Package ‘exams.forge’

March 5, 2026

Type Package

Title Support for Compiling Examination Tasks using the 'exams' Package

Version 1.0.13

Date 2026-03-04

Description The main aim is to further facilitate the creation of exercises based on the package 'exams' by Grün, B., and Zeileis, A. (2009) <doi:10.18637/jss.v029.i10>. Creating effective student exercises
involves challenges such as creating appropriate data sets and ensuring access to intermediate values
for accurate explanation of solutions. The functionality includes the generation of univariate and bivariate data including simple time series, functions for theoretical distributions and their approximation,
statistical and mathematical calculations for tasks in basic statistics courses as well as general tasks such as string manipulation, LaTeX/HTML formatting and the editing of XML task files for 'Moodle'.

License GPL-3

LazyData true

Depends R (>= 3.5.0), tools, polynom

Imports base64enc, httr, knitr, MASS, methods, magrittr, rjson, rstudioapi, spelling, stringdist, stringr, stranslate, tinytex, utils, xml2, xtable, yaml

Encoding UTF-8

VignetteBuilder knitr

Suggests rmarkdown, exams, extraDistr, hunspell

RoxygenNote 7.3.3

NeedsCompilation no

Author Sigbert Klinke [aut, cre] (ORCID: <`https://orcid.org/0000-0003-3337-1863`>), Kleio Chrysopoulou Tseva [ctb]

Maintainer Sigbert Klinke <`sigbert@hu-berlin.de`>

Repository CRAN

Date/Publication 2026-03-05 10:50:02 UTC

Contents

`exams.forge-package` *exams.forge: A brief description the package*

Description

The `exams.forge` package was created with the main goal of "forging" exam tasks in combination with the `exams` package, and it includes additional functions to simplify the creation of Moodle exercises. The package features various functions categorized into seven groups based on their characteristics. These categories are named: Data Conversion and Modification, Statistical Analysis, Mathematical Computations, Exercise Generation, String Manipulation, LaTeX and HTML Functions, and General Purpose Functions.

Details

This package is designed for educators who need to develop examination materials in the field of statistics, particularly for introductory courses like Statistics I and II, using the R programming language. The aim is to streamline the process of creating a large number of assessment items, enabling instructors to focus on improving the quality of the tasks themselves.

We would like to acknowledge the support provided by the Multimedia Funding Program. Their assistance has been invaluable to our project, and we extend our sincere gratitude for their contributions.

Features of the package

- Feature 1: `exams.forge` simplifies the generation of examination tasks by providing tools to create a diverse array of statistical exercises, ensuring unique problem sets for each student.
- Feature 2: It includes functions for precise data conversion, statistical analysis, and mathematical computations, enhancing the accuracy and relevance of generated exercises.
- Feature 3: The package supports multi-format rendering, allowing the seamless creation of LaTeX and HTML documents suitable for various educational platforms, such as Moodle.

Functions

Examples of functions included in the package:

- `ts_data`: Creates a univariate time series by combining elements of a linear or exponential trend, additive or multiplicative seasonal adjustment, and white noise. The resulting time series is structured as a `ts_data object`, allowing for further analysis and exploration.
- `lmatrix`: Creates a LaTeX or HTML representation of a matrix. This function is useful for integrating well-formatted matrices into LaTeX or HTML documents.
- `as_obs`: Creates a string representing observations with optional sorting and LaTeX formatting, useful for generating readable data representations in educational materials.

Usage

Example usage of the package and its functions.

```
library(exams.forge) # Generate a time series with specified parameters ts_eg <- ts_data(end
= 20, trend = TRUE, trend.coeff = c(1, 0.5), season = TRUE, season.coeff = c(0.2, 0.1),
error = TRUE, error.coeff = 0.1, digits = 2) print(ts_eg)
```

```
# Create a matrix mx_data <- matrix(1:6, ncol = 2) # Generate a LaTeX representation of
the matrix with tooltip eg_matrix <- lmatrix( m = mx_data, title = "Example LaTeX Matrix",
fmt = " byrow = TRUE, tooltip = "Die Tabelle hat cat(eg_matrix)
```

```
# Create a string representation of observations observations <- c(10, 20, 30, 40, 50)
observation_string <- as_obs(observations, last = " and ") print(observation_string)
```

Installation

To install this package please use the following command: `install.packages("exams.forge")`

Author(s)

Sigbert Klinke, Affiliation: Humboldt University of Berlin, School of Business and Economics, Chair of Statistics.

Maintainer

Sigbert Klinke <`sigbert@wiwi.hu-berlin.de`>

License

Gnu General Public License 3.

Author(s)

Maintainer: Sigbert Klinke <`sigbert@hu-berlin.de`> (ORCID)

Other contributors:

- Kleio Chrysopoulou Tseva <`kleio.chrysopoulou.tseva@hu-berlin.de`> [contributor]

`add_data` *Add Random Data to the Ends of a Vector*

Description

Generates and appends additional random values to the left and/or right ends of a numeric vector `x`. The range from which these values are drawn is determined by a specified "box" and a scaling factor.

Usage

```
add_data(x, box, n = c(0, 1), range = c(0, 1))
```

Arguments

`x`	`numeric` Data vector to which new values will be added.
`box`	`character` or `numeric` Defines the base range ("box") used for value generation. See **Details**.
`n`	`numeric` Number of values to add to each side of `x`. Defaults to `c(0, 1)`.
`range`	`numeric` Multiplicative factors of box length for determining value generation intervals. Defaults to `c(0, 1)`.

Details

The "box" defines the central range of the data and can be:

- `"boxplot"` — uses the 25th and 75th percentiles (`quantile(x, c(0.25, 0.75), na.rm = TRUE)`).
- `"range"` — uses the full range of the data (`range(x, na.rm = TRUE)`).
- A numeric vector of length two — used directly as the box boundaries.

The **box length** is the distance between the lower and upper box boundaries. The `range` parameter specifies how far left or right new values can be drawn, as a multiple of the box length.

For the left side, values are drawn uniformly from:

$$[\text{box}[1] - \text{range}[2] \times \text{box length},\ \text{box}[1] - \text{range}[1] \times \text{box length}]$$

For the right side, values are drawn uniformly from:

$$[\text{box}[2] + \text{range}[1] \times \text{box length},\ \text{box}[2] + \text{range}[2] \times \text{box length}]$$

The `n` parameter controls how many values are added:

- Single number — adds that many values to the right side only.
- Length-two vector — `n[1]` values to the left, `n[2]` values to the right.

Value

A numeric vector containing the original values from `x` plus the newly generated values.

Examples

```
x <- rnorm(8)

# Add one value to the right
add_data(x, "box", range = 1.5)

# Add one value to the right using data range
add_data(x, "range", range = 0.1)

# Add one value to the right, larger possible range
add_data(x, "box", range = c(1.5, 3))

# Add two values to the right
add_data(x, "range", n = 2, range = 0.1)

# Add two values to the left and three to the right
add_data(x, "range", n = c(2, 3), range = 0.1)
```

`affix` *Quote, Bracket, and Prefix/Suffix String Manipulation*

Description

A set of helper functions for adding or removing specific prefixes and/or suffixes to character vectors. This is useful for formatting strings in mathematical, XML, or other structured text contexts.

- `affix()` – Add any prefix and/or suffix to each element of a character vector; alias: `add_affix()`.
- `math()` – Add a dollar sign (`$`) to both ends of each element (LaTeX-style math); aliases: `add_math()`, `lmath()`.
- `bracket()` – Wrap each element in parentheses; aliases: `add_bracket()`, `brkt()`.
- `cdata()` – Wrap each element in `<![CDATA[ ... ]]>` (XML CDATA section); alias: `add_cdata()`.
- `unaffix()` – Remove a given prefix and/or suffix from each element; alias: `remove_affix()`.
- `unquote()` – Remove surrounding double quotes from each element; alias: `remove_quotes()`.
- `uncdata()` – Remove a surrounding CDATA section from each element.; alias: `remove_cdata()`.

Usage

```
affix(txt, prefix = "", suffix = "")

math(txt)

bracket(txt)

unaffix(txt, prefix = "", suffix = "")

unquote(txt)

uncdata(txt)

cdata(txt)

add_affix(txt, prefix = "", suffix = "")

add_cdata(txt)

add_math(txt)

lmath(txt)

add_bracket(txt)

brkt(txt)
```

```
remove_affix(txt, prefix = "", suffix = "")

remove_quotes(txt)

remove_cdata(txt)
```

Arguments

`txt`	`character` vector to modify.
`prefix`	`character(1)` Prefix to add or remove. Default is `""`.
`suffix`	`character(1)` Suffix to add or remove. Default is `""`.

Value

A modified `character` vector of the same length as `txt`.

Examples

```
x <- c("alpha", "beta", "gamma")

# Add a prefix and suffix
affix(x, "[", "]")
#> [1] "[alpha]" "[beta]" "[gamma]"

# Wrap with LaTeX math delimiters
math(x)
#> [1] "$alpha$" "$beta$" "$gamma$"

# Remove quotes
quoted <- c('"a"', '"b"')
unquote(quoted)
#> [1] "a" "b"

# Wrap and unwrap CDATA
cdata_x <- cdata("text")
uncdata(cdata_x)
```

`all_different` *Difference Testing Functions*

Description

These functions determine whether numeric values are sufficiently different from each other based on a specified tolerance.

- `all_different`: Checks if all differences between entries in a numeric object exceed a given tolerance.
- `values_different`: Adds candidate values to a set of initial values only if they are sufficiently different from all existing values.

Usage

```
all_different(obj, tol)

values_different(values, candidates, tol = 0.1)
```

Arguments

`obj`	Numeric vector, matrix, or data frame to test for differences. Non-numeric inputs will be coerced to numeric if possible.
`tol`	Numeric scalar specifying the minimum allowable difference between values.
`values`	Numeric vector of initial values.
`candidates`	Numeric vector of candidate values to add if sufficiently different.

Value

- `all_different`: Logical (`TRUE` if all differences exceed `tol`, `FALSE` otherwise).
- `values_different`: Numeric vector with original `values` plus any `candidates` that meet the difference criterion.

Examples

```
# Check if all values are sufficiently different
x <- runif(10)   # 10 random values between 0 and 1
all_different(x, tol = 0.01)
all_different(x, tol = 0.5)

# Add sufficiently different candidate values
starting_values <- c(0.1, 0.5, 0.9)
candidates <- c(0.15, 0.4, 0.8, 1.2)
values_different(starting_values, candidates, tol = 0.2)
```

`assoc` *Association and Correlation Measures*

Description

Compute association and correlation measures for categorical and ordinal data.

The following measures are implemented:

- **nom.cc**: Corrected contingency coefficient for nominal data.
- **nom.cramer**: Cramer’s V (or Phi) for nominal data.
- **ord.spearman**: Spearman’s rank correlation for ordinal data.
- **ord.kendall**: Kendall’s rank correlation for ordinal data.

Usage

```
nom.cc(tab, correct = FALSE)

nom.cramer(tab, ...)

ord.spearman(tab, ...)

ord.kendall(tab, ...)

cc_coef(tab, correct = FALSE)

cramer_vf(tab, ...)

cramer_coef(tab, ...)

kendall_corr(tab, ...)

spearman_corr(tab, ...)

rs_corr(tab, ...)
```

Arguments

`tab`	A contingency table (matrix or table) with absolute frequencies.
`correct`	Logical, whether to apply a correction (default: `FALSE`). Only used for `nom.cc`.
`...`	Additional parameters passed to correlation functions.

Details

These functions provide common measures of association:

- Nominal data: `nom.cc`, `nom.cramer`.
- Ordinal data: `ord.spearman`, `ord.kendall`.

Value

A numeric value representing the association or correlation measure.

Examples

```
# Create a random contingency table
tab <- matrix(round(10 * runif(15)), ncol = 5)

# Nominal association
nom.cc(tab)
nom.cc(tab, correct = TRUE)
nom.cramer(tab)

# Ordinal correlation
```

```
ord.spearman(tab)
ord.kendall(tab)

# Using aliases
cc_coef(tab)
cramer_vf(tab)
spearman_corr(tab)
kendall_corr(tab)
rs_corr(tab)
```

`assoc_data` *Optimize Frequency Table for a Target Association*

Description

Reorders the entries of a frequency table to approximate a given target association or correlation.

The reordering preserves the marginal frequencies of the table. Note that the target association may not always be achievable, especially for extreme values (e.g., +1, -1, or values near these limits).

Usage

```
assoc_data(
  tab,
  zero = FALSE,
  FUN = nom.cc,
  target = NA,
  tol = 0.001,
  maxit = 500,
  ...
)

reorder_association_data(
  tab,
  zero = FALSE,
  FUN = nom.cc,
  target = NA,
  tol = 0.001,
  maxit = 500,
  ...
)

dassoc(
  tab,
  zero = FALSE,
  FUN = nom.cc,
  target = NA,
  tol = 0.001,
```

```
    maxit = 500,
    ...
  )
```

Arguments

`tab`	table A contingency table of absolute frequencies.
`zero`	logical Whether zeros are allowed in the resulting table (default: `FALSE`).
`FUN`	function A function that computes the association or correlation from a frequency table (default: `nom.cc`).
`target`	numeric Desired association or correlation value (default: `NA`, which returns the original table).
`tol`	numeric Maximum allowed deviation between the achieved and target association (default: `0.001`).
`maxit`	integer Maximum number of iterations to reach the target (default: `500`).
`...`	Additional parameters passed to `FUN`.

Details

The function attempts to reorder the table entries to reach the target association. If the target is extreme (e.g., +1, -1, or values near these limits), a solution may not be possible. If `attr(joint, "iterations")` equals `maxit`, consider increasing `maxit`, reducing `tol`, or choosing a more feasible `target` value:

- Nominal measures: $0 \leq target \leq 1$
- Ordinal measures: $-1 \leq target \leq +1$

Value

A frequency table reordered to approximate the target association. The returned object includes attributes:

`iterations` Number of iterations performed.

`target` Achieved association or correlation value.

Examples

```
tab <- table_data(3, 2)
tab
tab2 <- assoc_data(tab, target = 0.5)
tab2
```

`as_result` *Results with Rounding*

Description

Rounds `x` according to `digits` and the rounding function `FUN`, and sets a tolerance for the result. If `tol` is not provided, it defaults to `2*10^(-digits)`.

Usage

```
as_result(x, digits, tol = NA, FUN = round2)

tol(x)

rounded(x)

val(x)

digits(x)

as_res(x, digits, tol = NA, FUN = round2)

tolerance(x)
```

Arguments

`x`	numeric: value to round
`digits`	integer or character: Number of digits to use for rounding, see Details.
`tol`	numeric: tolerance for the result (defaults to `2*10^(-digits)` if NA)
`FUN`	function: rounding function (default: internal `round2()`)

Details

Results with Rounding

Creates a structured result with a numeric value rounded according to specified digits, an optional tolerance, and a rounding function.

By default, rounding is performed using an **internal function** `round2()`, which is similar to `exams::round2()`, but users should not call it directly. You can also supply a custom rounding function via the `FUN` argument.

If `digits` is a character, the following abbreviations are recognized:

`"integer"` digits = 0

`"%"` digits = 2

`"probability"` digits = 4

Partial matching of these names is allowed.

Value

A list of class `result` containing:

`x`	Original value
`r`	Rounded value
`digits`	Digits used for rounding
`tol`	Tolerance for the result

Examples

```
x <- as_result(1/3, "probability")
tol(x)
rounded(x)
digits(x)
```

`as_string` *Convert Vectors to Strings or Formatted Representations*

Description

These functions convert vectors into human-readable string representations. They can join elements, create LaTeX-formatted fractions, or label observations.

Usage

```
as_string(txt, collapse = ", ", last = ", and ")

as_sum(txt)

as_obs(txt, name = "x", sorted = FALSE, ...)

as_fraction(val, latex = FALSE, sorted = FALSE, ...)

lobs(txt, name = "x", sorted = FALSE, ...)

lstring(txt, collapse = ", ", last = ", and ")

lfrac(val, latex = FALSE, sorted = FALSE, ...)
```

Arguments

`txt`	Character vector to merge into a single string (used in `as_string`, `as_obs`).
`collapse`	Character string inserted between elements (default: `", "`).
`last`	Character string used between the last two elements (default: `", and "`).
`name`	Character string used as the observation name (default: `"x"`; used in `as_obs`).

`sorted`	Logical; if `TRUE`, sort the vector before conversion (default: `FALSE`).
`...`	Additional arguments passed to underlying functions.
`val`	Numeric vector of values to convert into fractions (used in `as_fraction`).
`latex`	Logical; if `TRUE`, returns fractions in LaTeX format `\frac{.}{.}` (default: `FALSE`; used in `as_fraction`).

Value

A single string, or a vector of formatted strings (for fractions in `as_fraction`).

Examples

```
x <- runif(5)
y <- c(TRUE, FALSE, NA)

# Basic string conversion
as_string(x)
as_string(y)
as_string(as.character(x))
as_string(as.character(y))

# Observations
as_obs(x)
as_obs(sort(x), sorted = TRUE)

# Fraction conversion
x <- round(runif(5), 2)
as_fraction(x)
as_fraction(x, latex = TRUE)

# Summing elements as a string
y <- round(runif(5), 2)
as_sum(y)
```

`as_table` *Convert to Table*

Description

Converts a vector or matrix into a formatted horizontal table using `xtable`. The output is returned as a character vector, where each element corresponds to a line of the table, suitable for printing or further formatting.

Usage

```
as_table(
  x,
  caption = NULL,
```

```
  label = NULL,
  align = NULL,
  digits = NULL,
  display = NULL,
  auto = FALSE,
  ...
)

toTable(
  x,
  caption = NULL,
  label = NULL,
  align = NULL,
  digits = NULL,
  display = NULL,
  auto = FALSE,
  ...
)
```

Arguments

`x`	An R object of class found among `methods(xtable)`. See below on how to write additional method functions for `xtable`.
`caption`	Character vector of length 1 or 2 containing the table's caption or title. If length is 2, the second item is the "short caption" used when LaTeX generates a "List of Tables". Set to `NULL` to suppress the caption. Default value is `NULL`.
`label`	Character vector of length 1 containing the LaTeX label or HTML anchor. Set to `NULL` to suppress the label. Default value is `NULL`.
`align`	Character vector of length equal to the number of columns of the resulting table, indicating the alignment of the corresponding columns. Also, `"|"` may be used to produce vertical lines between columns in LaTeX tables, but these are effectively ignored when considering the required length of the supplied vector. If a character vector of length one is supplied, it is split as `strsplit(align, "")[[1]]` before processing. Since the row names are printed in the first column, the length of `align` is one greater than `ncol(x)` if `x` is a `data.frame`. Use `"l"`, `"r"`, and `"c"` to denote left, right, and center alignment, respectively. Use `"p{3cm}"` etc. for a LaTeX column of the specified width. For HTML output the `"p"` alignment is interpreted as `"l"`, ignoring the width request. Default depends on the class of `x`.
`digits`	Numeric vector of length equal to one (in which case it will be replicated as necessary) or to the number of columns of the resulting table **or** matrix of the same size as the resulting table, indicating the number of digits to display in the corresponding columns. Since the row names are printed in the first column, the length of the vector `digits` or the number of columns of the matrix `digits` is one greater than `ncol(x)` if `x` is a `data.frame`. Default depends on the class of `x`. If values of `digits` are negative, the corresponding values of `x` are displayed in scientific format with `abs(digits)` digits.

`display`	Character vector of length equal to the number of columns of the resulting table, indicating the format for the corresponding columns. Since the row names are printed in the first column, the length of `display` is one greater than `ncol(x)` if `x` is a `data.frame`. These values are passed to the `formatC` function. Use `"d"` (for integers), `"f"`, `"e"`, `"E"`, `"g"`, `"G"`, `"fg"` (for reals), or `"s"` (for strings). `"f"` gives numbers in the usual `xxx.xxx` format; `"e"` and `"E"` give `n.ddde+nn` or `n.dddE+nn` (scientific format); `"g"` and `"G"` put `x[i]` into scientific format only if it saves space to do so. `"fg"` uses fixed format as `"f"`, but `digits` as number of *significant* digits. Note that this can lead to quite long result strings. Default depends on the class of `x`.
`auto`	Logical, indicating whether to apply automatic format when no value is passed to `align`, `digits`, or `display`. This 'autoformat' (based on `xalign`, `xdigits`, and `xdisplay`) can be useful to quickly format a typical `matrix` or `data.frame`. Default value is `FALSE`.
`...`	Additional arguments passed to `print.xtable`, such as `type`, `file`, or `sanitize.text.function`.

Details

Convert a Vector or Matrix to a Horizontal Table

Value

A character vector, each element representing a line of the table.

Examples

```
x <- runif(5)
tab <- vec2mat(x, colnames = 1:length(x))
as_table(tab)
```

---

as_ts *Convert ts_data Object to Time Series*

Description

Transforms a `ts_data` object into a standard R `ts` (time series) object.

Usage

```
as_ts(ts)
```

Arguments

`ts`	A `ts_data` object containing the time series values and structure.

Details

The function preserves the original time indices and frequency of the input `ts_data`. It calculates the `deltat` based on the time vector `t` in `ts_data`.

Value

A `ts` object representing the same time series as the input `ts_data`.

Examples

```
# Create a ts_data object with a linear trend
ts_obj <- ts_data(12, trend.coeff = c(sample(0:10, 1), sample(1 + (1:10)/20, 1)))

# Convert to standard ts object
ts_standard <- as_ts(ts_obj)
```

`binom_param` *Generate Valid Binomial Parameters*

Description

Creates a data frame of possible combinations of `n` (number of trials) and `p` (probabilities) that satisfy specified constraints on the mean, standard deviation, and approximation conditions for normal or Poisson distributions.

The function applies the following rules:

- If `length(mean) == 1` and it is an integer, it specifies the number of digits to which the mean should be rounded.
- If `mean = NA` (default), all mean values are allowed.
- If `length(mean) > 1`, only combinations where `n * p` equals one of the specified means are retained.
- The same logic applies to `sd` for the standard deviation.

The `norm` and `pois` arguments can be logical, `NA`, or a custom function of the form `function(n, p)`. They control which `(n, p)` pairs are considered valid:

- `NA` allows all combinations.
- A function returns `TRUE` for valid combinations and `FALSE` for invalid ones.
- `TRUE` enforces standard approximation rules:
  - `norm`: `n * p * (1 - p) > 9` (normal approximation condition)
  - `pois`: `n > 10 & p < 0.05` (Poisson approximation condition)
- `FALSE` excludes combinations that meet the approximation condition.

Note: The resulting data frame may be empty if no combinations meet all criteria.

Usage

```
binom_param(n, p, mean = NA, sd = NA, norm = NA, pois = NA, tol = 1e-06)
```

Arguments

`n`	integer vector of trial counts
`p`	numeric vector of probabilities
`mean`	numeric or integer specifying required mean digits or specific mean values
`sd`	numeric or integer specifying required standard deviation digits or specific sd values
`norm`	logical, `NA`, or function: restricts combinations to those valid for normal approximation
`pois`	logical, `NA`, or function: restricts combinations to those valid for Poisson approximation
`tol`	numeric: tolerance for numerical comparisons (default: `1e-6`)

Value

A data frame with columns `n`, `p`, `mean`, and `sd` representing valid parameter combinations.

Examples

```
binom_param(1000:50000, (5:25)/100, mean = 0, sd = 0)
```

`breaks` *Generate Break Points for Equidistant or Quantile Bins*

Description

Creates a numeric vector of break points for the given data `x`. The resulting breaks define bins that are either equidistant (fixed width) or non-equidistant (quantile-based). If `width` is not specified, it defaults to `diff(pretty(x))[1]`. The `probs` argument can either be a single integer, specifying the number of quantiles, or a numeric vector of probabilities in the interval $[0, 1]$.

Usage

```
breaks(x, width = NULL, probs = NULL)

add_breaks(x, width = NULL, probs = NULL)

dbreaks(x, width = NULL, probs = NULL)
```

Arguments

`x`	numeric vector: the data to compute breaks for.
`width`	numeric, optional: desired bin width (default: `NULL`, auto-calculated).
`probs`	numeric, optional: number of quantiles (single integer) or vector of probabilities in $[0, 1]$ for non-equidistant bins (default: `NULL`).

Details

If `probs` is used, break points are rounded to the nearest multiple of `width`. Duplicates are removed, and the range is extended if necessary to include the full range of `x`.

Value

A numeric vector containing the break points.

Examples

```
x <- rnorm(100, mean = 1.8, sd = 0.1)
breaks(x)                # equidistant bins
breaks(x, width = 0.1)   # custom width bins
breaks(x, width = 0.1, probs = 4)  # quantile-based bins
```

`calledBy` *Check if a Function Was Called by Another Function*

Description

Determines whether the current function call was initiated by a specified function. This is useful for conditional behavior depending on the caller.

Usage

```
calledBy(fun = "exams2pdf")

called_by(fun = "exams2pdf")
```

Arguments

`fun`	Character string specifying the name of the calling function to check for. Defaults to `"exams2pdf"`.

Value

A logical value: `TRUE` if the current call was triggered by `fun`, otherwise `FALSE`.

Examples

```
funB <- function() { calledBy("funA") }
funA <- function() { funB() }
funA()  # Returns TRUE because funB was called by funA
```

`catif` *Conditional Cat Output*

Description

Prints text using `cat` only when a specified logical condition is `TRUE`.

Usage

```
catif(cond, ...)

condition_cat(cond, ...)
```

Arguments

`cond`	Logical value. If `TRUE`, the text provided in `...` is printed using `cat`; if `FALSE`, nothing is printed.
`...`	Additional arguments passed to `cat`.

Value

Invisibly returns the value of `cond`.

Examples

```
catif(TRUE, "PDF")          # This text is printed
catif(FALSE, "Moodle")      # Nothing is printed
condition_cat(TRUE, "Hello") # Alias works the same way
```

`CImulen_data` *Confidence Interval and Sample Size for the Population Mean Value*

Description

Data generation for the necessary sample size of a confidence interval, for the population mean value. Either the estimation error e or the length of the interval l must be given ($l = 2 * e$). It is ensured that the computed `s` deviates from `sigma`.

Usage

```
CImulen_data(
  sigma,
  e = NULL,
  l = NULL,
  conf.level = c(0.9, 0.95, 0.99),
  nmin = 30,
  size = NA,
  u = c(seq(0.1, 0.4, 0.001), seq(0.6, 0.9, 0.001)),
  full = FALSE
)

dcimulen(
  sigma,
  e = NULL,
  l = NULL,
  conf.level = c(0.9, 0.95, 0.99),
  nmin = 30,
  size = NA,
  u = c(seq(0.1, 0.4, 0.001), seq(0.6, 0.9, 0.001)),
  full = FALSE
)
```

Arguments

`sigma`	numeric: vector of possible variance
`e`	numeric: vector of estimation errors
`l`	numeric: vector of lengths of the interval
`conf.level`	numeric: vector of confidence levels of the interval (default: `c(0.9, 0.95, 0.99)`)
`nmin`	numeric: minimal value of necessary observation (default: `30`)
`size`	numeric: sample size for computing a sample standard deviation. Default `NA` means that the solution of the estimation is used
`u`	numeric: vector of quantiles used to sample the sample standard deviation (default: `c(seq(0.15, 0.45, 0.001), seq(0.55, 0.85, 0.001))`)
`full`	logical: if `TRUE` then a data frame with possible solution is returned, otherwise a list with a randomly chosen solution is returned (default: `FALSE`)

Value

A data frame or a list with:

- e: estimation error
- `sigma`: population variance
- `conf.level`: confidence level
- l: interval length

- `x`: $1 - alpha/2$
- `q`: $z_{1-alpha/2}$
- `q2`: $z_{1-alpha/2}^2$
- `n`: computed minimal sample size
- `N`: the smallest integer, no less than `n`
- `s`: sample standard deviation

Examples

```
# one solution
CImulen_data (1:10, e=(1:10)/10)
# all solutions
mul <- CImulen_data (1:10, e=(1:10)/10, full=TRUE)
str(mul)
```

`CImu_data` *Confidence Intervals for a Population Mean*

Description

Computes confidence intervals for a population mean (`mu`) using either supplied data or data generated from a normal distribution with specified parameters. The function calculates key statistics such as the sample mean, standard deviation, and confidence intervals at user-defined confidence levels. Results are returned as a structured list containing observed and/or theoretical values, interval endpoints, and related measures.

Usage

```
CImu_data(
  x = NULL,
  n = length(x),
  xbar = NULL,
  sd = NULL,
  conf.level = c(0.9, 0.95, 0.99),
  mu = NULL,
  sigma = NULL
)

dcimu(
  x = NULL,
  n = length(x),
  xbar = NULL,
  sd = NULL,
  conf.level = c(0.9, 0.95, 0.99),
  mu = NULL,
  sigma = NULL
)
```

Arguments

`x`	numeric vector of observed data. If `NULL`, data are simulated.
`n`	integer: sample size (if `n < 1`, defaults to `5`).
`xbar`	numeric: sample mean (computed from `x` if `NULL`).
`sd`	numeric: sample standard deviation (computed from `x` if `NULL`).
`conf.level`	numeric vector of confidence levels (default: `c(0.9, 0.95, 0.99)`).
`mu`	numeric: true population mean (used for simulation if `x` is `NULL`).
`sigma`	numeric: population standard deviation(s) (used for simulation if `x` is `NULL`).

Value

A list containing:

`a`	upper-tail probability `1 - (1 - conf.level) / 2`
`n`	sample size
`xbar`	sample mean
`mu`	theoretical mean (if provided)
`sd`	sample standard deviation
`sigma`	theoretical standard deviation (if provided)
`df`	degrees of freedom (if using a *t* distribution)
`q`	critical value(s) from the normal or *t* distribution
`ss`	standard deviation used in calculations (`sd` or `sigma`)
`e`	margin of error (half-width of the interval)
`l`	interval length
`v`	confidence interval endpoints

Examples

```
# Using observed data
x <- rnorm(100)
CImu_data(x, conf.level = 0.95)

# Simulating data internally
CImu_data(n = 100, conf.level = 0.95, mu = 0, sigma = 1)
```

`CIpilen_data` *Confidence Interval and Sample Size for the Population Proportion*

Description

Data generation for the necessary sample size of a confidence interval, for the population proportion, using z^2/l^2). Either the estimation error e or the length of the interval l must be given ($l = 2 * e$). It is ensured that the computed `p` deviates from `pi`.

Usage

```
CIpilen_data(
  pi,
  e = NULL,
  l = NULL,
  conf.level = c(0.9, 0.95, 0.99),
  nmin = 30,
  size = NA,
  u = c(seq(0.1, 0.4, 0.001), seq(0.6, 0.9, 0.001)),
  full = FALSE
)

dcipilen(
  pi,
  e = NULL,
  l = NULL,
  conf.level = c(0.9, 0.95, 0.99),
  nmin = 30,
  size = NA,
  u = c(seq(0.1, 0.4, 0.001), seq(0.6, 0.9, 0.001)),
  full = FALSE
)
```

Arguments

`pi`	numeric: vector of possible population proportions
`e`	numeric: vector of estimation errors
`l`	numeric: vector of lengths of the interval
`conf.level`	numeric: vector of confidence levels of the interval (default: `c(0.9, 0.95, 0.99)`)
`nmin`	numeric: minimal value of necessary observation (default: `30`)
`size`	numeric: sample size for computing a sample standard deviation. Default `NA` means that the solution of the estimation is used
`u`	numeric: vector of quantiles used to sample the sample standard deviation (default: `c(seq(0.15, 0.45, 0.001), seq(0.55, 0.85, 0.001)))`

`full`	logical: if `TRUE` then a data frame with possible solution is returned, otherwise a list with a randomly chosen solution is returned (default: `FALSE`)

Value

A data frame or a list with:

- e estimation error
- `pi` population proportion
- `conf.level` confidence level
- l interval length
- `x` $1 - alpha/2$
- `q` $z_{1-alpha/2}$
- `q2` $z^2_{1-alpha/2}$
- `n` computed minimal sample size
- `N` the smallest integer, no less than `n`
- `p` sample proportion

Examples

```
# one solution
CIpilen_data((1:9/10), (1:9)/10)
# all solutions
pil <- CIpilen_data((1:9/10), (1:9)/10, full=TRUE)
str(pil)
```

`cleanFile` *Clean or Schedule Deletion of a File*

Description

Deletes a file immediately after a specified delay or schedules it for deletion when the R session ends.

Usage

```
cleanFile(file, delay)
```

Arguments

`file`	Character string. Path to the file to be deleted.
`delay`	Numeric. Number of seconds to wait before deleting the file. If `delay = 0`, the file will persist until the R session ends.

Value

Invisibly returns the file path.

Examples

```
{ # Delete a temporary file after 2 seconds
  tmp <- tempfile();
  if (file.create(tmp)) cleanFile(tmp, delay = 1)
  # Keep a temporary file until the R session ends
  tmp2 <- tempfile()
  if (file.create(tmp2)) cleanFile(tmp2, delay = 0)
}
```

`combinatorics` *Combinatorics*

Description

- `permutation` computes the number of permutations
- `variation` computes the number of variations with and without replication
- `combination` computes the number of combinations with and without replication
- `combinatorics` computes all combinatorics results for `k` < `n` and returns it as list of:

  `permutation.n` $P(n)$
  `permutation.k` $P(k)$
  `permutation.nk` $P(n;k)$
  `variation` $V(n;k)$
  `variation.rep` $V^W(n;k)$
  `combination` $K(n;k)$
  `combination.rep` $K^W(n;k)$
- `lfact` computes the natural logarithm of the factorial of a given number `n`
- `lfactquot` calculates the natural logarithm of the quotient of factorials
- `lbinom` computes the natural logarithm of the binomial coefficient, "`n` choose `k`"

Usage

```
combinatorics(n, k)

variation(n, k, repl = FALSE)

combination(n, k, repl = FALSE)

permutation(n, k = rep(1, n))

lfact(n)
```

```
lfactquot(n, ...)

lbinom(n, k)

combo(n, k, repl = FALSE)

combs(n, k)

fact(n)

factquot(n, ...)

binom(n, k)
```

Arguments

`n`	numeric: total number of elements
`k`	numeric: number of elements to choose
`repl`	logical: with repetition (default: `FALSE`)
`...`	numeric: further arguments for `lfactquot`

Value

A list.

Examples

```
permutation(8)
permutation(8, c(1,3,2,2))
combination(8, 4)
combination(8, 4, TRUE)
variation(8, 4)
variation(8, 4, TRUE)
combinatorics(8, 4)
```

`cor_data` *Generate Data with a Target Correlation*

Description

Rearranges the order of `y` to construct a dataset with a target correlation `r` between `x` and `y`, as defined by `stats::cor()`. The marginal distributions of `x` and `y` are preserved, but the achieved correlation may deviate from the target. The algorithm iteratively adjusts the ordering of `y`; increasing `maxit` may improve accuracy if results are unsatisfactory.

Usage

```
cor_data(
  x,
  y,
  r,
  method = c("pearson", "kendall", "spearman"),
  ...,
  maxit = 1000
)

dcorr(x, y, r, method = c("pearson", "kendall", "spearman"), ..., maxit = 1000)
```

Arguments

`x`	numeric. Vector of `x` values.
`y`	numeric. Vector of `y` values.
`r`	numeric. Desired correlation.
`method`	character. Correlation coefficient to compute. Options are `"pearson"` (default), `"kendall"`, or `"spearman"`.
`...`	Additional arguments passed to `stats::cor()`.
`maxit`	integer. Maximum number of iterations (default: `1000`).

Value

A two-column matrix with `x` and reordered `y`. An attribute `interim` stores a matrix of intermediate values, which depends on `method`:

- `pearson`: Rows include x_i, y_i, $x_i - \bar{x}$, $y_i - \bar{y}$, squared deviations, and cross-products.
- `kendall`: Rows include x_i, y_i, p_i (concordant pairs), and q_i (discordant pairs).
- `spearman`: Rows include x_i, y_i, ranks of `x` and `y`, and squared rank differences.

In all cases, an additional column with row sums is appended.

Examples

```
x <- runif(6)
y <- runif(6)
xy <- cor_data(x, y, r = 0.6)
cbind(x, y, xy)
```

`data_n` *Generate Sequences of Sample Sizes*

Description

Generate sequences of integers representing sample sizes within a specified range.

Functions support two specialized sequences:

- `data_nsq` / `dnsq`: sample sizes that are perfect squares.
- `data_n25` / `dn25`: sample sizes divisible only by 2 and 5.

Usage

```
data_n(max, min = 5)

data_nsq(max, min = 5)

data_n25(max, min = 5)

dn(max, min = 5)

dn25(max, min = 5)

dnsq(max, min = 5)
```

Arguments

`max`	Integer. Maximum sample size.
`min`	Integer. Minimum sample size (default: `5`).

Value

An integer vector of sample sizes satisfying the criteria.

Examples

```
data_n(10)
data_nsq(1000)
data_n25(1000)
```

`data_prob2` *Probability/Frequency Matrix Generation*

Description

Generates a `nrow` × `ncol` matrix with probabilities / frequencies. If `data` is given it will be normalized such that `sum(data[is.finite(data)])==1`. If no `rownames` or `colnames` are given then event names from `LETTERS` are used. The returned matrix will have the following attributes:

- `marginals` a list of the row and column marginal distributions
- `byrow` a matrix with conditional probabilities by row
- `bycol` a matrix with conditional probabilities by column
- `expected` a matrix with the expected probabilities under independence
- `prob` a vector of all the probabilities computed (except the expected ones)

Usage

```
data_prob2(
  data = NULL,
  nrow = 2,
  ncol = 2,
  colnames = NULL,
  rownames = NULL,
  ...
)

prob_mx(data = NULL, nrow = 2, ncol = 2, colnames = NULL, rownames = NULL, ...)

dprob2(data = NULL, nrow = 2, ncol = 2, colnames = NULL, rownames = NULL, ...)
```

Arguments

`data`	an optional data vector. Non-atomic classed R objects are coerced by `as.vector` and all attributes are discarded.
`nrow`	numeric: desired number of rows (default: 2)
`ncol`	numeric: desired number of columns (default: 2)
`colnames`	character: names of column events
`rownames`	character: names of row events
`...`	further parameters given to `ddiscrete()`

Value

A matrix and some attributes.

Examples

```
x <- data_prob2()
str(x)
data_prob2(colnames="E")
data_prob2(nrow=3)
```

`ddiscrete` *Discrete Probability Function*

Description

Creates a discrete probability function based on `x` with a resolution `unit`. If `unit` is not given then `unit` will be 10, 100, 1000, ... depending on the length of the discrete probability function.

Usage

```
ddiscrete(x, unit = NULL, zero = FALSE)
```

Arguments

`x`	numeric: number of elements of vector of initial probabilities
`unit`	integer: reciprocal of the smallest non-zero probability (default: `NULL`)
`zero`	logical: zeros are allowed in the final probabilities (default: `FALSE`)

Value

A discrete probability function.

Examples

```
ddiscrete(runif(6))
ddiscrete(6)
ddiscrete(6, 20)
ddiscrete(c(1,0,0,0), zero=TRUE)
```

`ddiscrete2` *Bivariate Discrete Probability Function*

Description

Creates a bivariate discrete probability function based on the marginal probability functions `row` and `col`. If `unit` is not given then `unit` will be the product of the units used in `row` and `col`, otherwise it will appear as the least common multiple `unit` product of the units used in `row` and `col`. If `target` is `NA` then the common distribution of two independent random variables is returned, otherwise an iterative algorithm is run to approach a `target` association or correlation measure, see also `assoc_data()` (called internally). `zero` allows for zero entries in the common distribution. `FUN` computes the association or correlation measures based on a frequency table. `tol` gives the maximal deviation of the association or correlation measure and the `target` value. `maxit` limits the number of steps. Please note that a solution is not guaranteed, especially for extreme values for `target`, for example for $+1$, -1 or nearby values. If `attr(joint, "iterations")==maxit` then you need either to increase `maxit`, to decrease `tol` or to check if you have chosen an appropriate `target` value (for a nominal measure in $0 <= target <= 1$, for ordinal measure in $-1 <= target <= +1$).

Usage

```
ddiscrete2(
  row,
  col,
  unit = NULL,
  zero = FALSE,
  FUN = nom.cc,
  target = NA,
  tol = 0.001,
  maxit = 500,
  ...
)

biv_discrete_prob(
  row,
  col,
  unit = NULL,
  zero = FALSE,
  FUN = nom.cc,
  target = NA,
  tol = 0.001,
  maxit = 500,
  ...
)
```

Arguments

`row`	numeric: marginal `row` distribution

`col`	numeric: marginal `col` distribution
`unit`	integer: reciprocal of the smallest non-zero probability (default: `NULL`)
`zero`	logical: zeros are allowed in the final probabilities (default: `FALSE`)
`FUN`	function: association or correlation function (default: `nom.cc`)
`target`	numeric: target association or correlation (default: `NA`)
`tol`	numeric: tolerance for target association or correlation (default: `0.001`)
`maxit`	integer: maximal number of iterations (default: `100`)
`...`	further parameters for `FUN`

Value

A bivariate discrete probability function.

Examples

```
row <- ddiscrete(runif(5))
col <- ddiscrete(runif(3))
joint <- ddiscrete2(row, col)
joint
joint <- ddiscrete2(row, col, target=0.5)
joint
nom.cc(joint*attr(joint, "unit"))
```

`ddunif2` *Sum of Two Independent Discrete Uniform Distributions*

Description

Probability mass function, distribution function, quantile function and random generation for the sum of two independent discrete uniform distributions.

Usage

```
ddunif2(x, min = 1, max = 6)

pdunif2(q, min = 1, max = 6)

qdunif2(p, min = 1, max = 6)

rdunif2(n, min = 1, max = 6)

sum_discrete_unif_cdf(x, min = 1, max = 6)

sum_discrete_unif_pmf(q, min = 1, max = 6)

sum_discrete_unif_quantile(p, min = 1, max = 6)

sum_discrete_unif_rand(n, min = 1, max = 6)
```

Arguments

`x, q`	numeric: vector of quantiles
`min`	numeric: lower limit of the distribution (default: `1`)
`max`	numeric: upper limit of the distribution (default: `6`)
`p`	numeric: vector of probabilities
`n`	numeric: number of observations. If `length(n)>1`, the length is taken to be the number required.

Value

A numeric vector with the same length as `x`.

Examples

```
ddunif2(1:13)
pdunif2(1:13)
qdunif2((0:4)/4)
rdunif2(10)
```

`DiscreteUniform` *Discrete uniform distribution*

Description

Probability mass function, distribution function, quantile function and random generation for the discrete uniform distribution. Documentation, examples and interface are taken from `extraDistr::DiscreteUniform`.

Usage

```
ddunif(x, min, max, log = FALSE)

pdunif(q, min, max, lower.tail = TRUE, log.p = FALSE)

qdunif(p, min, max, lower.tail = TRUE, log.p = FALSE)

rdunif(n, min, max)
```

Arguments

`x, q`	vector of quantiles.
`min, max`	lower and upper limits of the distribution. Must be finite.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise, $P[X > x]$.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > `1`, the length is taken to be the number required.

Details

If `min` == `max`, then discrete uniform distribution is a degenerate distribution.

Examples

```
x <- rdunif(1e5, 1, 10)
xx <- -1:11
plot(prop.table(table(x)), type = "h")
lines(xx, ddunif(xx, 1, 10), col = "red")
hist(pdunif(x, 1, 10))
xx <- seq(-1, 11, by = 0.01)
plot(ecdf(x))
lines(xx, pdunif(xx, 1, 10), col = "red")
```

`distribution` *Class Distribution*

Description

Holds an univariate distribution including its parameters. The name of the distribution is used to determine the right use of the function. For example, in the case of function for quantiles: `paste0("q", name)`. Usually the full name has to be used; some abbreviated names are possible:

- `binom` binomial distribution, parameters: `size`, `prob`
- `hyper` hypergeometric distribution, parameters: `m`, `n`, `k`
- `geom` geometric distribution, parameters: `prob`
- `pois` Poisson distribution, parameters: `lambda`
- `unif` continuous uniform distribution, parameters: `min`, `max`
- `dunif` discrete uniform distribution, parameters: `min`, `max`
- `dunif2` continuous uniform distribution, parameters: `min`, `max`
- `exp` exponential distribution, parameter: `rate`
- `norm` normal distribution, parameters: `mean`, `sd`
- `lnorm` log-normal distribution, parameters: `meanlog`, `sdlog`
- `t` Student t distribution, parameter: `df`
- `chisq` chi-squared distribution, parameter: `df`
- `f` F distribution, parameters: `df1`, `df2`

Note that a probability mass/density, quantile and a cumulative distribution function must exist.

The following functions exists for `disributions`:

- `distribution` creates a distribution with `name` and parameters
- `quantile` computes the quantiles of a distribution using `paste0('q', name)`
- `cdf` computes the cumulative distribution function of a distribution using `paste0('p', name)`

- `pmdf` computes the probability mass/density function of a distribution using `paste0('d', name)`
- `prob` computes the probability for a interval between `min` and `max` (`max` included, `min` excluded)
- `prob1` computes the point probability f
- `is.distribution` checks if `object` is distribution object. If `name` is given then it checks whether the distribution type is the same
- `toLatex` generates a LaTeX representation of the distribution an its parameter

Usage

```
distribution(name, ...)

## Default S3 method:
distribution(name, ..., discrete = NA)

## S3 method for class 'distribution'
quantile(x, probs = seq(0, 1, 0.25), ...)

cdf(x, q, ...)

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

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

pmdf(d, x, ...)

## S3 method for class 'distribution'
toLatex(object, name = NULL, param = NULL, digits = 4, ...)

is.distribution(object, name = NULL)

prob(d, min = -Inf, max = +Inf, tol = 1e-06)

prob1(d, x, tol = 1e-06)

compute_cdf(x, q, ...)

compute_pmdf(d, x, ...)

compute_probability(d, min = -Inf, max = +Inf, tol = 1e-06)

point_probability(d, x, tol = 1e-06)

pprob(d, x, tol = 1e-06)
```

```
is_distribution(object, name = NULL)
```

Arguments

`name`	character: a replacement of the name of the distribution type
`...`	further named distribution parameters
`discrete`	logical: is the distribution discrete? (default: `NA`)
`x`	vector of values
`probs`	numeric: vector of probabilities with values in $[0, 1]$.
`q`	numeric: vector of quantiles
`object`	distribution object
`d`	distribution
`param`	character: names for the distribution parameters
`digits`	integer: number of digits used in `signif`
`min`	numeric: left border of interval
`max`	numeric: right border of interval
`tol`	numeric: tolerance for `max==min` (default: `1e-6`)

Value

A distribution object.

Examples

```
d <- distribution("norm", mean=0, sd=1)
quantile(d)
quantile(d, c(0.025, 0.975))
cdf(d, 0)
is.distribution(d)
is.distribution(d, "t")
toLatex(d)
```

`distributions` *Distributions*

Description

A data frame with the R function names, LaTeX names, discreteness and package origin of a distribution.

Usage

```
data(distributions)
```

Format

A data frame with columns `r`, `latex`, `discret` and `package`

Examples

```
data(distributions)
distributions
```

`divisor_25` *Number Properties*

Description

- `is_terminal` checks whether `x`'s can be expressed as a terminal fraction, basically `divisor_25(denominator(x))`
- `divisor_25` checks whether all `x`'s can be expressed as $2^x 5^y$
- `prime_numbers` returns all prime numbers up to a limit
- `primes` prime factorization of `x`, returns a matrix with the power of each prime number
- `has_digits` checks whether the `x`'s have only `digits` after the decimal point, basically `abs(x-round(x, digits))<tol`
- `all_integer` checks whether all `x`'s are integer, basically `all(has_digits(x,0))`

Usage

```
divisor_25(x)

denominator_25(x)

is_terminal(x)

round_25(x)

prime_numbers(n, sieve = FALSE)

primes(x, min = 2)

has_digits(
  x,
  digits = 2,
  tol = 10^{
     -digits - 6
 }
)

all_integer(x)
```

```
only_digits(
  x,
  digits = 2,
  tol = 10^{
     -digits - 6
 }
)

is_term(x)

denom_25(x)
```

Arguments

`x`	numeric: values to test/check
`n`	integer: find all prime numbers up to n
`sieve`	logical: should in any case the Sieve of Eratosthenes be used to compute prime numbers (default: `FALSE`)
`min`	integer: the minimum prime number used (default: `2`)
`digits`	numeric: number of digits to check (default: `2`)
`tol`	numeric: max. deviation from the rounded `x` (default: `1e-6`)

Value

logical

Examples

```
is_terminal(2/3)   # 0.6666... non-terminal
is_terminal(1/5)   # 0.2       terminal
divisor_25(1:25)
prime_numbers(100) # all prime numbers less equal 100
primes(1:20)       # prime factorization of 1 to twenty
```

`equal` *Conditional Value Matching*

Description

It performs a comparison by checking if either `abs(x - y) < tol` when `outer == FALSE`, or if an a exists or a `y[j]` for each `x[i]` such that the condition `abs(x[i] - y[j]) < tol` is satisfied.

Usage

```
equal(x, y, tol = 1e-06, outer = FALSE)

approx_equal(x, y, tol = 1e-06, outer = FALSE)
```

Arguments

`x`	numeric
`y`	numeric
`tol`	numeric: tolerance (default: `1e-6`)
`outer`	logical: compares directly or verifies whether `x` is present within `y` (default: FALSE).

Value

logical

Examples

```
equal(9*1/9, 1)
```

`equations` *Equations and Variables*

Description

`equations` defines a set of equations using the formula interface including a LaTeX representation of the formulae.

`variables` sets the variable values, the LaTeX representation and the solution interval. The first argument must be the `equations` object. A named parameter starts the setting for a specific variable, e.g. `..., s=1, pos(5), "s^2",...` sets for the variable `s` first its numerical value, second the solution interval and finally the LaTeX representation.

Usage

```
equations(...)

variables(...)
```

Arguments

`...`	For `equations`, an even number of parameters: formula, LaTeX representation, formula, LaTeX representation, etc. For `variables`, parameters to set one or more variables.

Value

(for `equations`) An `equations` object.

(for `variables`) The modified `equations` object.

Examples

```
# The equations describe the formulae for an confidence interval of the mean
e <- equations(o~x+c*s/sqrt(n), "v_o=\\bar{x}+c\\cdot\\frac{s^2}{n}",
               u~x-c*s/sqrt(n), "v_u=\\bar{x}-c\\cdot\\frac{s^2}{n}",
               e~c*s/sqrt(n),   "e  =c\\cdot\\frac{s^2}{\\sqrt{n}}",
               l~2*e,           "l  =2\\cdot e"
               )
print(e)
e <- variables(e,
               x=0,    "\\bar{x}",
               c=2.58, dbl(2),
               s=1,    pos(5), "s^2",
               n=25,   pos(5),
               l=pos(5),
               e=pos(5),
               u="v_u", o="v_o")
print(e)
```

`exams2call` *Traceback for* `exams2` *Functions*

Description

Returns a list with the functions' names and parameters called from `.traceback()`. The function name must start with `"exams2"`.

Usage

```
exams2call(prefix = "exams2")
```

Arguments

`prefix`	character: start of the function name (default: `"exams2"`)

Value

A list with the function name and its evaluated parameters.

Examples

```
   exams2call()                 # access current call stack
```

`exercise` *Data Exercise Structure*

Description

Data structure for exercise data.

Usage

```
exercise(exer, ...)

## Default S3 method:
exercise(exer = NULL, ...)

exercise_data(exer, ...)
```

Arguments

`exer`	an exercise object (default: `NULL`)
`...`	further parameters

Value

An exercise object.

Examples

```
exer <- exercise()           # new exercise
exer <- exercise(exer, x=3)  # add x to the exercise
```

`extremes` *Extremes*

Description

Computes the real valued extremes (minima, maxima, and saddle points) for a univariate polynomial. The computation can be limited to a specific type of extremes.

Usage

```
extremes(p, type = c("all", "minimum", "maximum", "saddle"), tol = 1e-09)
```

Arguments

`p`	a polynomial
`type`	character: either `all` (default), `minimum`, `maximum`, or `saddle`
`tol`	numeric: if the absolute value of the imaginary part of the zeroes of the derivative of `p` is smaller than `tol`, it will be considered as zero

Value

A numeric vector.

Examples

```
p <- polynomial(c(0,0,0,1))
extremes(p)
p <- integral(poly.calc(-1:1))
extremes(p)
```

`fcvt` *Number to String Conversion (Floating Point / Fractional Number)*

Description

Converts a number to a string containing either a floating point or a fractional number. Note that a repeating or recurring decimal, which is a number whose decimal representation becomes periodic, can also be expressed as a rational number. For example, $\frac{1}{3} = 0.333333333... = 0.\overline{3}$. It is the workhorse used in `num2str`.

- If `denom` is negative then always decimal point numbers are used (default).
- If `denom` is zero then a mix of decimal point and fractional numbers are used (whatever is shorter).
- If `denom` is one then fractional numbers are used except for integers.
- If `denom` is larger than one, then the denominator is set to `denom` if possible.

Usage

```
fcvt(x, nsmall = 15, plus = FALSE, denom = -1)
```

Arguments

`x`	numeric: numbers to convert
`nsmall`	integer: number of significant digits for the mantissa/significand (default: `16`)
`plus`	logical: for positive numbers a plus sign should be used (default: `FALSE`)
`denom`	integer: denominator for a fractional number

Value

A character.

Examples

```
x1 <- c(NA, NaN, -Inf, Inf, 0, pi*10^(-20:20))
fcvt(x1)
x2 <- c(-0.36, 3.6, -30.6, 0.36)
fcvt(x2)
x3 <- c((0:16)/8, 1/3)
fcvt(x3)           # as floating point number, equals denom=-1
fcvt(x3, denom=0)  # as floating point or fractional number
fcvt(x3, denom=1)  # as fractional number except for integers
fcvt(x3, denom=8)  # as fractional number with denominator denom if possible
```

`firstmatch` *Firstmatch*

Description

`firstmatch` seeks matches for the elements of its first argument among those of its second. For further details please check `base::charmatch()`. `charmatch` returns a zero if multiple matches are found, whereas `firstmatch` returns the first partial match if multiple matches are found.

Usage

```
firstmatch(x, table, nomatch = NA_integer_)
```

Arguments

`x`	character: the values to be matched; converted to a character vector if necessary
`table`	character: the values to be matched against; converted to a character vector if necessary
`nomatch`	integer: the value to be returned at non-matching positions (default: `NA_integer_`)

Value

An integer.

Examples

```
firstmatch("d", c("chisq", "cauchy"))
charmatch("c", c("chisq", "cauchy"))
firstmatch("c", c("chisq", "cauchy"))
firstmatch("ca", c("chisq", "cauchy"))
```

`fractions` *Fractions*

Description

Finds rational approximations to the components of a real numeric object, using a standard continued fraction method. Calls `MASS::fractions()` (Please refer to that for further details).

Usage

```
fractions(x, cycles = 10, max.denominator = 2000, ...)

approx_rational(x, cycles = 10, max.denominator = 2000, ...)
```

Arguments

`x`	any object of the numeric mode (missing values are allowed)
`cycles`	the maximum number of steps to be used in the continued fraction approximation process
`max.denominator`	an early termination criterion. If any partial denominator exceeds `max.denominator`, the continued fraction stops at that point
`...`	further arguments

Value

An object of the class `fractions`. A structure with a `.Data` component, the same as the numeric `x` input, but with the rational approximations held as the character vector attribute `fracs`. Arithmetic operations on `fractions` objects are possible.

Examples

```
X <- matrix(runif(25), 5, 5)
fractions(X) #;)
fractions(solve(X, X/5))
fractions(solve(X, X/5)) + 1
```

`gapply` *Apply Grid*

Description

Runs all combinations of elements in `...` as parameters of `FUN` (grid apply). `I(.)` can be used to avoid that an element is interpreted as a grid value. If an error occurs, then the result of `FUN` will not be stored. You may notice missing indices in the returning list.

Usage

```
gapply(FUN, ..., .simplify = TRUE)

apply_grid(FUN, ..., .simplify = TRUE)
```

Arguments

`FUN`	function or character: a string naming the function to be called
`...`	list: of arguments of the function to be called. The `names` attribute of `args` returns the argument `names`
`.simplify`	logical: should the result be simplified to a data frame (if possible)? (default: `TRUE`)

Value

A list or a data frame with the function results.

Examples

```
# 8 function calls: sum(1,3,5), sum(1,3,6), ..., sum(2,4,6)
gapply("sum", 1:2, 3:4, 5:6)
# 4 function calls: sum(1,3,5:6), sum(1,4,5:66), ..., sum(2,4,5:6)
gapply("sum", 1:2, 3:4, I(5:6))
```

`getNames` *Safely extract names from an object*

Description

Returns the names of an object as a character vector. Optionally enforces that names are non-NULL and non-empty when `strict = TRUE`. The function **does not modify the original object**.

Usage

```
getNames(x, strict = FALSE)
```

Arguments

`x`	An R object (vector, list, etc.). Must not be `NULL`.
`strict`	Logical; if TRUE, errors if `x` is `NULL`, any names are missing, or any names are empty strings.

Value

A character vector of names, same length as `x`.

- If `x` is an empty list or zero-length object, returns `character(0)`.
- If `names(x)` is `NULL` and `strict = FALSE`, returns a vector of empty strings of the same length as `x`.

Examples

```
x <- c(a = 1, b = 2)
getNames(x)

y <- 1:3
getNames(y)

z <- c(1, 2); names(z) <- c("a", "")
# getNames(z, strict = TRUE) # would throw an error: "`x` has empty name(s)"

# Empty list returns character(0)
getNames(list(), strict = TRUE)

# NULL input triggers an error
# getNames(NULL, strict = TRUE)
```

`grade` *Grades*

Description

Computes a grade based on the points of the grade scheme by the Humboldt University of Berlin. (See §96c and §102 in the Achte Änderung der Fächerübergreifenden Satzung zur Regelung von Zulassung, Studium und Prüfung der Humboldt-Universität zu Berlin (ZSP-HU))

Usage

```
grade(points, maxpts = max(points), fixed = TRUE)

hu_grade(points, maxpts = max(points), fixed = TRUE)
```

Arguments

`points`	numeric: points achieved in exam
`maxpts`	numeric: maximal number of achievable points in an exam (default: `max(points)`)
`fixed`	logical: a fixed or relative grade scheme (default: `TRUE`)

Value

Grades as a function of points.

Examples

```
x <- round(runif(100, 0, 22.4))
grade(x, 22)
```

`grouped_data` *Central Tendency Measures' Computation of Grouped Data*

Description

Computes mean, mode or quantile/median of grouped data.

Usage

```
grouped_data(x, n, compute = c("mean", "fine", "coarse"), tol = 1e-06)

grouped_stats(x, n, compute = c("mean", "fine", "coarse"), tol = 1e-06)

dgrouped(x, n, compute = c("mean", "fine", "coarse"), tol = 1e-06)
```

Arguments

`x`	numeric: borders
`n`	numeric: absolute frequencies for each group
`compute`	numeric/character: coefficient to compute
`tol`	numeric: tolerance for numerical comparison

Value

A list with the class, result and a table.

Examples

```
x <- 1:4
n <- ddiscrete(runif(3))
grouped_data(x, n)
```

`gsimplify` *Simplified* `hyperloop` *Object*

Description

Simplifies a `hyperloop` object if possible.

Usage

```
gsimplify(ga, exclude = NULL, subset = NULL)

simplify_hyperloop(ga, exclude = NULL, subset = NULL)

simple_hloop(ga, exclude = NULL, subset = NULL)
```

Arguments

`ga`	list: of a `hyperloop` object
`exclude`	character or integer: elements to exclude in each list element of `ga` (default: `NULL`)
`subset`	indices specifying elements of ga to extract (default: `NULL`)

Value

A data frame if possible, otherwise a list.

Examples

```
# calls: t.test(x, -1), t.test(x, 0), t.test(x, 1)
ga <- gapply(t.test, x=I(rnorm(100)), mu=-1:1)
# no simplication since `data.name` and `conf.int` have lengths larger  than one
gsimplify(ga)
#' simplification is now possible
gsimplify(ga, exclude=c("conf.int", "data.name"))
```

`histbreaks` *Histogram Breakpoints*

Description

Randomly selects `size` breakpoints from `breaks`. If `outer` is `TRUE`, then the first and last element of `breaks` is always included into the returned break points. If `size` is a vector, the number of breakpoints is first sampled from `size`.

Usage

```
histbreaks(breaks, size, outer = TRUE, ...)

rand_breaks(breaks, size, outer = TRUE, ...)

dhistbreaks(breaks, size, outer = TRUE, ...)
```

Arguments

`breaks`	numeric: a vector of possible break points
`size`	integer: number of break points
`outer`	logical: should be the first and last element of the included breaks (default: `TRUE`)
`...`	further parameters given if sampling of `size` is necessary, see base::sample

Value

A vector of breakpoints.

Examples

```
# Always includes 100 and 200
histbreaks(seq(100, 200, by=10), 4)
# Always includes 100 and 200 and chooses randomly between 3 to 5 break points
histbreaks(seq(100, 200, by=10), 3:5)
# May not include 100 and 200
histbreaks(seq(100, 200, by=10), 4, outer=FALSE)
```

`histdata` *Histogram Data*

Description

Returns data for a histogram. Calls internally `hist(..., plot=FALSE)`.

- `mean` returns the mean of the data.
- `quantile` and `median` return the quantile(s) or median with an attribute pos, the class number of the quantile(s), or the median.

Usage

```
histdata(x, breaks = "Sturges", probs = seq(0, 1, 0.25), ...)

## S3 method for class 'histogram'
quantile(x, probs = seq(0, 1, 0.25), ...)

## S3 method for class 'histogram'
```

```
median(x, ...)

## S3 method for class 'histogram'
mean(x, ...)

dhist(x, breaks = "Sturges", probs = seq(0, 1, 0.25), ...)
```

Arguments

`x`	numeric data or histogram data
`breaks`	one of: • a vector giving the breakpoints between histogram cells, • a function to compute the vector of breakpoints, • a single number giving the number of cells for the histogram, • a character string naming an algorithm to compute the number of cells (see 'Details'), • a function to compute the number of cells. In the last three cases the number is a suggestion only; as the breakpoints will be set to `pretty` values, the number is limited to `1e6` (with a warning if it was larger). If `breaks` is a function, the `x` vector is supplied to it as the only argument (and the number of breaks is only limited by the amount of available memory).
`probs`	numeric: probabilities to use if `breaks="Quantile"` (default: `seq(0, 1, 0.25)`)
`...`	further parameters used in graphics::hist

Value

Like in graphics::hist, but with this additional list of elements:

- `lower` lower class borders,
- `upper` upper class borders,
- `width` class widths,
- `relfreq` the relative class frequency,
- `cumfbrk` the cumulated relative frequency of the `breaks`,
- `maxdens` the indices of the maximal `density` values,
- `maxcount` the indices of the maximal `count` values
- `x` the original finite data, and
- `class` the class number for each value in `x`.

Examples

```
#1
x <- seq(0, 1, by=0.25)
print(hist(x, plot=FALSE))
histdata(x)
#2
```

```
x <- seq(0, 1, by=0.25)
print(hist(x, x, plot=FALSE))
histdata(x, x)
#3
print(hist(x, x, right=FALSE, plot=FALSE))
histdata(x, x, right=FALSE)
```

`histwidth` *Histogram Widths*

Description

Generates a set of class breaks and absolute frequencies for the range from `from` to `to`. Class widths are randomly sampled from the vector `widths`. The total number of classes (`nb`) must be an integer multiple of `min(widths)`; otherwise, the function stops with an error. If the initial frequencies (`n`) are too small, they can be scaled by an integer factor. The routine also checks whether the resulting class densities are terminating decimals.

Usage

```
histwidth(from, to, widths, dmax = 2000, maxit = 1000)

width_breaks(from, to, widths, dmax = 2000, maxit = 1000)

dhistwidth(from, to, widths, dmax = 2000, maxit = 1000)
```

Arguments

`from`	numeric: start value of the range.
`to`	numeric: end value of the range.
`widths`	numeric: vector of possible class widths to sample from.
`dmax`	numeric: maximum denominator allowed when checking fractional densities, see `fractions()`.
`maxit`	integer: maximum number of iterations when attempting to find a suitable break pattern.

Value

A list containing:

`breaks`	Numeric vector of class boundaries.
`n`	Integer vector of absolute frequencies for each class.
`decimal`	Logical, `TRUE` if all densities are terminating decimals.
`density`	Numeric vector of class densities.

Examples

```
l <- histwidth(1.6, 2.1, widths = c(0.05, 0.1, 0.15, 0.2))
x <- histx(l$breaks, l$n)
histdata(x, l$breaks)
# Fallback: use constant min(widths) if no valid break pattern
# is found within max iterations
l <- histwidth(1.6, 2.1, widths=0.05, dmax=10)
str(l)
```

`histx` *Midpoint-Based Data Creation for a Histogram*

Description

Given the breaks and the number of observations, a data set is generated with `stats::runif()`, using the class mids: $x_i = class\_mid_j + alpha * class\_width_j/2$. The default `alpha=0.99` ensures that generated observations do not lie on the class borders.

Usage

```
histx(breaks, n, alpha = 0.99)

gen_mid(breaks, n, alpha = 0.99)

dhistx(breaks, n, alpha = 0.99)
```

Arguments

`breaks`	numeric: class borders
`n`	numeric: number of observations in each class
`alpha`	numeric: how far the generated observations can be away from the class mids (default: `0.99`)

Value

The generated data set.

Examples

```
breaks <- sort(sample(seq(0.1, 0.9, by=0.1), 4))
bins   <- length(breaks)-1
n      <- rmultinom(1, size=100, prob=ddiscrete(runif(bins)))
histx(breaks, n)
```

`hm_cell` `html_mmatrix` *Modification*

Description

- `hm_cell` or `hm_index` modify a data cell format (`fmt="%s"`), value (unnamed parameter) or style (`text_align="left"`)
- `hm_col` or `hm_row` modify a row or column format (`fmt="%s"`), value (unnamed parameter) or style (`text_align="left"`)
- `hm_title` modifies the title attribute of an `html_matrix` based on specific arguments
- `hm_table` modifies the properties of the entire HTML table within an `html_matrix`
- `hm_tr` modifies the properties of one or more table rows (tr elements) in an `html_matrix`. Row indices for modification (`ind`) can be specified along with additional parameters to customize the row format, values, or style

Usage

```
hm_cell(x, row = NULL, col = NULL, ..., byrow = FALSE)

hm_index(x, ind, ...)

hm_title(x, ...)

hm_table(x, ...)

hm_row(x, ind, ...)

hm_col(x, ind, ...)

hm_tr(x, ind, ...)

modify_cell(x, row = NULL, col = NULL, ..., byrow = FALSE)

mod_cell(x, row = NULL, col = NULL, ..., byrow = FALSE)

modify_col(x, ind, ...)

mod_col(x, ind, ...)

modify_index(x, ind, ...)

mod_ind(x, ind, ...)

modify_row(x, ind, ...)
```

```
mod_row(x, ind, ...)

modify_table(x, ...)

mod_t(x, ...)

modify_title(x, ...)

mod_title(x, ...)

modify_tr(x, ind, ...)

mod_tr(x, ind, ...)
```

Arguments

`x`	an `html_matrix` object
`row`	integer: row(s) to access
`col`	integer: column(s) to access
`...`	further elements
`byrow`	logical: order indices by row or column (default: `FALSE`)
`ind`	integer vector or matrix: has access to various (row and columns) elements (first column: row, second column: column)

Value

A modified `html_matrix` object.

Examples

```
l <- html_matrix(matrix(1:6, ncol=2))
# replace l[1,1] by NA
hm_cell(l, 1, 1, NA)
# replace l[1,1] by NA and set the text_align to center
hm_cell(l, 1, 1, NA, text_align="center")
# replace l[1,3] and l[2,1] by NA
rcind <- cbind(c(1,3), c(2, 1))
hm_index(l, rcind, NA)
# set a new title
hm_title(l, "new title")
# set a new row or column title
hm_row(l, 2, "row 2")
hm_col(l, 1, "col 1")
# set fmt by column or row
print(hm_cell(l, fmt=c("%.0f", "%.1f", "%.2f"), byrow=FALSE), which="fmt")
print(hm_cell(l, fmt=c("%.0f", "%.1f"), byrow=TRUE), which="fmt")
```

`html_e2m` *HTML* `exams.forge`

Description

Creates an HTML page with all the contents of the XML tags whose names match `pattern`.

The default is to show the contents of all XML tags. The HTML page is stored in the HTML file `name`.

The default `name=NULL` creates a temporary file. If the name does not end in `.html`, then a `.html` is appended.

If `browseURL=TRUE` (default) then the HTML page will be displayed in the browser.

If necessary the contents of XML tags are concatenated with `"\n"`. For single XML tags this can be changed, e.g. `merge=list("questionlist"="<br>"` leads to the XML tag `<questionlist>...</questionlist>)` `"<br>"` being used ,instead of the `"\n"`.

Usage

```
html_e2m(
  exam,
  name = NULL,
  pattern = ".",
  mathjax = TRUE,
  browseURL = TRUE,
  overwrite = FALSE,
  header = 2,
  merge = list(questionlist = "<br>"),
  png = TRUE
)

toHTML_XML(
  exam,
  name = NULL,
  pattern = ".",
  mathjax = TRUE,
  browseURL = TRUE,
  overwrite = FALSE,
  header = 2,
  merge = list(questionlist = "<br>"),
  png = TRUE
)
```

Arguments

`exam`	list: returns a list from `exams.forge`

`name`	character: name of the HTML file (default: `NULL`)
`pattern`	character: string containing a regular expression to match the list elements (default: `.`)
`mathjax`	logical: should MathJax be loaded? (default: `TRUE`)
`browseURL`	logical: should the generated HTML be shown? (default: `TRUE`)
`overwrite`	logical: should the HTML file be overwritten (if it exists)? (default: `FALSE`)
`header`	integer: at which level of the list a `<h2>...</h2>` element should be included? (default: `2`)
`merge`	list: should elements with `.XXXXnn` at the end be merged? (default: `list('questionlist'=" ")`)
`png`	logical: if a entry ends with `.png` then the function will try to embed the PNG in the output

Value

Invisibly, the names of listed elements in the HTML file.

See Also

The aim is similar to `exams:::exams:::browse_exercise`, however, `html_e2m` takes the information form the XML file generated by the `exams.forge` package.

Examples

```
if (interactive()) {
  resexams <- readRDS(system.file("xml", "klausur-test.rds", package="exams.moodle"))
  html_e2m(resexams) # opens HTML file into browser
}
```

`html_matrix` *HTML Representation*

Description

Creates from a vector, a matrix, an array, or a table, an HTML representation of it. The HTML representation has one column and one row more than the data. The additional row and column are used in order to have a title (top left), the column names (top), and the row names (left).

You can set the style attributes (`<td style="...">`) via `hm_cell`, `hm_title`, `hm_col`, and `hm_row`. For example: `hm_cell(hm, 1, 1, text_align="right")` will lead to (`<td style="text-align:right;">`) for the cell (1,1), and any unnamed element will change the cell value. Note: since `-` is an operator in R, we use `_` instead. Of course, someone could use `"text-align"="right"`, but I am lazy.

Usage

```
html_matrix(x, ...)

## Default S3 method:
html_matrix(
  x,
  ...,
  byrow = FALSE,
  numeric = list(text_align = "right"),
  integer = list(text_align = "right"),
  char = list(text_align = "left"),
  logical = list(text_align = "right"),
  border = "#999999"
)

html_mx(x, ...)
```

Arguments

`x`	vector, matrix, array, table or html_matrix: input.
`...`	further parameters
`byrow`	logical: creates a row or column matrix if `x` is one-dimensional (default: `FALSE`)
`numeric`	list: of HTML style properties for a cell if `class(x[i,j])=="numeric"` (default: `list(text_align="right")`)
`integer`	list: of HTML style properties for a cell if `class(x[i,j])=="integer"` (default: `list(text_align="right")`)
`char`	list: of HTML style properties for a cell if `class(x[i,j])=="character"` (default: `list(text_align="left")`)
`logical`	list: of HTML style properties for a cell if `class(x[i,j])=="logical"` (default: `list(text_align="right")`)
`border`	character: vector of background color for a border cell (default: `"#999999"`))

Value

Returns an `html_matrix`.

Examples

```
m <- matrix(1:6, ncol=2)
m
l <- html_matrix(m)
l
```

`html_matrix_sk` `html_matrix` *Object Creation*

Description

My personal pipe creating an `html_matrix` object. Note that the length of `fmt` must be either `nrow(m)` or `ncol(m)` depending on `byrow`.

```
 html_matrix(m)
   tooltip(sprintf(tooltip, nrow(m), ncol(m)))
   hm_cell(fmt=fmt, byrow=byrow)
```

Usage

```
html_matrix_sk(
  m,
  title,
  fmt,
  byrow = TRUE,
  tooltip = "Die Tabelle hat %.0f Zeilen und %.0f Spalten",
  ...
)

lmatrix(
  m,
  title,
  fmt,
  byrow = TRUE,
  tooltip = "Die Tabelle hat %.0f Zeilen und %.0f Spalten",
  ...
)
```

Arguments

`m`	vector, matrix, array, table or html_matrix: input
`title`	character: text for the upper left entry
`fmt`	character: text format for rows (or columns)
`byrow`	logical: `fmt` by row or by column (default: `TRUE`)
`tooltip`	character: text for tooltip with column and row numbers (default: `"Die Tabelle hat %.0f Zeilen und %.0f Spalten"`)
`...`	further parameters given to `html_matrix`

Value

An `html_matrix` object.

Examples

```
m <- matrix(1:6, ncol=2)
html_matrix_sk(m, title="", fmt=c("%.0f", "%.1f"))
```

`hyperloop` *Hyperloop*

Description

Runs a function several times with all parameter combinations, and checks:

- if an argument is not a list, then it will be converted to an one element list
- if an error occurs then the result of `FUN` will not be stored

Usage

```
hyperloop(FUN, ..., .simplify = FALSE)

hloop(FUN, ..., .simplify = FALSE)
```

Arguments

`FUN`	function with named parameter(s)
`...`	named parameters which contain lists with possible parameter values
`.simplify`	logical: should the result be simplified to a data frame (if possible)? (default: `FALSE`)

Value

A `hyperloop` object as a list.

Examples

```
x   <- rnorm(100)
trm <- hyperloop(mean, x=list(x), trim=as.list(seq(0, 0.5, by=0.05)))
# automatic conversion of x to list(x)
trm <- hyperloop(mean, x=x, trim=as.list(seq(0, 0.5, by=0.05)))
unlist(trm)
```

`hyper_param` *Parameters for Hypergeometric Distributions*

Description

Generates a data frame with potential values for `m`, `n` and `k`. If `hyper2` is `FALSE` then the parametrization of `stats::dhyper()` is used, otherwise `n+m`, `m` and `k` is used and transformed to `m`, `n` and `k`. In accordance with specific conditions it holds that:

- if `length(mean)==1` and it's an integer, it signifies the desired number of digits for the mean
- if `mean` is set to `NA` (the default), all means are permissible
- when `length(mean) > 1`, the product $k * m/(n+m)$ must be one of the valid means
- the same rules apply to `sd`

The parameters `norm`, `pois` and `binom` can take on the values `NA`, `TRUE`, `FALSE`, or be defined as a function of the format: `function(m, n, k)`. These values determine which `(m, n, k)` combinations are eligible:

- for `NA`, all combinations of `(m, n, k)` are acceptable
- if specified as a function, only those combinations for which the function evaluates to `TRUE` are considered valid
- if set to `TRUE`, combinations are accepted only if they satisfy either the condition $k * m/(m+n) * (1 - m/(m+n)) >= 9$ (for `norm`, indicating a normal distribution approximation), the conditions $k/(n+m) < 0.05$, $m/(n+m) < 0.05$ and $k > 10$ (for `pois`, implying a Poisson distribution approximation) and the condition $k/(n+m) < 0.05$ (for `binom`, implying a binomial distribution approximation)
- if set to `FALSE`, the approximations should not hold for any combination.

Please be aware that there is no guarantee that the resulting data frame will include a valid solution.

Usage

```
hyper_param(
  m,
  n,
  k,
  mean = NA,
  sd = NA,
  norm = NA,
  pois = NA,
  binom = NA,
  tol = 1e-06,
  hyper2 = FALSE
)
```

Arguments

`m`	numeric: the number of white balls in the urn
`n`	numeric: the number of black balls in the urn
`k`	numeric: the number of balls drawn from the urn, hence must be in $0, 1, ..., m+n$
`mean`	integer or numeric: number of digits the mean should have
`sd`	integer or numeric: number of digits the standard deviation should have
`norm`	logical or function: normal approximation possible
`pois`	logical or function: poisson approximation possible
`binom`	logical or function: binomial approximation possible
`tol`	numeric: the tolerance for numerical comparison (default: '1e-6)
`hyper2`	logical: should the standard R parametrization `(m, n, k)` be used or `(n+m, m, k)`?

Value

A data frame with possible the choices of `n` , `p`, `mean` and `sd`.

Examples

```
hyper_param(7:14, 1:13, 3:10, norm=FALSE, pois=FALSE, binom=FALSE, hyper2=TRUE)
```

`hypothesis_latex` *Latex Hypothesis*

Description

Creates a data frame for a test hypothesis with various columns:

- `h0.left` left value of the null hypothesis, usually `\mu` or `\pi`
- `h0.operator` operator of the null hypothesis, one of the following: `eq`, `ne`, `lt`, `le`, `gt`, or `ge`
- `h0.right` right value of the null hypothesis, usually `\mu_0`, `\pi_0`, or a hypothetical value
- `h1.left` left value of the alternative hypothesis, usually `\mu` or `\pi`
- `h1.operator` operator of the alternative hypothesis, one of the following: `eq`, `ne`, `lt`, `le`, `gt`, or `ge`
- `h1.right` right value of the alternative hypothesis, usually `\mu_0`, `\pi_0`, or a hypothetical value
- `H0` latex representation of the null hypothesis
- `H1` latex representation of the alternative hypothesis
- `match.left` do the left value in the null and the alternative hypothesis match?
- `match.right` do the right value in the null and the alternative hypothesis match?
- `match.operator` do the operators in the null and the alternative hypothesis cover all real numbers?

- `match.right` do the right value in the null and alternative hypothesis match?
- `match.type` either `wrong`, `left.sided`, `right.sided`, `two.sided`, `greater`, or `less`.

If `null` is not given then it is determined from `alternative`. Otherwise hypotheses pairs are generated by all combinations from `alternative` and `null`. Valid values for `alternative`and `null` are `two.sided`, `greater`, `less`, `eq`, `ne`, `lt`, `le`, `gt`, or `ge`.

Usage

```
hypothesis_latex(
  left,
  alternative = NULL,
  null = NULL,
  right = paste0(left, "_0")
)

lhypo(left, alternative = NULL, null = NULL, right = paste0(left, "_0"))
```

Arguments

`left`	character: symbol, for example `"\mu"` or `"\pi"`
`alternative`	character: alternative hypotheses
`null`	character: null hypotheses (default: `NULL`)
`right`	character: a symbol (default: `paste0(left, "_0")`)

Value

A data frame with hypothesis pairs.

Examples

```
# Create one hypotheses pair
hypothesis_latex("\\mu")
hypothesis_latex("\\pi")
hypothesis_latex("\\mu", alternative="two.sided")
hypothesis_latex("\\mu", alternative="two.sided", null="lt")
hypothesis_latex("\\mu", alternative="ne", null="eq")
hypothesis_latex("\\mu", right=c(0,1))
hypothesis_latex("\\mu", alternative=c("eq", "ne", "lt", "le", "gt", "ge"))
hypothesis_latex("\\mu", alternative=c("eq", "ne", "lt", "le", "gt", "ge"),
                         null=c("eq", "ne", "lt", "le", "gt", "ge"))
```

`incomplete_table` *Relative Contingency Table Fill*

Description

Fills a relative contingency table with `n` missing values, such that the table entries can be recomputed. In case that no solution can be found, an error is generated.

Usage

```
incomplete_table(tab, n, maxit = 1000)

cont_table_fill(tab, n, maxit = 1000)
```

Arguments

`tab`	table: a contingency table
`n`	integer: number of missing values
`maxit`	integer: number of maximal iterations (default: `1000`)

Value

A contingency table including marginal values and total sum with missing values. The attribute `fillin` gives the necessary information about the order in which the entries can be calculated, while the attribute `full` presents the contingency table, including marginal values and total sum.

Examples

```
tab <- rbind(c(0.02, 0.04, 0.34), c(0.02, 0.28, 0.3))
incomplete_table(tab, 7)
```

`inline` *Text Knitting*

Description

Knits `txt` within an R code chunk.

Usage

```
inline(txt)

txt_knit(txt)
```

Arguments

`txt`	character

Value

Output.

Examples

```
result <- inline("2 + 2")
```

`is.prob` *Interval Checker*

Description

Checks if `x` is in an opened or closed interval between `min` and `max`. The default is set as such, that the chosen interval is an interval of $(0, 1)$. For example, in the case of `x` being a probability.

Usage

```
is.prob(x, open = TRUE, min = 0, max = 1)

is_prob_interval(x, open = TRUE, min = 0, max = 1)

is_prob(x, open = TRUE, min = 0, max = 1)

in_range(x, open = TRUE, min = 0, max = 1)
```

Arguments

`x`	numeric: values to check
`open`	logical: checks if the left and right borders are open or closed (default: `TRUE`)
`min`	numeric: minimal value (default: `0`)
`max`	numeric: maximal value (default: `1`)

Value

A logical vector with the same length as `x`.

Examples

```
is.prob(runif(1))
```

`knitif` *Knitting a Text Argument*

Description

Selects a text argument and returns the knitted result.

Usage

```
knitif(n, ..., envir = knit_global())

knit_select(n, ..., envir = knit_global())
```

Arguments

`n`	character: text argument to use
`...`	character: arguments to choose from
`envir`	environment: in which code chunks are to be evaluated (default: `[knitr::knit_global]`)

Value

A character.

Examples

```
knitif(runif(1)<0.5, 'TRUE'="`r pi`", 'FALSE'="$\\pi=`r pi`$")
```

`latexdef` *Exam PDF with LaTeX*

Description

If exams is called by `exams2pdf`,

- `latexdef` adds a TeX macro by `\def\name{body}` and
- `answercol` adds a `\def\answercol{n}` to modify the number of output columns for multiple-choice answers to the LaTeX file.

Usage

```
latexdef(name, body)

answercol(n)

add_answercol_def(n)
```

Arguments

`name`	character: macro name
`body, n`	character: macro body

Value

Nothing

Examples

```
answercol(2)
```

`lcmval` *Least Common Multiple*

Description

Computes the least common multiple for a numeric vector `x`.

Usage

```
lcmval(x)

lcm_vector(x)
```

Arguments

`x`	integer: numbers to find the least common multiple

Value

The least common multiple.

Examples

```
lcmval(c(144, 160))      # = 1440
lcmval(c(144, 160, 175)) # = 50.400
```

`lm1_data` *Simple Linear Regression and Data Generation*

Description

Creates data suitable for a simple linear regression. In the first step, data is computed using `pearson_data()`, satisfying the conditions $\sum_{i=1}^{nmax} x_i^2 = n$ and $\sum_{i=1}^{nmax} x_i = 0$ (similar conditions apply to y. The data are then rescaled with $x' = center[1] + scale[1] * x$ and $y' = center[2] + scale[2] * y$. Finally, a simple linear regression is performed on the transformed data.

Usage

```
lm1_data(
  r,
  n = 100,
  nmax = 6,
  maxt = 30,
  xsos = NULL,
  ysos = NULL,
  center = numeric(0),
  scale = numeric(0),
  ...
)

slr_data(
  r,
  n = 100,
  nmax = 6,
  maxt = 30,
  xsos = NULL,
  ysos = NULL,
  center = numeric(0),
  scale = numeric(0),
  ...
)
```

Arguments

`r`	numeric: desired correlation
`n`	integer: number to decompose as sum of squares, see `pearson_data()`.
`nmax`	integer: maximal number of squares in the sum, see `pearson_data()`.
`maxt`	numeric: maximal number of seconds the routine should run, see `pearson_data()`.
`xsos`	sos matrix: precomputed matrix, see `pearson_data()`.
`ysos`	sos matrix: precomputed matrix, see `pearson_data()`.
`center`	numeric(2): center of x and y data

`scale`	numeric(2): standard deviation for `x` and `y` data
`...`	further named parameters given to `stats::lm()`

Value

Returns an extended `lm` object and the additional list elements:

- `inter` contains intermediate results (the last column contains the row sums), and
- `xy` the generated x- and y-values.

Examples

```
data(sos100, package="exams.forge")
n   <- sample(5:10, 1)
lm1 <- lm1_data(0.6, nmax=n, xsos=sos100)
str(lm1)
```

`lmr_data` `lm` *Simple Linear Regression*

Description

Computes an `lm` object for a simple linear regression from a range of `x` and `y` values, including intermediate values. If `r` is not given then zero correlation is used (with `cor_data`). `digits` determines the rounding for the `x` and `y` values. If only one value is given, then it will be used for `x` and `y`. If no value is given then it will be determined from the `x` and `y` values by `3+ceiling(-log10(diff(range(.))))`.

Usage

```
lmr_data(xr, yr, n, r = 0, digits = NULL, ...)

lm_regression_data(xr, yr, n, r = 0, digits = NULL, ...)
```

Arguments

`xr`	numeric: range of x values
`yr`	numeric: range of y values
`n`	numeric: number of observations to generate
`r`	numeric: desired correlation, uses `cor_data`
`digits`	numeric(2): digits for rounding, for x `digits[1]` is used, for y `digits[2]` is used (default: `NULL`)
`...`	further parameters used in `cor_data`

Value

An object of the class `lm` with the additional components:

- `x` the generated x values
- `y` the generated y values
- `sumx` $\sum_{i=1}^n x_i$
- `sumy` $\sum_{i=1}^n y_i$
- `sumx2` $\sum_{i=1}^n x_i^2$
- `sumy2` $\sum_{i=1}^n y_i^2$
- `sumxy` $\sum_{i=1}^n x_i y_i$
- `meanx` the mean of x: $1/n \sum_{i=1}^n x_i$
- `meany` the mean of y: $1/n \sum_{i=1}^n y_i$
- `varx` the variation of x: $\sum_{i=1}^n (x_i - \bar{x})^2$
- `vary` the variation of y: $\sum_{i=1}^n (y_i - \bar{y})^2$
- `varxy` the common variation of x and y:$\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$
- `sxy` the covariance of x and y
- `rxy` the correlation of x and y
- `b0` the intercept of the linear regression
- `b1` the slope of the linear regression
- `r2` the coefficient of determination of the linear regression

Examples

```
# Engine displacement typically ranges from 500 to 2000 cm^3
# Fuel economy typically ranges from 2 to 8 liter/100 km
lmr <- lmr_data(c(500, 2000), c(2, 8), n=8)
str(lmr)
```

`lsumprod` *Supporting Functions for Math LaTeX Output*

Description

`lsumprod` creates a latex printout of $\sum_i x_i y_i$ with brackets if x_i or y_i starts with a `-`.

`lsum` creates a latex printout of x as sum.

`lprod` creates a latex printout of x as product.

`lvec` creates a latex printout of x as vector.

`lmean` creates a latex printout as $\frac{x_1+\ldots+x_n}{n}$.

`lvar` creates a latex printout as $\frac{(x_1-xbar)^2+\ldots+(x_n-xbar)^2}{n}$.

`lbr` creates a latex printout of x with brackets if x starts with a `-`.

`lsgn` creates a latex printout of x with a plus or minus at the beginning.

Usage

```
lsumprod(..., br = "(")

lsum(x)

lprod(x)

lvec(
  x,
  left = c("(", "[", "{", "|", "||", "<", "a", "c", "f"),
  right = NULL,
  collapse = ", "
)

lmean(x)

lvar(x, mu = NULL, br = "(")

lbr(x, br = c("(", "[", "{", "|", "||", "<", "a", "c", "f"), subset = NULL)

lsgn(x)

latex_sumprod(..., br = "(")

latex_sum(x)

latex_product(x)

latex_mean(x)

latex_var(x, mu = NULL, br = "(")

latex_bracket(
  x,
  br = c("(", "[", "{", "|", "||", "<", "a", "c", "f"),
  subset = NULL
)

latex_pmsign(x)
```

Arguments

`...`	further input values
`br`, `left`, `right`	character: which brackets to use. The possibilities are: • `(` (default) uses `\left(` and `\right(`, • `[` use `\left[` and `\right]`, • `{` use `\left\{` and `\right\}`,

- `|` use `\left|` and `\right|`,
- `||` uses `\left\|` and `\right\|`,
- `<`, `a` use `\left\langle` and `\right\rangle`,
- `c` use `\left\lceil` and `\right\rceil`, and
- `f` use `\left\lfloor` and `\right\rfloor`.

`x`	numeric: input values
`collapse`	character: an optional character string to separate the results (default: ', ')
`mu`	numeric: population mean (default: `NULL`)
`subset`	logical: indicates which elements have brackets added (default: `NULL` = all elements starting with `-`); missing values are taken as false.

Value

A character.

Examples

```
lsumprod(-2:2, (1:5)/10)
lbr(-2:2)
lsum(-2:2)
lmean(-2:2)
lvec(-2:2)
lvec(-2:2, '[')
lvec(0:1, '(', ']')
```

`makekey` *Character Key Generation*

Description

Generates a character key from a vector of integers.

Usage

```
makekey(index)

make_key(index)
```

Arguments

`index`	integer: vector of integer

Value

A character.

Examples

```
makekey(1)
makekey(1:2)
makekey(pi) # ;)
makekey(c(5,4))
```

`mcval`	*Most Common Value*

Description

Computes all modes (most common value).

Usage

```
mcval(x, ...)

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

## S3 method for class 'histogram'
mcval(x, exact = FALSE, ...)

compute_modes(x, ...)

mcv(x, ...)
```

Arguments

`x`	data object
`...`	further arguments
`exact`	logical: either compute the exact mode or use class mids (default: `FALSE`)

Value

A vector of modes.

Examples

```
x <- sample(1:5, 15, replace=TRUE)
mcval(x)
```

`meanint_data` *Integer Observations and Mean*

Description

The `meanint_data` function generates a set of integer observations with a specified integer mean. It takes the number of observations or `x` values and an optional range parameter, `r`, that defines the permissible range of x values (defaulting to the range of `x`). Additional parameters are passed to the `mean` function. The function employs a iterative process, adjusting individual observations to achieve an integer mean. It uses a random selection approach, modifying a randomly chosen observation and checking if the resulting mean is closer to an integer. The process continues until the mean becomes an integer.

Usage

```
meanint_data(x, r = range(x), ...)
```

Arguments

`x`	numeric: number of observations or x values
`r`	numeric: the range in which the x values allowed (default: `range(x)`)
`...`	further parameters given to `mean`

Value

A set of integer observations with an integer mean.

Examples

```
x <- meanint_data(10, c(1, 10))
mean(x)
```

`means_choice` *Means*

Description

Computes the means of `x`. The list returned has an attribute `"mindiff"` which contains the smallest distance between two mean values before rounding. If `winsor` and/or `trim` is set to `NA` then the trimmed and/or winsorized means are not computed. Currently implemented are:

`mean`	arithmetic mean
`median`	median
`harmonic`	harmonic mean
`geometric`	geometric mean

`mode` (first) mode

`trim` trimmed mean

`winsor` winsorized mean

Usage

```
means_choice(x, digits, na.rm = TRUE, trim = 0.2, winsor = 0.2)

means(x, digits, na.rm = TRUE, trim = 0.2, winsor = 0.2)
```

Arguments

`x`	numeric: data values
`digits`	numeric: integer indicating the number of decimal points for rounding (negative values are allowed)
`na.rm`	logical: should `NA`s be removed before?
`trim`	numeric: the fraction (0 to 0.5) of observations to be trimmed from each end of `x`
`winsor`	numeric: the fraction (0 to 0.5) of observations to be moved from each end of `x`

Value

A list with mean values.

Examples

```
x <- c(runif(9), 3)
means_choice(x, 2)
```

`mime_image` *MIME Image*

Description

Returns the MIME type of an image based on the `filename` extension. If a MIME type for a file extension cannot not found, then the extension itself will be returned.

Usage

```
mime_image(filename)

mime_img(filename)
```

Arguments

`filename`	character: file name

Value

A character.

Examples

```
mime_image("support.png")
mime_image("support.jpg")
```

`monomial` *Monomial*

Description

Creates a polynomial of the form $c * x^d$.

Usage

```
monomial(degree = 1, coefficient = 1)

monom(degree = 1, coefficient = 1)
```

Arguments

`degree`	integer: degree of the polynomial (default: 1)
`coefficient`	numeric: coefficient of the polynomial (default: 1)

Value

A polynomial

Examples

```
monomial()     # equivalent to polynomial()
monomial(3)    # x^3
monomial(3, 2) # 2*x^3
```

`moodle_m2s` *Moodle Multiple-Choice*

Description

The `exams` package does not support multiple-choice questions with multiple correct answers; it only allows for one answer to be chosen. However, Moodle does support such questions. The function reads the XML file generated by `exams.forge` and makes changes for all `mchoice` questions:

- `<single>...</single>` to `<single>true</single>`, and
- modifies the attribute `fraction` in the tags `<answer fraction="...">...</answer>`. If `fraction` is less than 0, it is set to zero, and if `fraction` is greater than 0, it is set to 100.

If the `file` does not end with `.xml`, then `.xml` is appended. At the end, the modified XML code is stored in `newfile`.

Usage

```
moodle_m2s(file, newfile = NULL, verbose = 1)

mchoice_moodle(file, newfile = NULL, verbose = 1)
```

Arguments

`file`	character: Moodle XML file with exercises to read from
`newfile`	character: Moodle XML file to write in (default: `file`)
`verbose`	integer: output generation (default: `1`)

Value

Invisibly, the written file name.

Examples

```
if (interactive()) {
  newfile <- tempfile(fileext=".xml")
 moodle_m2s(system.file("xml", "klausur-test.xml", package="exams.moodle"), newfile=newfile)
  file.edit(newfile)
}
```

`nearest_arg` *Nearest Candidate Value*

Description

It determines the nearest candidate value for each value in `arg`. As a replacement for `[base::match.arg]`, it is more error-tolerant, but detecting a wrong choice can be proven challenging.

Usage

```
nearest_arg(arg, choices, method = "cosine", ...)
```

Arguments

`arg`	character: vector or NULL
`choices`	character: vector of candidate values
`method`	character: method for distance calculation (default: `cosine`)
`...`	further parameters for stringdist::stringdistmatrix

Value

For each value in `arg` the (first) nearest element of `choices`.

Examples

```
# match.arg("tow.sided", c("two.sided", "less", "greater")) # will fail
nearest_arg("tow.sided", c("two.sided", "less", "greater"))
nearest_arg(c("two.sided", "less", "greater"), c("two.sided", "less", "greater"))
nearest_arg(c("two", "two", "ded", "ss", "ea"), c("two.sided", "less", "greater"))
```

`nosanitize` *Sanitization*

Description

`nosanitize` makes no sanitization on the strings.

Usage

```
nosanitize(str)
```

Arguments

`str`	character: vector to sanitize

Value

A sanitized character vector.

Examples

```
nosanitize("Test")
```

`now` *Current Time*

Description

Returns a time stamp based on the current time. `now` basically calls `gsub('.', '', sprintf('%.20f', as.numeric(Sys.time())), fixed=TRUE)`. To ensure that at each call a different time stamp is delivered `now` may call `gsub(...)` several times until two different results are delivered. The last one is then returned.

Usage

```
now(last = 35)
```

Arguments

`last`	integer: the amount of digits that should be returned (default: `35`)

Value

A character.

Examples

```
now()   # returns all digits
now(3)  # returns only the first three digits
```

`nsprintf` *sprintf with template depending on integer valued* `n`

Description

`nsprintf` creates a text dependent on the value(s) in `n`. In particular, we have

- `round_de`, it returns either `Runden Sie Ihr Ergebnis auf eine ganze Zahl`, `Runden Sie Ihr Ergebnis auf eine S` or `Runden Sie Ihr Ergebnis auf n Stellen nach dem Komma`
- `schoice_de` returns `Es kann eine oder mehrere Antworten richtig sein. Es ist ausreichend, eine richtige`

Usage

```
nsprintf(n, ...)

round_de(n)

schoice_de()

print_de(n, ...)
```

Arguments

`n`	integer: number(s) to be used
`...`	character: format strings to be used

Value

`sprintf`ed strings

Examples

```
nsprintf(0, '0' = "keine Netzunterbrechung", '1' = "eine Netzunterbrechung",
            "%i Netzunterbrechungen")
nsprintf(0:3, `0` = "keine Netzunterbrechung", `1` = "eine Netzunterbrechung",
              "%i Netzunterbrechungen")
```

`num2str` *Number to String Conversion*

Description

Converts a set of numeric variables to a list as string representation, either as decimal or as a fractional number.

Usage

```
num2str(..., denom = -1)
```

Arguments

`...`	numeric variables
`denom`	integer: denominator for fractional number

Value

A list.

Examples

```
x <- 1
l <- num2str(x)        # returns in l$x the string representation
l <- num2str(x, y=x+1)  # returns in l$x and l$y the string representations
```

`num_result` *Numeric Rounding List*

Description

`num_result` creates a list summarizing numeric values with rounding and tolerance. It returns the original values, rounded values, the digits used, and the tolerance.

The rounding is done with the internal function `fmt()` (similar to `exams::fmt()`). Users can pass additional arguments to `fmt()` via `...`.

- `x`: original numeric values
- `fx`: rounded values as character (via `fmt()`)
- `tolerance`: numeric tolerance for comparison
- `digits`: digits used for rounding

If `digits` is not provided:

- If `length(x)` > 1, `ceiling(-log10(min(diff(sort(x)), na.rm = TRUE)))` is used.
- If `length(x)` == 1, 3 + `ceiling(-log10(abs(x)))` is used.

If `tolerance` is not provided, it defaults to `tolmult * 10^(1 - digits)`.

`int_result()` is a shortcut for integer values (`digits = 0`, `tolerance = 0.1`).

Usage

```
num_result(x, digits = NULL, tolerance = NULL, tolmult = 2, ...)

int_result(x, ...)

num_res(x, digits = NULL, tolerance = NULL, tolmult = 2, ...)

int_res(x, ...)
```

Arguments

`x`	numeric: the input values
`digits`	numeric: number of digits to round to (default: `NULL`)
`tolerance`	numeric: optional numeric tolerance (default: `NULL`)
`tolmult`	numeric: multiplier for tolerance calculation (default: 2)
`...`	further arguments passed to `fmt()`. Common arguments include: • `digits`: number of digits for formatting • `zeros`: logical; pad with trailing zeros (default TRUE for digits < 4)

Value

A list with elements `x`, `fx`, `tolerance`, and `digits`.

Examples

```
# Example: numeric values
x <- rnorm(10, mean = 1.8, sd = 0.25)
num_result(c(mean(x), x), digits = 2)

# Example: integer result
int_result(mean(x))

# Example: different digits and tolerance
num_result(pi, 3)
num_result(pi, 6)
num_result(pi, 6, tolmult = 5)
num_result(pi, 6, tolmult = 5, tolerance = 1e-6)
```

`num_solve` *Target Variable Value*

Description

Given a set of equations and some variables, `num_solve` tries to compute the value of the `target` variable. The equations $y = f(x)$ are transformed to $f(x) - y$ and the functions try to compute the roots of the equations using [`stats::uniroot()`]. If the computation fails, then, `numeric(0)` is returned, otherwise the "original" value. If `target==''` then all computed values and steps are returned. The attribute `compute` contains a data frame.

`toLatex.equation_solve` returns a LaTeX representation of the solution way found by `num_solve()`.

Usage

```
num_solve(target, eqs, tol = 1e-06)

## S3 method for class 'equation_solve'
toLatex(object, ...)

sequation(target, eqs, tol = 1e-06)
```

Arguments

`target`	character: name of the variable value to compute
`eqs`	an `equations` object
`tol`	numeric: maximal tolerance for `stats::uniroot()`
`object`	object of a class for which a `toBibtex` or `toLatex` method exists.
`...`	further arguments

Value

(for `num_solve`) Returns `numeric(0)`, `numeric(1)`, or a list of all (computed) values.

(For `toLatex.equation_solve`) A character vector.

Examples

```
# The equations describe the formulae for an confidence interval of the mean
e <- equations(o~x+c*s/sqrt(n), "v_o=\\bar{x}+c\\cdot\\frac{s^2}{n}",
               u~x-c*s/sqrt(n), "v_u=\\bar{x}-c\\cdot\\frac{s^2}{n}",
               e~c*s/sqrt(n),   "e  =c\\cdot\\frac{s^2}{\\sqrt{n}}",
               l~2*e,           "l  =2\\cdot e"
               )
e <- variables(e,
               x=0,     "\\bar{x}",
               c=2.58, dbl(2),
               s=1,     pos(5), "s^2",
               n=25,    pos(5),
               l=pos(5),
               e=pos(5),
               u="v_u", o="v_o")
print(e)
# Find the confidence interval length
ns <- num_solve('l', e)
# Compute everything that is possible
 ns <- num_solve('', e)
toLatex(ns)
```

`open_files` *Open Multiple Files with Optional Menu Selection*

Description

The `open_files` function allows users to manage multiple files by either opening them interactively or reading their contents programmatically. It supports interactive selection through a menu for multiple file paths and adapts to the availability of the `rstudioapi` package for enhanced integration with the RStudio IDE.

Usage

```
open_files(file, open = interactive(), ...)
```

Arguments

`file`	character: vector of file paths, where each path specifies the location of a file to be opened or read; paths may be absolute or relative
`open`	logical: Whether to open the files interactively (`TRUE`) or read their contents programmatically (`FALSE`). Defaults to the result of `interactive()`.

`...`	Additional arguments to be passed to the file opening or editing function; these arguments are passed either to `rstudioapi::navigateToFile()` or `utils::edit()`, depending on the environment and the availability of the `rstudioapi` package.

Details

When the `file` argument contains more than one file path, the function calculates the longest common subsequence (LCS) of the directory structures for all provided file paths and presents the user with an interactive menu. The menu allows selection of files to open; the options include "All," "None," or individual files that match the common directory pattern. In non-interactive sessions, the function reads the content of the files instead.

Internally, the function determines whether the `rstudioapi` package is available, in which case it uses `navigateToFile` for opening files within the RStudio IDE; otherwise, it falls back to the base R function `edit`. In non-interactive sessions, the function reads file contents using `readLines` and stores them in a named list, with filenames as the names of the list elements.

The auxiliary `lcss` function calculates the longest common subsequence of directory components for the file paths; this helps in simplifying the user experience by grouping files with similar paths.

This documentation was created with the support of ChatGPT.

Value

Returns an invisible named list. Each element corresponds to a file from the input vector, with the value representing the result of the operation. When opening files, the result indicates success or failure. When reading files, the result contains the file's contents. If "None" is selected, the function returns `NULL`.

Examples

```
if (interactive()) {
  files <- pkg.files("poylnomials", package="polynom")
  open_files(files[2]) # open one file
  open_files(files)    # open several files
}
```

`pdensity` *Density Function*

Description

Creates a linear (`power=1`) or constant (`power=0`) density function in a interval

$$[a, b]$$

where `a` and `b` are sampled from `x`. It samples `size` elements without replacement and computes the value of the distribution function.

Usage

```
pdensity(x, size = 3, power = 1, tol = 1e-06)

sample_density(x, size = 3, power = 1, tol = 1e-06)
```

Arguments

`x`	numeric: range of density with $a = min(x, na.rm = TRUE)$ and $b = max(x, na.rm = TRUE)$
`size`	numeric: number of elements to be sampled (without replacement) from x
`power`	numeric: constant or linear density function
`tol`	numeric: disallow for density coefficients near zero (default: `1e-6`). A negative value will permit zero coefficients.

Value

A list with:

- `a` the minimum of the interval
- `i` the maximum of the interval
- `x` the `size` sampled values
- `fx` the distribution function at `x`
- `pcoeff` a polynomial (intercept = first value)
- `qcoeff` indefinite integral of the polynomial (intercept = first value)
- `pint` result of the `integral(pcoeff, c(a,b), 0:2)`

Examples

```
pdensity(-5:5)
pdensity(-5:5, power=1)
```

`pearson_data` *Pearson Data*

Description

Generates an integer data set for computing a correlation using `sumofsquares()`. If `n>100` and `nmax>6` it is better to use one of the precomputed solutions. Otherwise it may take up to `maxt` seconds. Please note that the correlation of the generated data set may differ from the desired correlation.

Usage

```
pearson_data(r, n = 100, nmax = 6, maxt = 30, xsos = NULL, ysos = NULL)

dpearson(r, n = 100, nmax = 6, maxt = 30, xsos = NULL, ysos = NULL)
```

Arguments

`r`	numeric: desired correlation
`n`	integer: number to decompose as sum of squares, see `sumofsquares()`.
`nmax`	integer: maximal number of squares in the sum, see `sumofsquares()`.
`maxt`	numeric: maximal number of seconds the routine should run, see `sumofsquares()`.
`xsos`	sos matrix: precomputed matrix
`ysos`	sos matrix: precomputed matrix

Value

A matrix with two columns and an attribute `interim` for intermediate values as matrix. The rows of the matrix contain : x_i, y_i, $x_i - barx$, $y_i - \bar{y}$, $(x_i - barx)^2$, $(y_i - \bar{y})^2$, and $(x_i - barx)((y_i - \bar{y})$. In a final step, a vector with the row of sums is appended as a further column.

Examples

```
data(sos100, package="exams.forge")
xy <- pearson_data(0.7, xsos=sos100)
colSums(xy)
colSums(xy^2)
sum(xy[,1]*xy[,2])
# my data
x <- 100+5*xy[,1]
y <- 100+5*xy[,2]
cor(x, y)
```

`pminimum` *Polynomial Minimum*

Description

Computes the minimum of a polynomial in the interval $[lower, upper]$. The values and the interval borders of the polynomial p are evaluated and the minimum value is returned.

Usage

```
pminimum(
  p,
  interval,
  lower = min(interval),
  upper = max(interval),
  tol = 1e-09
)

polynomial_minimum(
  p,
```

```
  interval,
  lower = min(interval),
  upper = max(interval),
  tol = 1e-09
)
```

Arguments

`p`	polynomial
`interval`	numeric: a vector containing the end-points of the interval to be searched for the minimum
`lower`	numeric: the lower end point of the interval to be searched (default: `min(interval)`)
`upper`	numeric: the upper end point of the interval to be searched (default: `max(interval)`)
`tol`	numeric: the desired accuracy (default: `1e-9`)

Value

The minimal function value.

Examples

```
p <- polynomial(c(-5, 3, -3, 1))
pminimum(p, -3, 3)
```

`pos` *Interval Ranges*

Description

Generates intervals based on powers of ten.

Usage

```
pos(pow)

neg(pow)

dbl(pow)

idbl(pow)

ipos(pow)

ineg(pow)
```

Arguments

`pow`	numeric: power of ten to create intervals

Value

A `numeric` object.

Examples

```
dbl(2)
dbl(3)
pos(3)
neg(3)
```

`pprobability` *Polynomial Probability*

Description

Creates for each value of a discrete random variable, a polynomial and estimates the least squares and the maximum likelihood solution. The following conditions stand:

- If `sample` is not given then the sample contains each `x` value once.
- If `sample` is an integer, then it is interpreted as the sample size and a sample is generated by `rmultinom(1, sample, ddiscrete(runif(length(x))))`.
- If `sample` is a vector, it is interpreted in such a way that the corresponding `x[i]` value occurs `i` times in the sample. Thus, `sum(sample)` is the sample size.
- If `coeff` is a `polylist` of `length(x)`, then these polynomials are taken.
- If `coeff` is a `matrix` with `length(x)`, columns and `power+1` rows, then the columns are interpreted as the coefficients of a polynomial.
- Otherwise `coeff` is interpreted as a vector from which the coefficient is sampled. The intercepts are sampled via `ddiscrete(runif(length(x)), zero=zero)`. If `coeff` is not given then it is ensured that the least squares and the maximum likelihood solution exists and the estimated probabilities are between zero and one. Otherwise, the results may contain `NA` or the estimated probabilities are outside the interval $[0; 1]$.

Usage

```
pprobability(
  x,
  power = 1,
  zero = FALSE,
  coef = round(seq(-1, 1, by = 0.1), 1),
  sample = rep(1, length(x)),
  pl = NULL,
  tol = 1e-09
)

polynomial_probability(
  x,
```

```
  power = 1,
  zero = FALSE,
  coef = round(seq(-1, 1, by = 0.1), 1),
  sample = rep(1, length(x)),
  pl = NULL,
  tol = 1e-09
)
```

Arguments

`x`	numeric: values of a discrete random variable
`power`	integer: the degree for the polynomials (default: `1`), must be larger 0
`zero`	logical: are zero coefficients and zero samples allowed? (default: `FALSE`)
`coef`	matrix: for each degree coefficients to sample from (default: `seq(-1, 1, by=0.1)`)
`sample`	integer: number of x values in the sample or sample size (default: `rep(1, length(x))`)
`pl`	polylist: a list of polynomials which describes the probability for x (default: `NULL`)
`tol`	numeric: tolerance to detect zero values (default: `1e-9`)

Value

A list with the components:

- `p`: the polynomials for the probabilities
- `ep`: the expected value as polynomial
- `x`: the values for the discrete random variable, the same as the input `x`
- `sample`: the sample given or generated
- `LS$pi`: the summands for the least squares problem
- `LS$pl`: the summands for the least squares problem in LaTeX
- `LS$pf`: the sum of `LS$pi`
- `LS$df`: the derivative of `LS$pf`
- `LS$pest`: the estimated parameter, minimum of `LS$pf`
- `LS$p`: the estimated probabilities
- `ML$pi`: the factors for the maximum likelihood problem
- `ML$pl`: the summands for the maximum likelihood problem in LaTeX
- `ML$pf`: the product of `ML$pi`
- `ML$df`: the derivative of `ML$pf`
- `ML$pest`: the estimated parameter, maximum of `ML$pf`
- `ML$p`: the estimated probabilities

Examples

```
# linear polynomials
pprobability(0:2)
pprobability(0:2, power=1)
# constant polynomials, some NAs are generated
pprobability(0:3, power=0)
# polynomials generated from a different set
pprobability(0:2, coef=seq(-2, 2, by=0.1))
pprobability(0:2, 0, coef=seq(-2, 2, by=0.1))
# polynomials (x, x, 1-2*x) are used
pprobability(0:2, 0, coef=matrix(c(0.4, 0.4, 0.3), ncol=3))
pprobability(0:2, 1, coef=polylist(c(0,1), c(0,1), c(1, -2)))
```

`print.equations` *print.equations*

Description

Prints an `equations` object with equations and variables. Internally, a data frame is generated, created and printed.

Usage

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

Arguments

`x`	an object used to select a method.
`...`	further arguments passed to or from other methods.

Value

The data frame invisibly generated.

Examples

```
# The equations describe the formulae for an confidence interval of the mean
e <- equations(o~x+c*s/sqrt(n), "v_o=\\bar{x}+c\\cdot\\frac{s^2}{n}",
               u~x-c*s/sqrt(n), "v_u=\\bar{x}-c\\cdot\\frac{s^2}{n}",
               e~c*s/sqrt(n),   "e  =c\\cdot\\frac{s^2}{\\sqrt{n}}",
               l~2*e,           "l  =2\\cdot e"
               )
print(e)
```

`print.html_matrix` *Print* `html_matrix`

Description

Prints an HTML matrix content or its components.

Usage

```
## S3 method for class 'html_matrix'
print(x, ..., which = "")
```

Arguments

`x`	an html_matrix object
`...`	further parameters
`which`	character: which component to print (default: `""`)

Value

An invisible character matrix.

Examples

```
m <- matrix(1:6, ncol=2)
l <- html_matrix_sk(m, title="1 to 6", fmt=rep("%f",ncol(m)))
print(l, which=NA)      # returns full style information
print(l, which="fmt")   # returns format information
print(l, which="value") # identical to print(l)
```

`prob_solve` *Total or Conditional Probability Computation*

Description

The following functions are available:

- `prob_solve` given a set of events it computes the total or conditional probability of the given event or `NA` if no solution could be found. For the naming of the events upper case letters must be used and the available operators are `!` (complementary event), `|` (conditional event), and `^` (intersection of events). The attribute `latex` of the return value contains the necessary computation steps for computation of the given event. If `getprob` is `TRUE` then additionally the attribute `prob`, a vector with all computed probabilities, and `compute`, which includes all computational steps, are generated.
- `print` shows the solution way in ASCII.
- `toLatex` shows the solution way in LaTeX/MathJax with an `align` environment.
- `lprob` converts `!A` to `\\bar{A}` and `A^B` to `A \\cap B`.

Usage

```
prob_solve(target, ...)

## Default S3 method:
prob_solve(target, ..., partition = NULL, getprob = FALSE, quiet = TRUE)

lprob(txt)

## S3 method for class 'prob_solve'
toLatex(object, ...)

## S3 method for class 'prob_solve'
print(x, type = c("numeric", "latex", "prob", "compute"), ...)

latex_prob(txt)

probability_solution(target, ...)

sprob(target, ...)
```

Arguments

`target`	character: target event
`...`	numeric: named events with given probabilities
`partition`	character or list: set of events which form a partition
`getprob`	logical: return all computed probabilities and used computation steps (default: `FALSE`)
`quiet`	logical: show all computation steps (default: `FALSE`)
`txt`	character: vector to convert ! to `\\bar` and `^` to '\cap'
`object`, `x`	`prob_solve` object
`type`	character: what to print, either `numeric` (solution, default), `latex` (solution steps in ASCII format), `prob` (optional: all probabilities computed), or `compute` (optional: all rules used)

Details

The program applies iteratively the following rules to find a solution:

- $P(A) = 1 - P(!A)$,
- $P(A|B) = 1 - P(!A|B)$,
- $P(A^B) = P(B^A)$,
- $P(B) = P(A^B) + P(!A^B)$,
- $P(A|B) = P(A^B)/P(B)$, and
- $P(A) = P(A|P1) + P(A|P2) + ... + P(A|Pn)$ for a partition $P1, P2, ..., Pn$.

Value

An object of the class `prob_solve` with the resulting probability, including the steps for computing. If `NA` is returned then no solution could be found.

Examples

```
prob_solve("!A", "A"=0.3)
prob_solve("!A|B", "A|B"=0.3)
prob_solve("B^A", "A^B"=0.3)
# P(B)   = P(A^B)+P(!A^B)
prob_solve("B", "A^B"=0.3, "!A^B"= 0.4)
prob_solve("A^B", "B"=0.7, "!A^B"= 0.4)
prob_solve("!A^B", "B"=0.7, "A^B"= 0.3)
# P(A|B) = P(A^B)/P(B)
prob_solve("A|B", "A^B"=0.3, "B"= 0.6)
prob_solve("A^B", "B"=0.6, "A|B"= 0.5)
prob_solve("B", "A|B"=0.5, "A^B"= 0.3)
#' latex, prob and compute attributes
pmt <- prob_solve("M|T", "M"=0.6, "T|M"=0.75, "T|!M"=0.39, quiet=FALSE, getprob=TRUE)
toLatex(pmt)
attr(pmt, "latex")
pmt <- prob_solve("M|T", "M"=0.6, "T|M"=0.75, "T|!M"=0.39, quiet=FALSE, getprob=TRUE)
attr(pmt, "prob")
print(pmt, "latex")
print(pmt, "prob")    # only if getprob=TRUE
print(pmt, "compute") # only if getprob=TRUE
# bayes theorem and total probability
prob_solve("Z", "Z|A"=0.1, "Z|B"=0.2, "Z|C"=0.3, partition=c("A", "B", "C"))
prob_solve("Z|A", "Z"=0.6, "Z|B"=0.2, "Z|C"=0.3, partition=c("A", "B", "C"))
prob_solve('A|K', "A"=0.55, "B"=0.35, "C"=0.1, "K|A"=0.4, "K|B"=0.1, "K|C"=0.1,
           partition=c("A", "B", "C"))
prob_solve('K', "A"=0.55, "B"=0.35, "C"=0.1, "K|A"=0.4, "K|B"=0.1, "K|C"=0.1,
           partition=c("A", "B", "C"))
```

`proptests` *Proportion Tests*

Description

`proptests` runs a bunch of modifications of the input parameters of `proptest` to generate all possible proportion tests. See under "Details" the detailed parameter values which are used. Note that not giving the parameter `hyperloop` will results in several hundred tests generated. Only the distinct tests will be returned, with the first element being `proptest`. If only a specific element of a `proptests` is of interest, provide the name of the element in `elem`. All `proptests` will then be returned where the value of `elem` is different.

Usage

```
proptests(proptest, elem = NULL, hyperloop = NULL)
```

Arguments

`proptest`	proptest: the base result from a valid t-test generated by `proptest_num()`
`elem`	character: element to extract (default: `NULL`)
`hyperloop`	named list: parameter values to run over (default: see above)

Details

The default `hyperloop` is:

```
list(x           = c(proptest$x, proptest$n-proptest$x)
   pi0      = c(proptest$pi0, 1-proptest$pi0, proptest$x/proptest$n, 1-proptest$x/proptest$n)
     alpha       = unique(c(proptest$alpha, 0.01, 0.05, 0.1)),
     alternative = c("two.sided", "greater", "less")
   )
```

Value

list of `proptest` objects is returned

Examples

```
basetest  <- proptest_num(x=3, n=8, alternative="greater")
# vary the number of observations
hyperloop <- list(pi0 = c(basetest$pi0, 1-basetest$pi0,
                          basetest$x/basetest$n, 1-basetest$x/basetest$n))
# return all different tests
tts       <- proptests(basetest, hyperloop=hyperloop)
# return all different random sampling functions
proptests(basetest, "X", hyperloop)
```

`proptest_data` *Binomial Test Data Creation*

Description

Creates data for a binomial test based on the properties for the test.

Usage

```
proptest_data(
  size = 10:100,
  prob = seq(0.05, 0.45, by = 0.05),
  reject = TRUE,
  alternative = c("two.sided", "less", "greater"),
  alpha = c(0.01, 0.05, 0.1),
  norm.approx = NA,
  maxit = 1000
```

```
)

prop_binomtest_data(
  size = 10:100,
  prob = seq(0.05, 0.45, by = 0.05),
  reject = TRUE,
  alternative = c("two.sided", "less", "greater"),
  alpha = c(0.01, 0.05, 0.1),
  norm.approx = NA,
  maxit = 1000
)

dbinomtest(
  size = 10:100,
  prob = seq(0.05, 0.45, by = 0.05),
  reject = TRUE,
  alternative = c("two.sided", "less", "greater"),
  alpha = c(0.01, 0.05, 0.1),
  norm.approx = NA,
  maxit = 1000
)
```

Arguments

`size`	numeric: vector of sample sizes (default `10:100`)
`prob`	numeric: vector of probabilities for the hypothetical proportion π_0 (default =`seq(0.05, 0.45, by=0.05)`)
`reject`	logical: should `x` generate a lead for the rejection of the null hypothesis (default `TRUE`), if equals `NA` then this will be ignored
`alternative`	character: a character string specifying the alternative hypothesis, must be one of `two.sided` (default), `greater` or `less`
`alpha`	numeric: vector of significance levels (default `c(0.01, 0.05, 0.1)`)
`norm.approx`	logical: should a normal approximation be possible $(size*prob*(1-prob) > 9)$
`maxit`	integer: maximal numbers of trials to find a solution (default `1000`)

Value

A list with the components:

- `pi0` hypothetical proportion
- `x` counts of successes in the sample
- `n` sample size
- `alpha` significance level
- `alternative` specifying the alternative hypothesis (either `two.sided`, `greater` or `less`)

Examples

```
proptest_data()
```

`proptest_num` *Proportion Tests*

Description

Computes all results for test on proportion using either `stats::binom.test()`, or a normal approximation without continuity correction. Either named parameters can be given or an `arglist` with the following parameters:

- `x` number of successes
- `n` sample size (default: `sd(x)`)
- `pi0` true value of the proportion (default: `0.5`)
- `alternative` a string specifying the alternative hypothesis (default: `"two.sided"`), otherwise `"greater"` or `"less"` can be used
- `alpha` significance level (default: `0.05`)
- `binom2norm` can the binomial distribution be approximated by a normal distribution? (default: `NA` = use `binom2norm` function)

Usage

```
proptest_num(..., arglist = NULL)

prop_binomtest_num(..., arglist = NULL)

nbinomtest(..., arglist = NULL)
```

Arguments

`...`	named input parameters
`arglist`	list: named input parameters, if given `...` will be ignored

Details

The results of `proptest_num` may differ from `stats::binom.test()`. `proptest_num` is designed to return results when you compute a binomial test by hand. For example, for computing the test statistic the approximation $t_n \approx N(0;1)$ is used if $n > n.tapprox$. The `p.value` is computed by stats::binom.test and may not be reliable, for Details see Note!

Value

A list with the input parameters and the following:

- `X` distribution of the random sampling function
- `Statistic` distribution of the test statistics
- `statistic` test value

- `critical` critical value(s)
- `criticalx` critical value(s) in x range
- `acceptance0` acceptance interval for H0
- `acceptance0x` acceptance interval for H0 in x range
- `accept1` is H1 accepted?
- `p.value` p value for test (note: the p-value may not be reliable see Notes!)
- `alphaexact` exact significance level
- `stderr` standard error of the proportion used as denominator

Note

The computation of a p-value for non-symmetric distribution is not well defined, see `https://stats.stackexchange.com/questions/140107/p-value-in-a-two-tail-test-with-asymmetric-null-distribut`

Examples

```
n <- 100
x <- sum(runif(n)<0.4)
proptest_num(x=x, n=n)
```

`q2norm` *Mean and Standard Deviation for Normal Distribution*

Description

Given two (or more) quantiles it computes an (approximate) mean and standard deviation for a corresponding normal distribution.

Usage

```
q2norm(x, probs = c(0.025, 0.975))
```

Arguments

`x`	numeric(2): the quantiles
`probs`	numeric(2): probabilities with values in $[0, 1]$ (default: `c(0.025, 0.975)`)

Value

A list with a component `mean` and `sd`.

Examples

```
q2norm(c(100,200))
```

`random` *Random*

Description

Returns a index from `1:length(v)` randomly ordered.

Usage

```
random(v)

rand(v)
```

Arguments

`v`	vector: vector with elements

Value

Index

Examples

```
random(-3:3)
```

`refer` *Generate Vector Element Names*

Description

Creates names for elements of a vector.

Usage

```
refer(x, fmt = "%s_{%.0f}", to = deparse(substitute(x)), index = 1:length(x))

refer2vector(
  x,
  fmt = "%s_{%.0f}",
  to = deparse(substitute(x)),
  index = 1:length(x)
)
```

Arguments

`x`	vector: a vector to create the names for
`fmt`	character: format string for `sprintf` (default: `"%s_{%.0f}"`
`to`	character: base name of elements
`index`	numeric: vector with indices (default: `1:length(x)`)

Value

A character vector

Examples

```
x <- runif(5)
refer(x)                  # LaTeX default
refer(x, fmt="%s[%.0f]")  # R default
```

`replace_fmt` *Replace*

Description

In a text it replaces names with:

- values which are formatted with `exams::fmt()`, or
- strings

Usage

```
replace_fmt(txt, digits = 2L, ...)
```

Arguments

`txt`	character: text where the replacement is done
`digits`	numeric or list: number of digits to round
`...`	names to replace with values

Value

A character with replaced names.

Examples

```
replace_fmt("\\frac{x}{y}", x=2, y=3)
replace_fmt("\\frac{x}{y}", x=2, y=3, digits=0)
replace_fmt("\\frac{x}{y}", x=2, y=3, digits=list(0))
replace_fmt("\\frac{x}{y}", x=2, y=3, digits=list(2, y=0))
replace_fmt("\\frac{x}{y}", x="\\\\sum_{i=1}^n x_i", y="\\\\sum_{i=1}^n y_i")
```

`rv` *Random Variable*

Description

Formats a random variable and its meaning for R Markdown.

Usage

```
rv(symbol, explanation)

rmdFormatRV(symbol, explanation)

lrv(symbol, explanation)
```

Arguments

`symbol`	character: symbol
`explanation`	character: meaning

Value

A formatted string.

Examples

```
rv("X", "Waiting time in minutes until next event")
```

`sample_size_freq` *Sample Size Consistency Checker*

Description

Checks if a vector of possible sample sizes and relative frequencies create integer absolute frequencies.

Usage

```
sample_size_freq(n, f, which = NA)

dnsizefreq(n, f, which = NA)
```

Arguments

`n`	numeric: vector of sample size(s) to check
`f`	numeric: vector of relative frequencies
`which`	numeric: if several `n`'s are possible then `which` is returned (default: `NA` = choose a random one)

Value

One sample size.

Examples

```
f <- ddiscrete(runif(5), unit=100)
sample_size_freq(seq(10, 200, 1), f)
sample_size_freq(seq(10, 200, 1), f, which=200)
```

`scale_to` *Rescaling*

Description

Rescales `x` such that for the rescaled data it holds: `mean(scale_to(x, mean=target))==target` and `sd(scale_to(x, sd=target)==abs(target)`. A negative value of `sd` will change the sign of the `x` values.

Usage

```
scale_to(x, mean = 0, sd = 1)
```

Arguments

`x`	numeric: vector of values
`mean`	numeric: mean of the rescaled `x` (default: `0`)
`sd`	numeric: standard deviation of the transformed `x` (default: `1`)

Value

Rescaled data.

Examples

```
x <- runif(50)
y <- scale_to(x, mean=0.1, sd=0.2)
mean(y)
sd(y)
y <- scale_to(x, mean=0.1, sd=-0.2)
mean(y)
sd(y)
```

`select_menu` *Display a Menu for User Selection*

Description

Presents a list of choices to the user, allowing them to select one or more options by entering numbers. If the user inputs an empty line, the function skips the selection and returns `NULL`.

Usage

```
select_menu(
  choices,
  title = NULL,
  msg = "Enter one or more numbers, or an empty line to open files: ",
  width = getOption("width")
)
```

Arguments

`choices`	character: vector of options to display to the user. Each choice will be prefixed with a numbered label.
`title`	character: optional string to display as the title of the menu. Default is `NULL`, meaning no title is displayed.
`msg`	character: string specifying the prompt message for user input. Default is `"Enter one or more numbers, or an empty line to open files: "`.
`width`	integer: specifying the maximum line width for displaying the menu. Defaults to the value of the `width` option (`getOption("width")`).

Details

This documentation was created with the support of ChatGPT.

Value

A character vector of selected choices, or `NULL` if the user enters an empty line.

See Also

Based on `select_menu()` in the package `remotes`

Examples

```
## Not run:
# Example usage:
options <- c("Apple", "Banana", "Cherry", "Date")
selected <- select_menu(options, title = "Choose your fruits:")
cat("You selected:", paste(selected, collapse = ", "), "\n")
```

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

`skalenniveau` *Skalenniveau*

Description

A data frame with the variables and level of measurement type. The names are in German.

Usage

```
data(skalenniveau)
```

Format

A data frame with columns `var`, and `type`.

Examples

```
data(skalenniveau)
head(skalenniveau)
```

`solution` *Solutions*

Description

Creates a `solution` object and prints a meta information block for the following:

- `solution` the default is `sol_num`
- `sol_num` for a numerical solution
- `sol_int` for an integer solution
- `sol_mc` for a multiple choice solution
- `sol_ans` for the answer list of a multiple choice solution
- `sol_tf` for the solution list (True or False) of a multiple choice solution
- `sol_info` for creating a Meta-Information block

Usage

```
solution(x, ...)

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

sol_int(x, tol = NA, digits = NA)

sol_num(x, tol = NA, digits = NA)

sol_mc(x, y, sample = NULL, shuffle = order, none = NULL)

sol_ans(x, ...)

sol_tf(x, ...)

sol_info(x, ...)

sol_mc_ans(x, ...)

sol_meta(x, ...)

sol_mc_tf(x, ...)
```

Arguments

`x`	numeric solution or false MC solutions
`...`	further parameters
`tol`	numeric: tolerance for a numeric solution (default: `NA`)
`digits`	integer: number of digits for rounding (default: `NA`)
`y`	true MC solutions
`sample`	integer: sampling numbers for false and/or true solutions (default: `NULL`)
`shuffle`	logical or function: shuffling or ordering of solutions (default `order`)
`none`	character: if you do not wish to choose any of the false and/or true solutions offered (default: `NULL`)

Details

For numerical solutions you can set `tol` and/or `digits`. If they are not set, they are automatically selected. If `tol` is not set and `length(x)>1` then the tolerance is chosen as `min(diff(sort(x)))/2`. Otherwise, as `max(0.001, 0.001*abs(x))`. I If `tol` is negative, `tolerance` is set to `10^tol`, otherwise it is used as it is. If `digits` is not set, `ceiling(-log10(tolerance))` is used.

Value

A `solution` object.

Examples

```
s <- sol_num(pi)
sol_info(s)
# set same tolerances, e.g. for a probability
sol_num(0.1)
sol_num(0.1, tol=0.001)
sol_num(0.1, tol=-3)
# MC: Which are prime numbers?
prime <- c(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
nonprime <- setdiff(2:30, prime)
# choose five false and two correct solutions
s <- sol_mc(nonprime, prime, sample=c(5,2), none="There are no prime numbers in the list")
sol_ans(s)
sol_tf(s)
sol_info(s)
```

`sos100` *Precomputed Sum of Squared Data*

Description

Five data matrices with precomputed results from `sumofsquares(n, 10, zerosum=TRUE, maxt=Inf)` for `n=100`, `n=200`, `n=400`, and `n=800`.

More generally, datasets can be accessed using `sos(n, nmax)`, which retrieves or computes results from `sumofsquares(n, nmax, zerosum=TRUE, maxt=Inf)`.

The function `sos()` retrieves a dataset of the form `sosN`, where `N` is the input parameter `n`. The function tries several sources in order:

1. Load from the package `exams.forge` if available.
2. Load from the user data directory (see `R_user_dir`).
3. Download from a GitHub repository of `exams.forge`.
4. Compute the dataset on the fly using `sumofsquares()`.

Usage

```
data(sos100)
data(sos200)
data(sos400)
data(sos800)

sos200

sos400

sos800

sos(n, nmax = 10, quiet = !interactive(), download = interactive())
```

Arguments

`n`	Integer, specifying the dataset index (e.g., `n = 3` returns `sos3`).
`nmax`	Integer, the maximum value passed to `sumofsquares()`. Defaults to `10`.
`quiet`	Logical. If `FALSE`, progress messages are printed. Defaults to `!interactive()`.
`download`	Logical. If `TRUE`, the function attempts to download the data from the GitHub repository of `exams.forge`. Defaults to `interactive()`.

Format

For each line of a matrix it holds $\sum_{i=1}^k x_i^2 = n$ and $\sum_{i=1}^k x_i = 0$. It contains all integer solutions up to `k<=10`. `NA` means that this entry is not used.

Details

In steps 3 and 4, the dataset is cached in the user data directory (compressed with `xz`). If the directory does not exist and the session is interactive, the user is prompted for permission to create it; otherwise a temporary directory is used. In non-interactive sessions, the directory is used only if it already exists, otherwise `tempdir()` is used silently.

The computation step calls:

```
sumofsquares(n, nmax, maxt = Inf, zerosum = TRUE)
```

Value

An R object named `sosN`, corresponding to the requested dataset. If no solution exists for the given `n`, a matrix with a single row filled with `NA` values is returned.

See Also

`R_user_dir`, `sumofsquares`

Examples

```
data(sos100)
head(sos100)
rowSums(sos100^2, na.rm=TRUE)
rowSums(sos100, na.rm=TRUE)
x100 <- sos(100) # part of the package
rowSums(x100, na.rm=TRUE)     # only zeros
rowSums(x100^2, na.rm=TRUE)   # only 100's
x10 <- sos(10)   # must be computed
rowSums(x10, na.rm=TRUE)     # only zeros
rowSums(x10^2, na.rm=TRUE)   # only 144's
```

`spell` *RMarkdown Spell Check*

Description

Performs a spell check on RMarkdown files ignoring some `exams` keywords using `spelling::spell_check_files()`.

Usage

```
spell(
  path,
 ignore = c("Meta", "information", "extype", "num", "mchoice", "schoice", "Solution",
   "exsolution", "extol", "exname", "Question", "align", "begin", "bigg", "cases",
    "cdot", "end", "frac", "infty", "int", "left", "left.", "leq", "mu", "qquad",
     "right", "sum", "text", "vert"),
  lang = Sys.getenv("LANG")
)

rm_spell_check(
  path,
 ignore = c("Meta", "information", "extype", "num", "mchoice", "schoice", "Solution",
   "exsolution", "extol", "exname", "Question", "align", "begin", "bigg", "cases",
    "cdot", "end", "frac", "infty", "int", "left", "left.", "leq", "mu", "qquad",
     "right", "sum", "text", "vert"),
  lang = Sys.getenv("LANG")
)
```

Arguments

`path`	path to file or to spell check
`ignore`	character vector with words which will be added to the hunspell::dictionary
`lang`	set `Language` field in `DESCRIPTION` e.g. `"en-US"` or `"en-GB"`. For supporting other languages, see the hunspell vignette.

Value

A data frame with problematic words.

Examples

```
# none
```

`sqrtnp` *Calculating Square Roots of* np*(1-p) Combinations*

Description

Computes `sqrt(n*p*(1-p))` for all combinations of `n` and `p`. If the result has only `digits` after the decimal point, then `n`, `p`, and `sqrt(n*p*(1-p))` are returned in a data frame.

Usage

```
sqrtnp(n, p, digits = 2, tol = 10^(-digits - 4))
```

Arguments

`n`	numeric: vector of observations numbers
`p`	numeric: vector of probabilities
`digits`	numeric: number of digits to check (default: `2`)
`tol`	numeric: tolerance (default: `10^(-digits-4)`)

Details

If `abs(v-round(v, digits))<tol` then a number `v` is considered as a number with only `digits` after the decimal point.

Value

A data frame with the columns `n`, `p`, `np` ($= np$) and `snp` ($= sqrt(np(1-p))$).

Examples

```
n <- 30:250
p <- (10:40)/100
sqrtnp(n, p)
```

`sumofsquares` *Sum of Squared Integers*

Description

Decomposes an integer `n` into a sum of squared integers ($n = \sum_{i=1}^{k} x_i^2;\ 1 \leq x_i < n$) with $k \leq nmax$. If `zerosum` is true then it is ensured that $\sum_{i=1}^{k} c_i x_i = 0$ with $c_i = -1$ or $c_i = +1$. The computation of the x_i's is limited by `maxt` seconds, which may result that not all possible solutions are found. To reduce computing time, `rbind`'s in the function are replaced by allocating matrices with `size` rows to fill in the results. Note that the following data sets are available:

- `sos100=sumofsquares(100, 10, zerosum=TRUE, maxt=Inf)`,
- `sos200=sumofsquares(200, 10, zerosum=TRUE, maxt=Inf)`,
- `sos400=sumofsquares(400, 10, zerosum=TRUE, maxt=Inf)`, and
- `sos800=sumofsquares(800, 10, zerosum=TRUE, maxt=Inf)`

Usage

```
sumofsquares(n, nmax = 10, zerosum = FALSE, maxt = 30, size = 100000L)

sum_sq(n, nmax = 10, zerosum = FALSE, maxt = 30, size = 100000L)
```

Arguments

`n`	integer: number to decompose as sum of squares
`nmax`	integer: maximum number of squares in the sum
`zerosum`	logical: should the solution sum up to one (default: `FALSE`)
`maxt`	numeric: maximal number of seconds the routine should run
`size`	numeric: length of additional matrix size (default: `100000L`)

Value

A matrix with `nmax` column with x_i's. `NA` means number has not been used. If no solution exists for the given `n`, a matrix with a single row filled with `NA` values is returned.

Examples

```
sos <- sumofsquares(100, 6) # 23 solutions
head(sos)
table(rowSums(!is.na(sos)))
# one solution with one or two x_i
# five solutions with four x_i
# six solutions with five x_i
# ten solutions with six x_i
rowSums(sos^2, na.rm=TRUE)  # all 100
sos <- sumofsquares(100, 6, zerosum=TRUE)
head(sos)
rowSums(sos^2, na.rm=TRUE)  # all 100
rowSums(sos, na.rm=TRUE)    # all 0
```

`sumofsquares1` *sumofsquares1*

Description

Decomposes an integer `n2` into a sum of squared integers ($n2 = \sum_{i=1}^{nobs} x_i^2$). If `n` is not `NA` then it is ensured that $\sum_{i=1}^{nobs} x_i = 0$. Note if `nobs<=10` then the following data sets are available:

- `sos100=sumofsquares(100, 10, zerosum=TRUE, maxt=Inf)`,
- `sos200=sumofsquares(200, 10, zerosum=TRUE, maxt=Inf)`,
- `sos400=sumofsquares(400, 10, zerosum=TRUE, maxt=Inf)`, and
- `sos800=sumofsquares(800, 10, zerosum=TRUE, maxt=Inf)`

Usage

```
sumofsquares1(n2, nobs = 10, n = 0, x = runif(nobs), maxit = 1000)
```

Arguments

`n2`	integer: number to decompose as sum of squares
`nobs`	integer: length of return values
`n`	integer: additional sum condition (default: `0`)
`x`	numeric: vector of `nobs` starting values (default: `runif(nobs)`)
`maxit`	integer: maximal number of iterations

Value

A integer vector of length `nobs`.

Examples

```
sumofsquares1(100, 20)
sumofsquares1(100, 20)
```

`t2norm` *Distribution Approximations*

Description

These functions check whether a normal approximation is appropriate for a given distribution. They return `TRUE` if the approximation condition is met, and `FALSE` otherwise. The threshold parameter `c` can be set directly or retrieved via `getOption()`.

The functions apply the following rules:

- `t2norm`: `n > c` with default `c = 30`.
- `binom2norm`:
  - If `type = "single"` (default), the approximation is valid if `size × prob × (1 - prob) > c`.
  - If `type = "double"`, the approximation requires both `size × prob > c` and `size × (1 - prob) > c`, with default `c = 9`.
- `clt2norm`: `n > c` with default `c = 30`. Note that the existence of expectation and variance, required by the Central Limit Theorem, cannot be checked automatically.

Usage

```
t2norm(n, c = getOption("distribution.t2norm", 30))

binom2norm(
  size,
  prob,
  c = getOption("distribution.binom2norm", 9),
  type = c("single", "double")
)

clt2norm(n, c = getOption("distribution.clt2norm", 30))

approx_binom2norm(
  size,
  prob,
  c = getOption("distribution.binom2norm", 9),
  type = c("single", "double")
)

approx_clt2norm(n, c = getOption("distribution.clt2norm", 30))

approx_t2norm(n, c = getOption("distribution.t2norm", 30))
```

Arguments

`n`	integer: number of observations (for `t2norm` and `clt2norm`)

`c`	numeric: threshold parameter for approximation (default via `getOption()` or a default value)
`size`	integer: number of trials (for `binom2norm`)
`prob`	numeric: probability of success on each trial (for `binom2norm`)
`type`	character: approximation type, `"single"` or `"double"` (for `binom2norm`)

Value

logical: `TRUE` if the approximation is valid, `FALSE` otherwise

Examples

```
# Check for 5 and 50 observations
t2norm(n = c(5, 50))
binom2norm(size = c(5, 50), prob = 0.5)
binom2norm(size = c(5, 50), prob = 0.5, type = "double")
```

`table_data` *Frequency Table*

Description

Creates a frequency table where all entries can be written as $2^{p_{ij}}5^{q_{ij}}$. It holds that $p_{ij} < m2$ and $q_{ij} < m5$. If the algorithm does not find a solution, then an error is thrown. Try to increase `unit` to 20, 50, 100 and so on. Once a table is found, the table is normalized by dividing all entries by a number such that the entries are still integer. Finally, a multiplicator of the form 2^p5^5 is randomly chosen, ensuring that the sum of the entries is less than, or equal to `n`.

Usage

```
table_data(
  nrow,
  ncol,
  unit = 10,
  maxit = 1000,
  n = 100,
  m2 = ceiling(log(n)/log(2)),
  m5 = ceiling(log(n)/log(5))
)

freq_table(
  nrow,
  ncol,
  unit = 10,
  maxit = 1000,
  n = 100,
  m2 = ceiling(log(n)/log(2)),
```

```
  m5 = ceiling(log(n)/log(5))
)

dtable(
  nrow,
  ncol,
  unit = 10,
  maxit = 1000,
  n = 100,
  m2 = ceiling(log(n)/log(2)),
  m5 = ceiling(log(n)/log(5))
)
```

Arguments

`nrow`	integer: number of rows
`ncol`	integer: number of columns
`unit`	integer: reciprocal of smallest non-zero probability (default: `10`)
`maxit`	integer: maximal number of iterations (default: `1000`)
`n`	integer: maximal sum of table entries (default: `100`)
`m2`	integer: maximal power of two used on normalized the table (default: `ceiling(log(n)/log(2))`)
`m5`	integer: maximal power of five used on normalized the table (default: `ceiling(log(n)/log(5))`)

Value

A frequency table where all entries can be written as $2^{p_{ij}}5^{q_{ij}}$.

Examples

```
tab22 <- table(2, 2)
tab22
divisor_25(tab22)
nom.cc(tab22)        # Should be zero
#
table(3, 2)
table(4, 2)
```

`template` *Template*

Description

A text template where R code can be embedded.

Usage

```
template(tmpl, ...)
```

Arguments

`tmpl`	character: template
`...`	named parameter used in the template

Value

A character where the R code is replaced by its evaluation.

Examples

```
tmpl <- "`r a`+`r b`"
template(tmpl, a=1, b=2)
```

`toHTML.html_matrix` *Export Matrices as HTML or LaTeX*

Description

Convert a matrix to a formatted representation in HTML or LaTeX:

- `toHTML`: Returns an HTML table of a matrix. Optionally displays it in a browser via a temporary file using `utils::browseURL()`.
- `toLatex`: Returns a LaTeX table representation. Supports a subset of style options.
- `toHTMLorLatex`: Chooses HTML or LaTeX output based on the presence of `exams2pdf` in the call stack.

Usage

```
## S3 method for class 'html_matrix'
toHTML(x, browser = FALSE, delay = 2, ...)

## S3 method for class 'html_matrix'
toLatex(object, ...)

toHTMLorLatex(x, ...)
```

Arguments

`x, object`	An object of class `html_matrix`.
`browser`	Logical; if `TRUE`, the HTML output is opened in a browser (default `FALSE`). Only used by `toHTML`.
`delay`	Numeric; seconds to wait before deleting temporary HTML files. A value of `0` keeps the file until the R session ends. Only used by `toHTML`.
`...`	Additional arguments passed to `utils::browseURL()` (only relevant for `toHTML`).

Value

A character string containing the HTML or LaTeX representation of the matrix.

Examples

```
library("tools")
m  <- matrix(1:12, ncol = 4)
hm <- html_matrix(m)
# toHTML(hm, browser = TRUE) # opens into browser
toHTML(hm)
toLatex(hm)
toHTMLorLatex(hm)
```

`toLatex.polynomial` *LaTeX Representation of a Polynomial*

Description

Returns a LaTeX representation of the polynomial.

Usage

```
## S3 method for class 'polynomial'
toLatex(
  object,
  digits = TRUE,
  decreasing = FALSE,
  variable = "x",
  simplify = TRUE,
  tol = 1e-09,
  ...
)
```

Arguments

`object`	polynomial
`digits`	numeric or logical: how to convert to text (default: `NA`)
`decreasing`	logical: order of the terms by increasing or decreasing powers (default: `FALSE`)
`variable`	character: name of variable used (default: `"x"`)
`simplify`	logical: should the polynomial representation be simplified (default: `TRUE`)
`tol`	numeric: tolerance (default: `1e-9`). A negative value will keep zeros and ones too, but: • If a coefficient is smaller than `tol` then zero terms are not kept • If a absolute value of coefficient minus one is smaller than `tol` then coefficient is not kept
`...`	unused parameters

Value

A character

Examples

```
p <- polynomial(c(-1,0,2)/3)
toLatex(p, 4)
toLatex(p, FALSE)
toLatex(p, TRUE)
toLatex(p, variable="z")
toLatex(p, decreasing=TRUE)
p <- polynomial(c(0,1,2)/3)
toLatex(p)
toLatex(p, tol=-1)
```

`tooltip` *Tooltip*

Description

Adds a text tooltip to the HTML matrix.

Usage

```
tooltip(x, tooltip = NULL)

add_tooltip(x, tooltip = NULL)
```

Arguments

`x`	an html_matrix object
`tooltip`	character: text to show (default: `NULL`)

Value

An html_matrix object

Examples

```
library("magrittr")
library("tools")
m    <- matrix(1:12, ncol=4)
hm   <- html_matrix_sk(m, title='', fmt=rep("%f", ncol(m))) %>%
          add_tooltip(sprintf("Table has %0.f rows and %0.f columns", nrow(.), ncol(.)))
if (interactive()) html <- toHTML(hm, browser=TRUE)
```

---

toRMarkdown *toRMarkdown*

Description

Conversion to R Markdown.

Usage

```
toRMarkdown(txt)
```

Arguments

`txt`	character: vector with lines of Moodle Markdown

Value

Lines with RMarkdown

Examples

```
txt <- c("[image]\n",
        "Ein Paar hat 8 gute Bekannte, von denen die beiden 5 zum Essen einladen möchten.",
      "Wie viele verschiedene Reihenfolgen des Eintreffens der eingeladenen 5 Gäste gibt es?\n",
          ": 56",
          "; 120",
          "; 336",
          "; 2002",
          "; 6720",
          "; 32768",
          "; 40320",
          "; Keine Antwort ist richtig")
toRMarkdown(txt)
```

---

toString.polynomial *Text Representation of a Polynomial*

Description

Creates a text representation for a polynomial, in the following scenarios:

- if `digits` is `TRUE` then `as.character(.)` is used
- if `digits` is `FALSE` then `./.` is used
- if `digits` is numeric then `as.character(round(., digits))` is used

Usage

```
## S3 method for class 'polynomial'
toString(
  x,
  digits = TRUE,
  decreasing = FALSE,
  variable = "x",
  simplify = TRUE,
  tol = 1e-09,
  ...
)
```

Arguments

`x`	polynomial: vector of coefficients (first is intercept)
`digits`	numeric or logical: how to convert to text (default: `NA`)
`decreasing`	logical: order of the terms by increasing or decreasing powers (default: `FALSE`)
`variable`	character: name of the variable used (default: `"x"`)
`simplify`	logical: should the polynomial representation be simplified (default: `TRUE`)
`tol`	numeric: tolerance (default: `1e-9`). A negative value will keep zeros and ones too, but: • If a coefficient is smaller than `tol` then zero terms are not kept. • If a absolute value of coefficient minus one is smaller than `tol` then coefficient is not kept
`...`	unused parameters

Value

A character

Examples

```
p <- polynomial(c(-1,0,2)/3)
toString(p, 4)
toString(p, FALSE)
toString(p, TRUE)
toString(p, variable="z")
toString(p, decreasing=TRUE)
p <- polynomial(c(0,1,2)/3)
toString(p)
toString(p, tol=-1)
```

`to_choice` *Questions and Solutions List Generation*

Description

Creates a list with the elements `questions` and `solutions` values. A value can be either an entry in a vector or a row in a data frame. `correct` is a logical vector which contains `TRUE` if its value represents a correct answer and `FALSE` if it represents a wrong answer. The values can be shuffled or ordered (default).

If `shuffle` is a integer of length 1 then one correct answer is chosen, and `shuffle` wrong answers are chosen. If `shuffle` is a integer of length larger than 1, then `shuffle[1]` correct answers are chosen and `shuffle[2]` wrong answers are chosen. If any `shuffle` entry is zero or negative, then no shuffling will be done. If `order` is a function then it is expected that the function delivers an index for the reordering of the values. Otherwise a `shuffle` for all values is applied.

The shuffling works in two steps:

1. Sample within the correct and wrong value according to `shuffle`
2. Apply shuffling (`order=NULL`) or ordering (default: `order=order`) of all selected answers

Usage

```
to_choice(
  df,
  correct,
  shuffle = c(NA_integer_, NA_integer_),
  orderfun = order,
  ...
)

choice_list(
  df,
  correct,
  shuffle = c(NA_integer_, NA_integer_),
  orderfun = order,
  ...
)
```

Arguments

`df`	vector or data frame: values, in a data frame each row holds one value
`correct`	logical: answer is correct (`TRUE`) or not (`FALSE`)
`shuffle`	integer: the numbers of correct and wrong values to shuffle (default: `c(NA,NA)`). `NA` means no shuffling
`orderfun`	function: ordering of the shuffled values (default: `order`)
`...`	further named parameters used in `shuffle`

Value

list with questions and solutions

Examples

```
answer   <- runif(5)
correct  <- (1:5)==3 # Third answer is correct, the rest wrong
sc       <- to_choice(answer, correct)
str(sc)           # Answers are ordered by size
sc$questions <- c(format(sc$questions, nsmall=2), "No answer is correct") # Additional answer
sc$solutions <- c(sc$solutions, FALSE)                                    # TRUE or FALSE?
sc           <- to_choice(answer, correct, shuffle=2)
str(sc)      # One correct answer and two wrong answers selected
```

`transformif` *Transformation*

Description

Transforms `x` if `cond` is `TRUE` by $\log(a+b*x)$ if `p==0` and $(a+b*x)^p)$. Otherwise the transformation can be either applied to each element of `x`, or to all elements of `x`.

Usage

```
transformif(x, cond, a = -abs(min(x)), b = 1, p = 1)
```

Arguments

`x`	vector: values
`cond`	logical: condition if transformation should be applied
`a`	numeric: shift (default: `-abs(min(x))`)
`b`	numeric: scale (default: `1`))
`p`	numeric: power (default: `1`))

Value

A transformed vector

Examples

```
x <- rnorm(5)
transformif(x, min(x)<0)  # all transformed elements > 0
transformif(x, x<0)       # only negative elements are transformed
```

ts_data *Time Series*

Description

Creates an univariate time series based on a linear or an exponential trend, an additive or multiplicative seasonal adjustment and with white noise.

Usage

```
ts_data(
  end,
  trend = TRUE,
  trend.coeff = c(1, 1),
  season = TRUE,
  season.coeff = NULL,
  error = TRUE,
  error.coeff = NULL,
  digits = NA
)

dts(
  end,
  trend = TRUE,
  trend.coeff = c(1, 1),
  season = TRUE,
  season.coeff = NULL,
  error = TRUE,
  error.coeff = NULL,
  digits = NA
)
```

Arguments

`end`	integer: length of time series
`trend`	logical: if `TRUE` a linear trend otherwise a exponential trend (default: `TRUE`)
`trend.coeff`	numeric: coefficients for a linear model (default: `c(1,1)`)
`season`	logical: if `TRUE` an additive seasonal adjustment is done otherwise, a multiplicative seasonal adjustment (default: `TRUE`)
`season.coeff`	numeric: coefficients for the adjustment (default: `NULL`). If `NULL` then no seasonal adjustment is made.
`error`	logical: if `TRUE` an additive error term is used, otherwise, a multiplicative error term (default: `TRUE`).
`error.coeff`	numeric: standard deviation(s) for white noise error (default: `NULL`). If `NULL` then no error is added.

`digits`	integer: number of digits to round the time series (default: `NA`). If `NA` then no rounding is done.

Value

A ts_data object with the following list of elements:

- `t` the time points
- `s` the season for the time points
- `xt` the time series values

Examples

```
# Time series from linear trend
ts <- ts_data(12, trend.coeff= c(sample(0:10, 1), sample(1+(1:10)/20, 1)))
ts
# Time series from exponential trend
ts <- ts_data(12, trend.coeff= c(sample(0:10, 1), sample(1+(1:10)/20, 1)), trend=FALSE)
ts
# Time series from linear trend and additive seasonal adjustment (quartely data)
ts <- ts_data(12, trend.coeff=c(sample(0:10, 1), sample(1+(1:10)/20, 1)),
                  season.coeff=sample((-20:20)/20, 4))
ts
# Time series from linear trend and additive seasonal adjustment (half-yearly data)
ts <- ts_data(12, trend.coeff=c(sample(0:10, 1), sample(1+(1:10)/20, 1)),
                  season.coeff=sample((-20:20)/20, 2))
ts
# Time series from linear trend and mutliplicative seasonal adjustment (quartely data)
ts <- ts_data(12, trend.coeff=c(sample(0:10, 1), sample(1+(1:10)/20, 1)),
                  season.coeff=sample((-20:20)/20, 4), season=FALSE)
ts
```

`ts_moving_average` *Moving Average*

Description

Computes the moving average for a `ts_data` object.

Usage

```
ts_moving_average(ts, order)

ts_ma(ts, order)
```

Arguments

`ts`	a `ts_data` object
`order`	integer: order of the moving average

Value

Returns an extended `ts_data` object with list elements:

- `filter` the filter used
- `moving.average` the computed moving average

Examples

```
# trend from a quadratic model
ts <- ts_data(12, trend.coeff=c(sample(0:10, 1), sample(1+(1:10)/20, 1), 0.5))
ts_moving_average(ts, 3)
```

`ts_trend_season` *Trend and Season Model*

Description

Estimate a trend and season model from a `ts_data` object.

Usage

```
ts_trend_season(ts, trend = NULL, season = NULL)

ts_ts(ts, trend = NULL, season = NULL)
```

Arguments

`ts`	`ts_data` object
`trend`	numeric or logical: if `trend` is TRUE then a linear trend will be estimated, otherwise an exponential trend. If `trend` is numeric this is considered as trend value
`season`	numeric or logical

Value

Returns an extended `ts_data` object with the following list of elements:

- `t` the time points
- `s` the season for the time points
- `xt` the time series values
- `trend` the fitted trend values
- `trend.coeff` the trend coefficients
- `trend.linear` the trend type, if `NA` then it is unknown
- `season` the fitted season values
- `season.t` the fitted season values for the time series

- `trend.season` the fitted values for trend and season
- `trend.linear` the trend type, if `NA` then it is unknown
- `var` the variance of the residuals
- `r.square` the R^2 of the final model

Examples

```
ts <- ts_data(12, trend.coeff= c(sample(0:10, 1), sample(1+(1:10)/20, 1)))
ts_trend_season(ts)
```

`ttests` *T-tests*

Description

`ttests` runs a variety of modifications to the input parameters of `ttest`, in order to generate all possible t-tests. See under "Details" the detailed parameter values which are used. Note that not giving the parameter `hyperloop` will results in approx. 5000 t-tests generated. Returned will be only the different t-tests with the first element being `ttest`. If only a specific element of a `ttest` is of interest then just give the name of the element in `elem` and then all `ttests` will be returned where `elem` is different.

Usage

```
ttests(ttest, elem = NULL, hyperloop = NULL)
```

Arguments

`ttest`	ttest: the base result from a valid t-test generated by `ttest_num()`
`elem`	character: element to extract (default: `NULL`)
`hyperloop`	named list: parameter values to run over (default: see above)

Details

The default `hyperloop` is:

```
list(n           = c(1, ttest$n, ttest$n+1),
     mu0         = c(ttest$mu0, ttest$mean),
     mean        = c(ttest$mu0, ttest$mean),
     sigma       = c(ttest$sigma, ttest$sd, sqrt(ttest$sigma), sqrt(ttest$sd)),
     sd          = c(ttest$sigma, ttest$sd, sqrt(ttest$sigma), sqrt(ttest$sd)),
     norm        = c(TRUE, FALSE),
     alpha       = unique(c(ttest$alpha, 0.01, 0.05, 0.1)),
     alternative = c("two.sided", "greater", "less")
    )
```

Value

A list of `ttest` objects is returned

Examples

```
basetest  <- ttest_num(mean=0.5, sd=1.25, n=50, sigma=1)
# vary the number of observations
hyperloop <- list(n=c(1, basetest$n, basetest$n^2))
# return all different t-tests
tts       <- ttests(basetest, hyperloop=hyperloop)
# return all different random sampling functions
ttests(basetest, "Xbar", hyperloop)
```

`ttest_data` *T-tests and Data Creation*

Description

Creates data for a t-test, for one mean, based on the test's properties.

Usage

```
ttest_data(
  size = (3:20)^2,
  mean = -5:5,
  sd = seq(0.1, 1, by = 0.1),
  reject = NA,
  alternative = c("two.sided", "less", "greater"),
  alpha = c(0.01, 0.05, 0.1),
  z = seq(-4.49, 4.49, by = 0.01),
  use.sigma = TRUE
)

dt1(
  size = (3:20)^2,
  mean = -5:5,
  sd = seq(0.1, 1, by = 0.1),
  reject = NA,
  alternative = c("two.sided", "less", "greater"),
  alpha = c(0.01, 0.05, 0.1),
  z = seq(-4.49, 4.49, by = 0.01),
  use.sigma = TRUE
)
```

Arguments

`size`	numeric: vector of possible sample sizes (default `(3:20)^2,`)
`mean`	numeric: vector of possible means (default `-5:5`)
`sd`	numeric: vector of possible standard deviations (default `sd=seq(0.1, 1, by=0.1`)
`reject`	logical: should `x` generate a lead for the rejection of the null hypothesis (default `TRUE`), if equals `NA` then this will be ignored
`alternative`	character: a character string specifying the alternative hypothesis, must be one of `two.sided` (default), `greater` or `less`
`alpha`	numeric: vector of significance levels (default `c(0.01, 0.05, 0.1)`)
`z`	numeric: vector of possible z values (default `seq(-4.49, 4.49, by=0.01)`)
`use.sigma`	logical: should the standard deviation of the population (default) or the sample be used?

Value

A list with the components:

- `mu0` hypothetical mean
- `sigma` standard deviation in the population
- `sd` vector of possible standard deviations in the sample
- `xbar` mean in the sample
- `n` sample size
- `alpha` significance level
- `alternative` specifying the alternative hypothesis (either `two.sided`, `greater` or `less`)
- `altsd` alternative values usable for `sd` (if `use.sigma==TRUE`) or `sigma` (if `use.sigma==FALSE`)

Examples

```
ttest_data()
```

`ttest_num` *T-tests*

Description

Computes all results for a t-test. Note that the results may differ from `stats::t.test()`, see the "Details". Either named parameters can be given, or a `list` with the parameters. You must provide either `x` or `mean`, `sd` and `n`. If `x` is given then any values given for `mean`, `sd` and `n` will be overwritten. Also either `sd` or `sigma` or both must be given.

- `x` sample (default: `numeric(0)`)
- `mean` sample mean (default: `mean(x)`)
- `n` sample size (default: `length(x)`)

- `sd` sample standard deviation (default: `sd(x)`)
- `sigma` population standard deviation (default: `NA` = unknown)
- `mu0` true value of the mean (default: `0`)
- `alternative` a string specifying the alternative hypothesis (default: `"two.sided"`), otherwise `"greater"` or `"less"` can be used
- `alpha` significance level (default: `0.05`)
- `norm` is the population normal distributed? (default: `FALSE`)
- `n.clt` when the central limit theorem holds (default: `getOption("n.clt", 30)`)
- `t2norm` does the approximation $t_n \approx N(0; 1)$ hold? `(default:` NA= `uset2norm`‘ function)

Usage

```
ttest_num(..., arglist = NULL)
```

Arguments

`...`	named input parameters
`arglist`	list: named input parameters, if given `...` will be ignored

Details

The results of `ttest_num` may differ from `stats::t.test()`. `ttest_num` is designed to return results when you compute a t-test by hand. For example, for computing the test statistic the approximation $t_n \approx N(0; 1)$ is used if $n > n.tapprox$. The `p.value` is computed from the cumulative distribution function of the normal or the t distribution.

Value

A list with the input parameters and the following:

- `Xbar` distribution of the random sampling function $\bar{X}$, only available if `sigma` given
- `Statistic` distribution of the test statistics
- `statistic` test value
- `critical` critical value(s)
- `criticalx` critical value(s) in x range
- `acceptance0` acceptance interval for H0
- `acceptance0x` acceptance interval for H0 in x range
- `accept1` is H1 accepted?
- `p.value` p value for test

Examples

```
x <- runif(100)
ttest_num(x=x)
ttest_num(mean=mean(x), sd=sd(x), n=length(x))
ret <- ttest_num(x=x)
ret$alternative <- "less"
ttest_num(arglist=ret)
```

`unique_elem` *Unique Elements*

Description

Deletes all elements from a `hyperloop` object that are identical. Since the result in each run can be a list itself, only specific list elements can be used for comparison.

Usage

```
unique_elem(x, elem = NULL)
```

Arguments

`x`	a `hyperloop` object
`elem`	character: list elements which are used to check if `hyperloop` results are identical

Value

A reduced `hyperloop` object

Examples

```
x <- rnorm(100)
# 6 results: 3 different mu's, 2 var.equals
hl <- hyperloop(t.test, x=x, mu=list(-1, 0, 1), var.equal=list(TRUE, FALSE))
# reduction to 3 elements since var.equal does not play any role
length(unique_elem(hl))
# reduction to 1 element since the mean of x always the same
length(unique_elem(hl, "estimate"))
```

`unique_max` *Unique Maximum*

Description

Checks if `x` has a unique maximum. The largest and the second largest value must have at least a distance of `tol`.

Usage

```
unique_max(x, tol = 0.001)
```

Arguments

`x`	numeric: values to check
`tol`	numeric: minimum distance between the largest and the second largest value (default: `1e-3`)

Value

Logical

Examples

```
x <-runif(100)
unique_max(x)
unique_max(x, tol=0.1)
```

`vec2mat` *Vector to Matrix Conversion*

Description

Converts a vector to a horizontal or vertical matrix and sets `row-` or `colnames`. If `rownames` or `colnames` are given, then existing row names or column names are overwritten.

Usage

```
vec2mat(x, colnames = NULL, rownames = NULL, horizontal = TRUE)

to_mat(x, colnames = NULL, rownames = NULL, horizontal = TRUE)
```

Arguments

`x`	vector
`colnames`	character: vector of new column names (default: `NULL`)
`rownames`	character: vector of new row names (default: `NULL`)
`horizontal`	logical: horizontal or vertical matrix (default: `TRUE`)

Value

A matrix

Examples

```
x <- runif(5)
vec2mat(x)
vec2mat(x, horizontal=FALSE)
```

`zebra` *Apply Zebra Striping to an HTML Matrix*

Description

This function applies alternating background colors (zebra striping) to an `html_matrix` object, either by rows or by columns.

Usage

```
zebra(x, col = c("#FFFFFF", "#CCCCCC"), byrow = TRUE)
```

Arguments

`x`	An `html_matrix` object to which the zebra striping will be applied.
`col`	A character vector of colors to use for striping. Defaults to `c("#FFFFFF", "#CCCCCC")`.
`byrow`	Logical; if `TRUE`, colors are applied to rows, otherwise to columns. Default is `TRUE`.

Value

An `html_matrix` object with updated background colors.

Examples

```
library("magrittr")
m  <- matrix(13:24, ncol = 4)
hm <- html_matrix(m) %>% zebra()
html <- toHTML(hm)
if (interactive()) toHTML(hm, browser = TRUE) # opens in browser
```

Index