value

Numeric value.

Examples

```
# For very large integer objects, sum_i is faster than sum
x <- rpois(100000, lambda = 5)
mean(x) == mean_i(x)
benchmark(mean(x), mean_i(x), replications = 1000)

# For smaller integer objects, sum_i is slower than sum
x <- rpois(1000, lambda = 5)
mean(x) == mean_i(x)
benchmark(mean(x), mean_i(x), replications = 1000)
```

---

min_n *Minimum of Numeric Values*

Description

Written in C++, this function tends to run faster than `min` for large numeric vectors/matrices.

Usage

```
min_n(x)
```

Arguments

x	Numeric vector.

Value

Numeric value.

Examples

```
# For large objects, min_n is faster than min
x <- rnorm(100000)
min(x) == min_n(x)
benchmark(min(x), min_n(x), replications = 1000)

# For smaller objects, min_n is slower than min
x <- rnorm(100)
min(x) == min_n(x)
benchmark(min(x), min_n(x), replications = 20000)
```

mle_gamma *Maximum Likelihood Estimation for X[1], ..., X[n] ~ Gamma(alpha, beta)*

Description

Performs maximization via nlminb. alpha and beta correspond to the shape and scale (not shape and rate) parameters described in GammaDist.

Usage

```
mle_gamma(x, alpha = NULL, beta = NULL, estimate_var = FALSE, ...)
```

Arguments

x	Numeric vector.
alpha	Numeric value specifying known alpha.
beta	Numeric value specifying known beta.
estimate_var	Logical value for whether to return Hessian-based variance-covariance matrix.
...	Additional arguments to pass to nlminb.

Value

List containing:

1. Numeric vector of parameter estimates.
2. Variance-covariance matrix (if estimate_var = TRUE).
3. Returned nlminb object from maximizing the log-likelihood function.
4. Akaike information criterion (AIC).

Examples

```
# Generate 1,000 values from Gamma(0.5, 1) and estimate alpha and beta
set.seed(123)
x <- rgamma(1000, shape = 0.5, scale = 1)
mle_gamma(x)
```

`mle_gamma_lnorm` *Maximum Likelihood Estimation for X[1], ..., X[n] ~ Gamma(alpha, beta) Lognormal(mu, sigsq)*

Description

Each observation is assumed to be the product of a Gamma(alpha, beta) and Lognormal(mu, sigsq) random variable. Performs maximization via `nlminb`. alpha and beta correspond to the shape and scale (not shape and rate) parameters described in `GammaDist`, and mu and sigsq correspond to meanlog and sdlog^2 in `Lognormal`.

Usage

```
mle_gamma_lnorm(x, gamma_mean1 = FALSE, lnorm_mean1 = TRUE,
  integrate_tol = 1e-08, estimate_var = FALSE, ...)
```

Arguments

`x`	Numeric vector.
`gamma_mean1`	Whether to use restriction that the Gamma variable is mean-1.
`lnorm_mean1`	Whether to use restriction that the lognormal variable is mean-1.
`integrate_tol`	Numeric value specifying the `tol` input to `hcubature`.
`estimate_var`	Logical value for whether to return Hessian-based variance-covariance matrix.
`...`	Additional arguments to pass to `nlminb`.

Value

List containing:

1. Numeric vector of parameter estimates.
2. Variance-covariance matrix (if `estimate_var = TRUE`).
3. Returned `nlminb` object from maximizing the log-likelihood function.
4. Akaike information criterion (AIC).

Examples

```
# Generate 1,000 values from Gamma(0.5, 1) x Lognormal(-1.5/2, 1.5) and
# estimate parameters
## Not run:
set.seed(123)
x <- rgamma(1000, 0.5, 1) * rlnorm(1000, -1.5/2, sqrt(1.5))
mle_gamma_lnorm(x, control = list(trace = 1))

## End(Not run)
```

`mle_lnorm` *Maximum Likelihood Estimation for X[1], ..., X[n] ~ Lognormal(mu, sigsq)*

Description

Performs maximization via `nlminb`. mu and sigsq correspond to meanlog and sdlog^2 in `Lognormal`.

Usage

```
mle_lnorm(x, mu = NULL, sigsq = NULL, estimate_var = FALSE, ...)
```

Arguments

`x`	Numeric vector.
`mu`	Numeric value specifying known mu.
`sigsq`	Numeric value specifying known sigsq.
`estimate_var`	Logical value for whether to return Hessian-based variance-covariance matrix.
`...`	Additional arguments to pass to `nlminb`.

Value

List containing:

1. Numeric vector of parameter estimates.
2. Variance-covariance matrix (if `estimate_var = TRUE`).
3. Returned `nlminb` object from maximizing the log-likelihood function.
4. Akaike information criterion (AIC).

Examples

```
# Generate 1,000 values from Lognormal(0.5, 1) and estimate mu and sigsq
set.seed(123)
x <- rlnorm(1000, meanlog = 0.5, sdlog = sqrt(1))
mle_lnorm(x)
```

`mle_lnorm_lnorm` *Maximum Likelihood Estimation for X[1], ..., X[n] ~ Lognormal(mu1, sigsq1) Lognormal(mu2, sigsq2)*

Description

Each observation is assumed to be the product of a Lognormal(mu1, sigsq1) and Lognormal(mu2, sigsq2) random variable, with mu2 and sigsq2 known. Performs maximization via `nlminb`. mu and sigsq correspond to meanlog and sdlog^2 in `Lognormal`.

Usage

```
mle_lnorm_lnorm(x, mu2 = NULL, sigsq2 = NULL, estimate_var = FALSE,
  ...)
```

Arguments

`x`	Numeric vector.
`mu2`	Numeric value specifying known mu2.
`sigsq2`	Numeric value specifying known sigsq2.
`estimate_var`	Logical value for whether to return Hessian-based variance-covariance matrix.
`...`	Additional arguments to pass to `nlminb`.

Value

List containing:

1. Numeric vector of parameter estimates.
2. Variance-covariance matrix (if `estimate_var = TRUE`).
3. Returned `nlminb` object from maximizing the log-likelihood function.
4. Akaike information criterion (AIC).

Examples

```
# Generate 1,000 values from Lognormal(0.5, 1) x Lognormal(0.75, 1.5) and
# estimate parameters based on known mu and sigsq for one of them
set.seed(123)
x <- rlnorm(1000, 0.5, sqrt(1)) * rlnorm(1000, 0.75, sqrt(1.5))
mle_lnorm_lnorm(x, mu2 = 0.75, sigsq2 = 1.5)
```

moving_mean *Moving Averages*

Description

Calculates moving averages or maximum moving average. For optimal speed, use `integer = TRUE` if `x` is an integer vector and `integer = FALSE` otherwise.

Usage

```
moving_mean(x, window, integer = FALSE, max = FALSE)
```

Arguments

`x`	Integer or numeric vector.
`window`	Integer value specifying window length.
`integer`	Logical value for whether `x` is an integer vector.
`max`	Logical value for whether to return maximum moving average (as opposed to vector of moving averages).

Value

Numeric value or vector depending on `max`.

Examples

```
# 5-unit moving average for integer vector of length 10
x <- rpois(10, lambda = 3)
moving_mean(x, 5)
```

n_2t_equal *Calculate Per-Group Sample Size for Two-Sample Equal Variance T-Test*

Description

Same idea as `power.t.test`. Less flexible, but faster.

Usage

```
n_2t_equal(d, sigsq, alpha = 0.05, beta = 0.2)
```

Arguments

d	Numeric value specifying true difference in group means.
sigsq	Numeric value specifying the variance of observations.
alpha	Numeric value specifying type-1 error rate.
beta	Numeric value specifying type-2 error rate.

Value

Numeric value indicating per-group sample size, rounded up to the nearest whole number.

Examples

```
# Per-group sample size for 90% power to detect difference of 0.2 with
# sigsq = 1
n_2t_equal(d = 0.2, sigsq = 1, beta = 0.1)
```

---

n_2t_unequal *Calculate Per-Group Sample Size for Two-Sample Unequal Variance T-Test*

Description

Unequal variance version of n_2t_equal. Assumes an equal sample size for both groups, which is actually not optimal.

Usage

```
n_2t_unequal(d, sigsq1, sigsq2, alpha = 0.05, beta = 0.2)
```

Arguments

d	Numeric value specifying true difference in group means.
sigsq1, sigsq2	Numeric value specifying the variance of observations in each group.
alpha	Numeric value specifying type-1 error rate.
beta	Numeric value specifying type-2 error rate.

Value

Numeric value indicating per-group sample size, rounded up to the nearest whole number.

Examples

```
# Per-group sample size for 90% power to detect difference of 0.2 with
# sigsq's of 1 and 1.25
n_2t_unequal(d = 0.2, sigsq1 = 1, sigsq2 = 1.25, beta = 0.1)
```

odds_prob *Convert Odds to Probability*

Description

Defined simply as `log(x / (x + 1))`.

Usage

```
odds_prob(x)
```

Arguments

x	Numeric vector.

Value

Numeric vector.

plot_ll *Plot Log-Likelihood vs. Values of One Parameter*

Description

Generates plot of log-likelihood vs. one parameter of interest while other parameters are held fixed at certain values (e.g. MLEs). This is not a profile likelihood, and is mainly intended for use with a Shiny app.

Usage

```
plot_ll(start, objective, lower = -Inf, upper = Inf, xaxis_param = 1,
  xaxis_range = NULL, param_values = NULL, mles = NULL,
  return_info = FALSE)
```

Arguments

start	See nlminb.
objective	See nlminb.
lower	See nlminb.
upper	See nlminb.
xaxis_param	Integer value specifying which parameter should be plotted on the x-axis.
xaxis_range	Numeric vector specifying x-axis range over which to vary the parameter of interest. Only values with likelihood ratio > 0.01 are ultimately plotted.

param_values	Numeric vector of values to use for other parameters in model, in case you want an additional curve for log-likelihood function vs. parameter of interest at certain non-MLE values for other parameters. For example, if there are 3 parameters and xaxis_param = 2, you could set param_values = c(0, NA, 0).
mles	Numeric vector of previously obtained maximum likelihood estimates.
return_info	Logical value for whether to return the estimated MLEs and 99.99% confidence intervals for parameters rather than create the plot.

Details

Note that objective should be the negative log-likelihood function, since internal optimization uses (nlminb), which does minimization.

Value

Plot of log-likelihood vs. value of parameter of interest, generated by ggplot.

Examples

```
# Generate normal data, define log-likelihood function, and plot likelihood
set.seed(123)
x <- rnorm(100, mean = 0.5, sd = sqrt(0.25))
ll.f <- function(theta) {
  return(-sum(dnorm(x, log = TRUE, mean = theta[1], sd = sqrt(theta[2]))))
}
plot_ll(start = c(0, 1), objective = ll.f, lower = c(-Inf, 1e-6))
```

---

pooled_var *Pooled Sample Variance*

Description

Calculates pooled sample variance used in equal variance two-sample t-test.

Usage

```
pooled_var(x, y, integer = FALSE)
```

Arguments

x, y	Integer or numeric vectors.
integer	Logical value for whether x and y are integer vectors.

Value

Numeric value.

power_2t_equal *Calculate Power for Two-Sample Equal Variance T-Test*

Description

Same idea as power.t.test. Less flexible, but faster.

Usage

```
power_2t_equal(n = 100, d, sigsq, alpha = 0.05)
```

Arguments

n	Numeric value specifying per-group sample size.
d	Numeric value specifying true difference in group means. Should be positive.
sigsq	Numeric value specifying the variance of observations.
alpha	Numeric value specifying type-1 error rate.

Value

Numeric value.

Examples

```
# Power to detect difference of 0.2 with 100 subjects per group and sigsq = 1
power_2t_equal(n = 100, d = 0.2, sigsq = 1)
```

power_2t_unequal *Calculate Power for Two-Sample Unequal Variance T-Test*

Description

Unequal variance version of power_2t_equal. Assumes an equal sample size for both groups, which is actually not optimal.

Usage

```
power_2t_unequal(n = 100, d, sigsq1, sigsq2, alpha = 0.05)
```

Arguments

n	Numeric value specifying per-group sample size.
d	Numeric value specifying true difference in group means. Should be positive.
sigsq1, sigsq2	Numeric value specifying the variance of observations in each group.
alpha	Numeric value specifying type-1 error rate.

Value

Numeric value.

Examples

```
# Power to detect difference of 0.2 with 100 subjects per group and sigsq's
# of 1 and 1.25
power_2t_unequal(n = 100, d = 0.2, sigsq1 = 1, sigsq2 = 1.25)
```

---

prob_logit *Convert Probability to Logit*

Description

Defined simply as `log(x / (1 - x))`.

Usage

```
prob_logit(x)
```

Arguments

x	Numeric vector.

Value

Numeric vector.

---

prob_odds *Convert Probability to Odds*

Description

Defined simply as `x / (1 - x)`.

Usage

```
prob_odds(x)
```

Arguments

x	Numeric vector.

Value

Numeric vector.

quant_groups *Split Continuous Variable into Quantile Groups*

Description

Splits a continuous variable into quantiles groups. Basically combines `quantile` and `cut` into a single function. Note that `create_qgroups` will likely supersede this function in future versions of **dvmisc**.

Usage

```
quant_groups(x, groups = 4, probs = NULL, quantile.list = NULL,
  cut.list = NULL)
```

Arguments

`x`	Numeric vector.
`groups`	Numeric value specifying number of quantile groups.
`probs`	Numeric vector specifying probabilities.
`quantile.list`	Arguments to pass to `quantile`.
`cut.list`	Arguments to pass to `cut`.

Value

Factor variable.

Examples

```
# Convert values from N(0, 1) into quintiles (i.e. 5 groups)
x <- rnorm(1000)
groups <- quant_groups(x, 5)
table(groups)
```

quant_groups_svy *Split Continuous Variable into Quantile Groups (Survey Version)*

Description

Complex survey version of `quant_groups`. Speeds up process of creating quantile groups based on survey weighted percentiles.

Usage

```
quant_groups_svy(x, by = NULL, groups = 4, probs = NULL, design)
```

Arguments

x	Formula, e.g. ~varname.
by	Formula, e.g. ~varname.
groups	Numeric value specifying number of quantile groups.
probs	Numeric vector.
design	A svydesign or svrepdesign object.

Value

Factor variable.

---

reverse_cut *Reverse Cut*

Description

Convenience function to get reversed factor levels from cut. Currently requires specifying breaks as vector of cutpoints rather than number of desired intervals.

Usage

```
reverse_cut(x, breaks, include.lowest = FALSE, right = TRUE, ...)
```

Arguments

x, breaks, include.lowest, right	See cut. specifying number of intervals is not currently supported).
...	Arguments to pass to cut.

Value

Factor variable.

Examples

```
# In mtcars dataset, create 3 mpg groups
table(cut(mtcars$mpg, breaks = c(-Inf, 15, 20, Inf)))

# Repeat with reverse_cut to get factor levels ordered from high to low
table(reverse_cut(mtcars$mpg, breaks = c(Inf, 20, 15, -Inf)))

# You can specify breaks from low to high, but then include.lowest and right
# arguments get confusing
table(reverse_cut(mtcars$mpg, breaks = c(-Inf, 15, 20, Inf), right = TRUE))
```

sliding_cor *Moving Correlations as Short Vector Slides Across Long Vector*

Description

Uses C++ code for efficiency.

Usage

```
sliding_cor(short, long)
```

Arguments

short	Numeric vector.
long	Numeric vector.

Value

Numeric vector.

Examples

```
short <- rnorm(4)
long <- rnorm(10)
sliding_cor(short, long)
```

sliding_cov *Moving Covariance as Short Vector Slides Across Long Vector*

Description

Uses C++ code for efficiency.

Usage

```
sliding_cov(short, long)
```

Arguments

short	Numeric vector.
long	Numeric vector.

Value

Numeric vector.

Examples

```
short <- rnorm(4)
long <- rnorm(10)
sliding_cov(short, long)
```

`sumsim` *Summarize Simulation Results*

Description

Creates table summarizing results of statistical simulations, providing common metrics of performance like mean bias, standard deviation, mean standard error, mean squared error, and confidence interval coverage.

Usage

```
sumsim(estimates, ses = NULL, truth = NULL, theta_0 = 0,
  statistics = c("mean_bias", "sd", "mean_se", "mse", "coverage"),
  alpha = 0.05, digits = 3, listwise_deletion = TRUE)
```

Arguments

`estimates`	Numeric matrix where each column gives the point estimates for a particular method across multiple trials.
`ses`	Numeric matrix where each column gives the standard errors for a particular method across multiple trials.
`truth`	Numeric value specifying the true value of the parameter being estimated.
`theta_0`	Numeric value specifying null value for hypothesis test `H_0: theta = theta_0`. Only used for calculating empirical power.
`statistics`	Numeric vector specifying which performance metrics should be calculated. Possible values are `"n"` for number of trials, `"mean"`, `"median"`, `"mean_bias"`, `"median_bias"`, `"sd"`, `"iqr"`, `"mean_se"` (for mean standard error), `"mse"` (for mean squared error), `"coverage"` (for confidence interval coverage), `"ci_width"` for median confidence interval width, and `"power"` for empirical power.
`alpha`	Numeric value specifying alpha for confidence interval. Set to `0.05` for the usual 95% CI, `0.1` for a 90% CI, and so forth.
`digits`	Numeric value or vector specifying the number of decimal places to include.
`listwise_deletion`	Logical value for whether to remove trials in which any of the estimators have missing values.

Value

Numeric matrix.

Examples

```
# For X ~ N(mu, sigma^2), the MLE for sigma^2 is the sample variance with n
# in the denominator, but the unbiased version with (n - 1) is typically used
# for its unbiasedness. Compare these estimators in 1,000 trials with n = 25.
MLE <- c()
Unbiased <- c()
for (ii in 1: 1000) {
   x <- rnorm(n = 25)
   MLE[ii] <- sum((x - mean(x))^2) / 25
   Unbiased[ii] <- sum((x - mean(x))^2) / 24
 }
sumsim(estimates = cbind(MLE, Unbiased), truth = 1)
```

---

sum_i *Sum of Integer Values*

Description

Written in C++, this function runs faster than `sum` for large integer vectors/matrices.

Usage

```
sum_i(x)
```

Arguments

x	Integer vector or matrix.

Value

Numeric value.

Examples

```
# For very large integer objects, sum_i is faster than sum
x <- rpois(100000, lambda = 5)
sum(x) == sum_i(x)
benchmark(sum(x), sum_i(x), replications = 1000)

# For smaller integer objects, sum_i is slower than sum
x <- rpois(1000, lambda = 5)
sum(x) == sum_i(x)
benchmark(sum(x), sum_i(x), replications = 1000)
```

trim *Trim Tail Values off of a Vector*

Description

Returns input vector with tail values trimmed off of it. User can specify tail probability to trim or lower and upper cutpoints for values to retain.

Usage

```
trim(x, p = NULL, tails = "both", cutpoints = NULL,
  keep.edge = TRUE)
```

Arguments

x	Numeric vector.
p	Numeric value giving tail probability to trim from x. Can leave as NULL if you specify cutpoints.
tails	Numeric value indicating which tail should be trimmed. Possible values are "both", "lower", and "upper".
cutpoints	Numeric vector indicating what range of values should be retained. For example, set to c(0, 1) to trim all values below 0 or greater than 1. Can leave as NULL if you specify p.
keep.edge	Logical value indicating whether values in x that are on the edge of being trimmed (i.e. equal to one of the endpoints) should be retained.

Value

Numeric vector.

See Also

inside

Examples

```
# Generate data from N(0, 1) and then trim the lower and upper 1\%
x <- rnorm(1000)
y <- trim(x, p = 0.01)

# Generate data from N(0, 1) and then trim values outside of (-1.5, 1.5)
x <- rnorm(100000)
y <- trim(x, cutpoints = c(-1.5, 1.5))
```

truerange *Range of a Vector (Not Min/Max!)*

Description

The base R function `range` returns the minimum and maximum of a vector, but the "range" is actually defined as the difference between the minimum and maximum. This function calculates the actual range. It is equivalent to the base R code `diff(range(x))`, but a bit simpler and much faster.

Usage

```
truerange(x, integer = FALSE)
```

Arguments

`x`	Integer or numeric vector.
`integer`	Logical value for whether `x` is an integer vector.

Value

Integer or numeric value.

Examples

```
# truerange vs. diff(range()) for integer vector
x <- rpois(1000, lambda = 5)
all.equal(diff(range(x)), truerange(x, TRUE))
benchmark(diff(range(x)), truerange(x, TRUE), replications = 2000)

# truerange vs. diff(range()) for numeric vector
x <- rnorm(1000)
all.equal(diff(range(x)), truerange(x))
benchmark(diff(range(x)), truerange(x), replications = 2000)
```

which.max2 *Return Index of (First) Maximum of a Vector*

Description

Returns index of maximum for vectors and index or (row, column) position for matrices. For optimal speed, use `integer = TRUE` if `x` is an integer vector/matrix and `integer = FALSE` otherwise. Typically faster than `which.max` for matrices and for large vectors.

Usage

```
which.max2(x, arr.ind = FALSE, integer = FALSE)
```

Arguments

x	Integer or numeric vector/matrix.
arr.ind	Logical value for whether to return (row, col) position rather than vector position, if x is a matrix.
integer	Logical value for whether x is an integer vector/matrix.

Value

Numeric value.

Examples

```
# which.max2 vs. which.max for integer vector
x <- rpois(10000, lambda = 5)
all.equal(which.max(x), which.max2(x, integer = TRUE))
benchmark(which.max(x), which.max2(x, integer = TRUE), replications = 10000)

# which.max2 vs. which.max for numeric vector
x <- rnorm(10000)
all.equal(which.max(x), which.max2(x))
benchmark(which.max(x), which.max2(x), replications = 10000)
```

---

which.min2 *Return Index of (First) Minimum of a Vector*

Description

Returns index of minimum for vectors and index or (row, column) position for matrices. For optimal speed, use integer = TRUE if x is an integer vector/matrix and integer = FALSE otherwise. Typically faster than which.min for matrices and for large vectors.

Usage

```
which.min2(x, arr.ind = FALSE, integer = FALSE)
```

Arguments

x	Integer or numeric vector/matrix.
arr.ind	Logical value for whether to return (row, col) position rather than vector position, if x is a matrix.
integer	Logical value for whether x is an integer vector/matrix.

Value

Numeric value.

Examples

```
# which.min2 vs. which.min for integer vector
x <- rpois(10000, lambda = 10)
all.equal(which.min(x), which.min2(x, integer = TRUE))
benchmark(which.min(x), which.min2(x, integer = TRUE), replications = 10000)

# which.min2 vs. which.min for numeric vector
x <- rnorm(10000)
all.equal(which.min(x), which.min2(x))
benchmark(which.min(x), which.min2(x), replications = 10000)
```

---

which_max_im *Return (Row, Column) Index of (First) Maximum of an Integer Matrix*

Description

Written in C++, this function tends to run much faster than the equivalent (if maximum is unique) base R solution `which(x == max(x), arr.ind = TRUE)`.

Usage

```
which_max_im(x)
```

Arguments

`x`	Integer matrix.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_max_nv` for numeric vector.
`which_max_iv` for integer vector.
`which_max_nm` for numeric matrix.
`which_max_im` for integer matrix.

Value

Integer vector.

Examples

```
# which_max_im is typically much faster than
# which(x == max(x), arr.ind = TRUE)
x <- matrix(rpois(100, lambda = 15), ncol = 10)
all(which(x == max(x), arr.ind = TRUE) == which_max_im(x))
benchmark(which(x == max(x), arr.ind = TRUE), which_max_im(x),
          replications = 5000)
```

---

which_max_iv *Return Index of (First) Maximum of an Integer Vector*

Description

Written in C++, this function tends to run faster than `which.max` for large integer vectors.

Usage

```
which_max_iv(x)
```

Arguments

x	Integer vector.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_max_nv` for numeric vector.
`which_max_iv` for integer vector.
`which_max_nm` for numeric matrix.
`which_max_im` for integer matrix.

Value

Integer value.

Examples

```
# For long vectors, which_max_iv is faster than which.max
x <- rpois(10000, lambda = 15)
which.max(x) == which_max_iv(x)
benchmark(which.max(x), which_max_iv(x), replications = 5000)

# For shorter vectors, which_max_iv is slower than which.max
x <- rpois(100, lambda = 15)
which.max(x) == which_max_iv(x)
benchmark(which.max(x), which_max_iv(x), replications = 20000)
```

`which_max_nm` *Return (Row, Column) Index of (First) Maximum of a Numeric Matrix*

Description

Written in C++, this function tends to run much faster than the equivalent (if maximum is unique) base R solution `which(x == max(x), arr.ind = TRUE)`.

Usage

```
which_max_nm(x)
```

Arguments

`x`	Numeric matrix.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_max_nv` for numeric vector.
`which_max_iv` for integer vector.
`which_max_nm` for numeric matrix.
`which_max_im` for integer matrix.

Value

Integer vector.

Examples

```
# which_max_nm is typically much faster than
# which(x == max(x), arr.ind = TRUE)
x <- matrix(rnorm(100), ncol = 10)
all(which(x == max(x), arr.ind = TRUE) == which_max_nm(x))
benchmark(which(x == max(x), arr.ind = TRUE), which_max_nm(x),
          replications = 5000)
```

`which_max_nv` *Return Index of (First) Maximum of a Numeric Vector*

Description

Written in C++, this function tends to run faster than `which.max` for large numeric vectors.

Usage

```
which_max_nv(x)
```

Arguments

`x`	Numeric vector.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_max_nv` for numeric vector.
`which_max_iv` for integer vector.
`which_max_nm` for numeric matrix.
`which_max_im` for integer matrix.

Value

Integer value.

Examples

```
# For long vectors, which_max_nv is faster than which.max
x <- rnorm(100000)
which.max(x) == which_max_nv(x)
benchmark(which.max(x), which_max_nv(x), replications = 500)

# For shorter vectors, which_max_nv is slower than which.max
x <- rnorm(100)
which.max(x) == which_max_nv(x)
benchmark(which.max(x), which_max_nv(x), replications = 10000)
```

`which_min_im` *Return (Row, Column) Index of (First) Minimum of an Integer Matrix*

Description

Written in C++, this function tends to run much faster than the equivalent (if minimum is unique) base R solution `which(x == min(x), arr.ind = TRUE)`.

Usage

```
which_min_im(x)
```

Arguments

`x`	Integer matrix.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_min_nv` for numeric vector.
`which_min_iv` for integer vector.
`which_min_nm` for numeric matrix.
`which_min_im` for integer matrix.

Value

Integer vector.

Examples

```
# which_min_im is typically much faster than
# which(x == min(x), arr.ind = TRUE)
x <- matrix(rpois(100, lambda = 10), ncol = 10)
all(which(x == min(x), arr.ind = TRUE) == which_min_im(x))
benchmark(which(x == min(x), arr.ind = TRUE), which_min_im(x),
          replications = 5000)
```

`which_min_iv` *Return Index of (First) Minimum of an Integer Vector*

Description

Written in C++, this function tends to run faster than `which.min` for large integer vectors.

Usage

```
which_min_iv(x)
```

Arguments

`x`	Integer vector.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_min_nv` for numeric vector.
`which_min_iv` for integer vector.
`which_min_nm` for numeric matrix.
`which_min_im` for integer matrix.

Value

Integer value.

Examples

```
# For long vectors, which_min_iv is faster than which.min
x <- rpois(10000, lambda = 15)
which.min(x) == which_min_iv(x)
benchmark(which.min(x), which_min_iv(x), replications = 5000)

# For shorter vectors, which_min_iv is slower than which.min
x <- rpois(100, lambda = 15)
which.min(x) == which_min_iv(x)
benchmark(which.min(x), which_min_iv(x), replications = 20000)
```

which_min_nm *Return (Row, Column) Index of (First) Minimum of a Numeric Matrix*

Description

Written in C++, this function tends to run much faster than the equivalent (if minimum is unique) base R solution `which(x == min(x), arr.ind = TRUE)`.

Usage

```
which_min_nm(x)
```

Arguments

x	Numeric matrix.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_min_nv` for numeric vector.
`which_min_iv` for integer vector.
`which_min_nm` for numeric matrix.
`which_min_im` for integer matrix.

Value

Integer vector.

Examples

```
# which_min_nm is typically much faster than
# which(x == min(x), arr.ind = TRUE)
x <- matrix(rnorm(100), ncol = 10)
all(which(x == min(x), arr.ind = TRUE) == which_min_nm(x))
benchmark(which(x == min(x), arr.ind = TRUE), which_min_nm(x),
          replications = 5000)
```

which_min_nv *Return Index of (First) Minimum of a Numeric Vector*

Description

Written in C++, this function tends to run faster than `which.min` for large numeric vectors.

Usage

```
which_min_nv(x)
```

Arguments

x	Numeric vector.

Details

For optimal speed, choose the version of this function that matches the class of your `x`:

`which_min_nv` for numeric vector.
`which_min_iv` for integer vector.
`which_min_nm` for numeric matrix.
`which_min_im` for integer matrix.

Value

Integer value.

Examples

```
# For long vectors, which_min_nv is faster than which.min
x <- rnorm(100000)
which.min(x) == which_min_nv(x)
benchmark(which.min(x), which_min_nv(x), replications = 1000)

# For shorter vectors, which_min_nv is slower than which.min
x <- rnorm(100)
which.min(x) == which_min_nv(x)
benchmark(which.min(x), which_min_nv(x), replications = 10000)
```

Index

abline, *9*
aov, *23*
arrows, *9*
axis, *9*, *15*, *21*, *23*

bmi3, 3
bmi4, 3

chisq.test, *21*
clean_glm, 4
cleancut, 4
create_qgroups, 5, *6*, *35*
create_qgroups_svy, 6
cut, *4–7*, *17*, *35, 36*
cut_decreasing, 7

Distributions, *14*
do.call, *19*
dots_bars, 8
dvmisc, 10
dvmisc-package *(*dvmisc*)*, 10

expand.grid, *11*
expand_grid, 11

fisher.test, *21*
formatp, *5*

GammaDist, *11*, *25, 26*
gammareg, 11
get_mse, 12
ggplot, *32*
glm, *4, 5*, *12, 13*

hcubature, *12*, *26*
head, *13*
headtail, 13
hist, *14, 15*
histo, 14

inside, 16, *40*
interaction, *6, 7*
interval_groups, 17
iterate, 18

kable, *4, 5*

legend, *9*, *15*, *21*, *23*
lines, *21*, *23*
list_override, 19
lm, *12, 13*
logit_prob, 20
Lognormal, *26–28*
lognormalreg, 20
logodds_graph, 21

max_n, 22
mean, *23*
mean_i, 23
means_graph, 22
min_n, 24
mle_gamma, 25
mle_gamma_lnorm, 26
mle_lnorm, 27
mle_lnorm_lnorm, 28
moving_mean, 29
mutate, *5*

n_2t_equal, 29, *30*
n_2t_unequal, 30
nlminb, *12*, *20, 21*, *25–28*, *31, 32*

odds_prob, 31

plot, *9*, *21*, *23*
plot_ll, 31
points, *9*, *15*
pooled_var, 32
power.t.test, *29*, *33*
power_2t_equal, *33*, 33
power_2t_unequal, 33
prob_logit, 34

prob_odds, 34

quant_groups, *35*, 35
quant_groups_svy, 35
quantile, *5, 6*, *35*

range, *41*
reverse_cut, 36

sliding_cor, 37
sliding_cov, 37
sum, *39*
sum_i, 39
sumsim, 38
svyquantile, *7*

t.test, *23*
tail, *13*
transmute, *5*
trim, 40
truerange, 41

which.max, *41*
which.max2, 41
which.min, *42*, *48*, *50*
which.min2, 42
which_max_im, *43*, 43, *44–46*
which_max_iv, *43, 44*, 44, *45, 46*
which_max_nm, *43–45*, 45, *46*
which_max_nv, *43–46*, 46
which_min_im, *47*, 47, *48–50*
which_min_iv, *47, 48*, 48, *49, 50*
which_min_nm, *47–49*, 49, *50*
which_min_nv, *47–50*, 50