# Package ‘exams.forge’

July 10, 2024

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Support for Compiling Examination Tasks using the 'exams' Package</td>
</tr>
<tr>
<td>Version</td>
<td>1.0.10</td>
</tr>
<tr>
<td>Date</td>
<td>2024-07-08</td>
</tr>
</tbody>
</table>

**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, extraDistr, exams, knitr, MASS, methods, magrittr, psych, rjson, rstudioapi, spelling, stringdist, stringr, stranslate, tinytex, utils, xml2, xtable, yaml

**Encoding**

UTF-8

**VignetteBuilder**

knitr

**Suggests**

rmarkdown

**RoxygenNote**

7.3.2

**NeedsCompilation**

no

**Author**

Sigbert Klinke [aut, cre]

**Maintainer**

Sigbert Klinke <sigbert@hu-berlin.de>

**Repository**

CRAN

**Date/Publication**

2024-07-10 16:40:02 UTC
## Contents

- `exams_forge-package` ................................................. 4
- `add_data` .......................................................... 6
- `affix` ................................................................. 7
- `all_different` ......................................................... 8
- `assoc_data` .......................................................... 9
- `as_result` ........................................................... 10
- `as_string` ............................................................ 11
- `as_table` ............................................................. 12
- `as_ts` ................................................................. 14
- `binom_param` ......................................................... 15
- `breaks` ............................................................... 16
- `calledBy` ............................................................. 17
- `catif` ................................................................. 17
- `CImulen_data` ....................................................... 18
- `CImu_data` ........................................................... 19
- `CIpilen_data` ......................................................... 21
- `combinatorics` ....................................................... 23
- `cor_data` ............................................................. 24
- `data_n` ................................................................. 25
- `data_prob2` ........................................................... 26
- `ddiscrete` ............................................................. 27
- `ddiscrete2` ............................................................ 28
- `ddunif2` ............................................................... 30
- `distribution` .......................................................... 31
- `distributions` ........................................................ 33
- `divisor_25` ........................................................... 34
- `equal` ................................................................. 35
- `equations` ............................................................ 36
- `exams2call` ........................................................... 37
- `exercise` ............................................................... 37
- `extremes` ............................................................. 38
- `fcvt` ................................................................. 39
- `firstmatch` ............................................................ 40
- `fractions` ............................................................. 40
- `gapply` ............................................................... 41
- `grade` ................................................................. 42
- `grouped_data` ......................................................... 43
- `gsimplify` ............................................................ 43
- `histbreaks` ........................................................... 44
- `histdata` .............................................................. 45
- `histwidth` ............................................................ 46
- `histx` ................................................................. 47
- `hm_cell` ............................................................... 48
- `html_e2m` ............................................................. 50
- `html_matrix` ........................................................ 51
- `html_matrix_sk` ...................................................... 53
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>hyperloop ..................................................</td>
</tr>
<tr>
<td>hyper_param ..................................................</td>
</tr>
<tr>
<td>hypothesis_latex .............................................</td>
</tr>
<tr>
<td>incomplete_table ...........................................</td>
</tr>
<tr>
<td>inline .......................................................</td>
</tr>
<tr>
<td>is.prob ......................................................</td>
</tr>
<tr>
<td>knitif .......................................................</td>
</tr>
<tr>
<td>latexdef .....................................................</td>
</tr>
<tr>
<td>lcmval .......................................................</td>
</tr>
<tr>
<td>lm1_data .....................................................</td>
</tr>
<tr>
<td>lmr_data .....................................................</td>
</tr>
<tr>
<td>lsumprod .....................................................</td>
</tr>
<tr>
<td>makekey ......................................................</td>
</tr>
<tr>
<td>mcval .......................................................</td>
</tr>
<tr>
<td>meanint_data .................................................</td>
</tr>
<tr>
<td>means_choice ................................................</td>
</tr>
<tr>
<td>mime_image ..................................................</td>
</tr>
<tr>
<td>monomial .....................................................</td>
</tr>
<tr>
<td>moodle_m2s ..................................................</td>
</tr>
<tr>
<td>nearest_arg ..................................................</td>
</tr>
<tr>
<td>nom.cc .......................................................</td>
</tr>
<tr>
<td>nosanitize ...................................................</td>
</tr>
<tr>
<td>now ..........................................................</td>
</tr>
<tr>
<td>nsprintf .....................................................</td>
</tr>
<tr>
<td>num2str .....................................................</td>
</tr>
<tr>
<td>num_result ...................................................</td>
</tr>
<tr>
<td>num_solve ...................................................</td>
</tr>
<tr>
<td>pdensity .....................................................</td>
</tr>
<tr>
<td>pearson_data ................................................</td>
</tr>
<tr>
<td>pminimum ....................................................</td>
</tr>
<tr>
<td>pos ..........................................................</td>
</tr>
<tr>
<td>ppprobability ...............................................</td>
</tr>
<tr>
<td>print.equations .............................................</td>
</tr>
<tr>
<td>print.html_matrix ..........................................</td>
</tr>
<tr>
<td>prob_solve ..................................................</td>
</tr>
<tr>
<td>proptests ....................................................</td>
</tr>
<tr>
<td>proptest_data ...............................................</td>
</tr>
<tr>
<td>proptest_num ................................................</td>
</tr>
<tr>
<td>q2norm ......................................................</td>
</tr>
<tr>
<td>random .......................................................</td>
</tr>
<tr>
<td>refer .........................................................</td>
</tr>
<tr>
<td>replace_fmt ..................................................</td>
</tr>
<tr>
<td>rv ..........................................................</td>
</tr>
<tr>
<td>sample_size_freq ...........................................</td>
</tr>
<tr>
<td>scale_to .....................................................</td>
</tr>
<tr>
<td>skalenniveau .................................................</td>
</tr>
<tr>
<td>solution .....................................................</td>
</tr>
<tr>
<td>sos100 ......................................................</td>
</tr>
</tbody>
</table>
exams.forge-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.

```r
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
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.
add_data

Author(s)

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

add_data Vector Data Addition

Description

Adds data values to a given data vector x.

Usage

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

Arguments

x numeric: data vector
box character or numeric: a basic box is used
n numeric: number of data values on the left, the right, or both sides of x (default: c(0, 1))
range numeric: determines the range where the additional data values will be drawn from (default: c(0,1))

Details

Based on the data x, or the range(box), a box is computed. The length of the box gives the multiplier for the range. Then a left and right interval, from which the additional values are drawn uniformly, is computed: $\left[ leftboxvalue - range[2] \times boxlength; \left( leftboxvalue - range[1] \times boxlength \right) \right]$ (left interval) and $\left[ rightboxvalue + range[1] \times boxlength; \left( rightboxvalue + range[2] \times boxlength \right) \right]$ (right interval).

For box, "boxplot" can be also used and quantile(x, c(0.25, 0.75), na.rm=TRUE) can be used instead of range(x, na.rm=TRUE).

n can be a single number which will add n data values at the right side of x. If n is a vector of length two, then n[1] data values will be added at the left side of x and n[2] data values will be added at the right side of x.

Value

a data vector with new values
Examples

```r
x <- rnorm(8)
# add one value to the right
add_data(x, "box", range=1.5)
add_data(x, "range", range=0.1)
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 and Prefix and/or Suffix Manipulation*

**Description**

affix adds a prefix and/or a suffix to a (character) vector
math adds a `$` as pre- and suffix to a (character) vector
bracket adds a ( as prefix and ) as suffix to a (character) vector
unaffix deletes a pre- and/or suffix to a (character) vector
unquote deletes double quotes at the beginning and the ending of a (character) vector
uncdata deletes a `<![CDATA[ as prefix and ]]>` as suffix
cdata adds a `<![CDATA[ as prefix and ]]>` as suffix

**Usage**

```r
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)
```
all_different

Description
Tests if the differences between the entries in obj are larger than tol.

Usage
all_different(obj, tol)

Arguments
- obj: object: numeric R object that can be converted to a vector
- tol: numeric: minimum value

Value
logical
assoc_data

Examples

```r
x <- runif(10)
all_different(x, 0.0001)
all_different(x, 1)
```

---

**Description**

Given a frequency table, the function reorders the observations such that the given target association will be approximated and the marginal frequencies remain unchanged. Note that the target association may not be reached! `zero` allows for zero entries in the common distribution. If target is `NA` then the table is simply returned. `FUN` computes the association (or correlation) measure 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 of 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 check if you have chosen an appropriate target value (for a nominal measure in `0 <= target <= 1`, for ordinal measure in `-1 <= target <= +1`). `attr(joint, "target")` contains the achieved association.

**Usage**

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

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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
<td>table: table of absolute frequencies</td>
</tr>
<tr>
<td>zero</td>
<td>logical: zeros are allowed in the final probabilities (default: FALSE)</td>
</tr>
<tr>
<td>FUN</td>
<td>function: association or correlation function (default: nom.cc)</td>
</tr>
<tr>
<td>target</td>
<td>numeric: target association or correlation (default: NA)</td>
</tr>
<tr>
<td>tol</td>
<td>numeric: tolerance for target association or correlation (default: 0.001)</td>
</tr>
<tr>
<td>maxit</td>
<td>integer: maximal number of iterations (default: 100)</td>
</tr>
<tr>
<td>...</td>
<td>further parameters</td>
</tr>
</tbody>
</table>

Value

a modified frequency table

Examples

tab <- table_data(3, 2)
tab

```r
tab2 <- assoc_data(tab, target=0.5)
tab2
```

<table>
<thead>
<tr>
<th>as_result</th>
<th>Results with Rounding</th>
</tr>
</thead>
</table>

Description

Rounds x according to digits, FUN and sets a tolerance for the result. If the tolerance is not stated, consider it the maximum of $2 \times 10^{\text{-digits}}$.

Usage

```r
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: Digits that should be used for rounding or "integer" for digits=0, "%" for digits=2, or "probability" for digits=4. Abbreviations for the names can be used
tol numeric: tolerance for result
FUN function: rounding function (default: round2)

Value

A list with the original and a rounded value, digits used and tolerance.

Examples

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

as_string Vector to String Conversion

Description

Converts a character vector into a single string.

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
- **collapse**: character: glues text between elements (default: ","")
- **last**: character: glues text between the two last elements (default: ", and ")
- **name**: character: observation name (default: "x")
- **sorted**: logical: sorted or not sorted observations (default: FALSE)
- **val**: numeric: values to convert into fractions
- **latex**: logical: use of LaTeX \frac{.}{.} or not (default: FALSE)

Value

A string

Examples

```r
x <- runif(5)
y <- c(TRUE, FALSE, NA)
as_string(x)
as_string(y)
# toString
as_string(as.character(x))
as_string(as.character(y))
#
as_obs(x)
as_obs(sort(x), sorted=TRUE)
#
x <- round(runif(5), 2)
as_fraction(x)
as_fraction(x, TRUE)
#
y <- round(runif(5), 2)
as_sum(x)
```

---

**as_table**  
*Conversion to Table*

Description

Converts a vector into a horizontal table.
Usage

```r
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.

... further parameters for print.xtable

Value
A string.

Examples

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

---

as_ts Time Series

Description
Converts a ts_data object into a time series object (ts).

Usage

as_ts(ts)

Arguments

ts ts_data object

Value
A ts object.
Examples

# Time series from linear trend
ts <- ts_data(12, trend.coff= c(sample(0:10, 1), sample(1+(1:10)/20, 1)))
as_ts(ts)

---

### binom_param

#### Binomial Parameters

**Description**

Generates a data frame with potential values for size and prob, and is subjected to specific conditions:

- 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 size * prob must be one of the valid means.
- The same rules applies to sd.

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

- For NA, all combinations of (size, prob) are acceptable.
- If specified as a function, only those combinations for which the function returns TRUE are considered valid.
- If set to TRUE, combinations are accepted only if they satisfy either the condition size * prob * (1 - prob) > 9 (for norm, indicating a normal distribution approximation), or the conditions prob < 0.05 and n > 10 (for pois, implying a Poisson 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**

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

**Arguments**

- `n` integer: vector number of observations
- `p` numeric: vector of probabilities
- `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
- `tol` numeric: the tolerance for numerical comparison (default: ‘1e-6)
Value

a data frame with possible choices of n, p, mean and sd

Examples

binom_param(1000:50000, (5:25)/100, 0, 0)

breaks

Breaks

Description

Creates a number of equidistant or non-equidistant breaks for given data x. If width is not given then it will be set to diff(pretty(x))[1]. probs can either be a single integer, giving the number of quantiles, or a vector of probabilities with values in [0, 1]. Please note that if width is too large, then using probs may result in equidistant breaks too.

Usage

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

Arguments

x numeric: data
width numeric: class width (default: NULL)
probs numeric: number of non-equidistant classes (default: NULL)

Value

A numeric vector of breaks.

Examples

x <- rnorm(100, mean=1.8, sd=0.1)
bbreaks(x)
bbreaks(x, 0.1)
bbreaks(x, 0.1, probs=4)
calledBy

Function Calling

description
Checks if the result from base::sys.calls contains a call from fun.

Usage
calledBy(fun = "exams2pdf")
called_by(fun = "exams2pdf")

Arguments
fun character: name of the calling function (default: exams2pdf)

Value
logical

Examples
funb <- function() { calledBy('funa') }
funa <- function() { funb() }
funa()

catif

Condition cat

description
Calls cat if cond==TRUE.

Usage
catif(cond, ...)

Arguments
cond logical: condition, if true then cat is called, otherwise not
... further parameters
Value

Invisibly cond.

Examples

catif(TRUE, "PDF")    # Should appear
catif(FALSE, "Moodle") # Should not appear

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 ($l = 2 \times e$). 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
Confidence Intervals

Description

The CImu_data function is designed for the generation of confidence intervals pertaining to a population mean \( \mu \). The function accommodates scenarios in which a dataset \( x \) is either provided or generated through a random sampling process from a normal distribution, with user-specified parameters such as a mean \( \mu \) and a standard deviation \( \sigma \). Subsequently, the function computes essential statistical measures, including the sample mean \( \bar{x} \) and the standard deviation \( s \). Confidence intervals for the population mean are then calculated at user-defined confidence levels (conf.level). The output is a structured list containing pertinent statistics, encompassing the mean, sample standard deviation, confidence intervals, and other relevant details.
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 data values
n numeric: length of the vector x (if n<1 then n=5 is used)
xbar numeric: sample mean
sd numeric: sample standard deviation
conf.level numeric: vector of confidence levels of the interval (default: c(0.9, 0.95, 0.99))
mu numeric: true value of the mean
sigma numeric: vector of possible variance

Value

a list with
  • a with 1-(1-conf.level)/2
  • n number observations if given
  • xbar mean of observations if not given
  • mu theoretical mean if given
  • sd standard deviation of observations
  • sigma theoretical standard deviation if given
  • df degrees of freedom if a t distribution is used
  • q if sigma=NULL
  • ss either sd or sigma
CIpilen_data

- e margin of error (half of the length of the confidence interval(s))
- l length of the confidence interval(s)
- v endpoints of the confidence interval(s)

Examples

```r
# with data
x <- rnorm(100)
CImu_data(x, conf.level=0.95)
# simulate 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 $p_i$.

Usage

```r
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** $z_{1-\alpha/2}$
- **q^2** $z_{1-\alpha/2}^2$
- **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
p1l <- CIpilen_data((1:9/10), (1:9)/10, full=TRUE)
str(p1l)
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, "$n \text{ choose } k$"

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)
Parameters

- **n**: numeric: total number of elements
- **k**: numeric: number of elements to choose
- **repl**: logical: with repetition (default: FALSE)
- **...**: numeric: further arguments for \( \text{lfactquot} \)

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

**Correlation and Data Generation**

**Description**

Generates a data set based on \( x \) and \( y \) for a given target correlation \( r \) according to \texttt{stats::cor()}. The algorithm modifies the order of the \( y \)'s, therefore is guaranteed that the (marginal) distribution of \( x \) and \( y \) will not be modified. Please note that it is not guaranteed that the final correlation will be the desired correlation; the algorithm interactively modifies the order. If you are unsatisfied with the result, it might help to increase \( \text{maxit} \).

**Usage**

```r
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; given x values
- **y**: numeric; given y values
- **r**: numeric; desired correlation
- **method**: character; indicates which correlation coefficient is to be computed (default: "pearson")
- ... further parameters given to `stats::cor()`
- **maxit**: numeric; maximal number of iterations (default: 1000)

Value

A matrix with two columns and an attribute `interim` for intermediate values as matrix. The rows of the matrix contain:

- if `method=="pearson"`: $x_i, y_i, x_i - \bar{x}, y_i - \bar{y}, (x_i - \bar{x})^2, (y_i - \bar{y})^2$, and $(x_i - \bar{x})(y_i - \bar{y})$.
- if `method=="kendall"`:
  - $x_i$: The original x values.
  - $y_i$: The original y values.
  - $p_i$: The number of concordant pairs.
  - $q_i$: The number of discordant pairs.
- if `method=="spearman"`: $x_i, y_i, p_i$ (concordant pairs), and $q_i$ (disconcordant pairs). In a final step a vector with the row sums is appended as further column.

Examples

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

---

**data_n**

<table>
<thead>
<tr>
<th>Number of Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Description**

Generates a sequence of sample sizes in a range from min=5 to a max:

- whose root is an integer (`data_nsq`), and
- that are divisible only by 2 and 5 (`data_n25`)
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

A sequence of integers.

Examples

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

data_prob2 Probability/Frequency Matrix Generation

Description

Generates a nrow x 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)
ddiscrete

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

```r
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**

```r
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"))
**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)`
**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 distributions:

- `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 and 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].
distributions

- **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

```r
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)
```

---

### Description

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

### Usage

```r
data(distributions)
```

### Format

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

### Examples

```r
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 $2^55^v$
- **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 $\text{abs}(x-\text{round}(x,\text{digits}))<\text{tol}$
- **all_integer** checks whether all x's are integer, basically $\text{all}(\text{has_digits}(x,0))$

**Usage**

```r
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^{\text{-digits - 6}}
)

all_integer(x)

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


equal

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 \( \text{abs}(x - y) < \text{tol} \) when \( \text{outer} \) == FALSE, or if an a exists or a \( y[j] \) for each \( x[i] \) such that the condition \( \text{abs}(x[i] - y[j]) < \text{tol} \) 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

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

---

### 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

```r
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

```r
# 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 $\text{Traceback for exams2 Functions}$

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

A list with the function name and its valuated parameters.

**Examples**

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

---

exercise $\text{Data Exercise Structure}$

**Description**

Data structure for exercise data.

**Usage**

```r
exercise(exer, ...)
```

## Default S3 method: 
```r
exercise(exer = NULL, ...)
```
```r
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(p, type = c("all", "minimum", "maximum", "saddle"), tol = 1e-09)

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)
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

```r
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

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

```r
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**

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

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**

```r
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

```r
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

```r
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`)
grade

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))

<table>
<thead>
<tr>
<th>grade</th>
<th>Grades</th>
</tr>
</thead>
</table>

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

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

```r
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

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

```r
# 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 simplification 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

<table>
<thead>
<tr>
<th>histdata</th>
<th>Histogram Data</th>
</tr>
</thead>
</table>

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

```r
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

```r
#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

Creates a set of breaks and absolute frequencies in the range from 'from' to 'to'. The class widths are sampled from *widths*. The resulting numbers could be multiplied with an integer, if the sum(n) is too small. Additionally, it is checked whether the generated densities are terminating decimals.
histx

Usage

histwidth(from, to, widths, dmax = 100)

width_breaks(from, to, widths, dmax = 100)

dhistwidth(from, to, widths, dmax = 100)

Arguments

from: numeric: start value
to: numeric: end value
widths: numeric: a vector of width to sample from
dmax: numeric: max. denominator value

Value

A list with breaks, n’s for each class and decimal if all densities are terminating decimals.

Examples

l <- histwidth(1.6, 2.1, widths=c(0.05, 0.1, 0.15, 0.2))
l
x <- histx(l$breaks, l$n)
histdata(x, l$breaks)

Description

Midpoint-Based Data Creation for a Histogram

Given the breaks and the number of observations, a data set is generated with stats::runif(), using the class mids: \( x_i = \text{class mid}_j + \alpha * \text{class width}_j / 2 \). 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

```r
q <- sort(sample(seq(0.1, 0.9, by=0.1), 4))
qx <- pnorm(q)
histx(qx, diff(q))
```

---

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

```r
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

```r
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**

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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exam</td>
<td>list: returns a list from exams.forge</td>
</tr>
<tr>
<td>name</td>
<td>character: name of the HTML file (default: NULL)</td>
</tr>
<tr>
<td>pattern</td>
<td>character: string containing a regular expression to match the list elements (default: .)</td>
</tr>
<tr>
<td>mathjax</td>
<td>logical: should MathJax be loaded? (default: TRUE)</td>
</tr>
<tr>
<td>browseURL</td>
<td>logical: should the generated HTML be shown? (default: TRUE)</td>
</tr>
<tr>
<td>overwrite</td>
<td>logical: should the HTML file be overwritten (if it exists)? (default: FALSE)</td>
</tr>
<tr>
<td>header</td>
<td>integer: at which level of the list a (&lt;h2&gt;.../h2&gt;) element should be included? (default: 2)</td>
</tr>
<tr>
<td>merge</td>
<td>list: should elements with .XXXnn at the end be merged? (default: list('questionlist'='&lt;br&gt;'))</td>
</tr>
<tr>
<td>png</td>
<td>logical: if a entry ends with .png then the function will try to embed the PNG in the output</td>
</tr>
</tbody>
</table>

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

```r
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 \(\text{hm}_\text{cell}, \text{hm}_\text{title}, \text{hm}_\text{col}, \text{hm}_\text{row}\). For example: \(\text{hm}_\text{cell}(\text{hm},1,1,\text{text_align}="\text{right}\)") will lead to \(\langle td \text{style}="\text{text-align:right;}>\) 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
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.

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

Usage

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

```r
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 a 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)
**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 \(n+m, m, k\) is used and transformed to \(m, n, k\). 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 \(k \times m / (n + m)\) 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 \(k \times m / (m + n) \times (1 - m / (m + n)) \geq 9\) (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**

```r
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 \(0, 1, \ldots, m+n\)
- `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 \(n, p, \text{mean}\) and \(sd\).

Examples

```r
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:

- \(h_0.\text{left}\) left value of the null hypothesis, usually \(\mu\) or \(\pi\)
- \(h_0.\text{operator}\) operator of the null hypothesis, one of the following: eq, ne, lt, le, gt, or ge
- \(h_0.\text{right}\) right value of the null hypothesis, usually \(\mu_0, \pi_0\), or a hypothetical value
- \(h1.\text{left}\) left value of the alternative hypothesis, usually \(\mu\) or \(\pi\)
- \(h1.\text{operator}\) operator of the alternative hypothesis, one of the following: eq, ne, lt, le, gt, or ge
- \(h1.\text{right}\) right value of the alternative hypothesis, usually \(\mu_0, \pi_0\), or a hypothetical value
- \(H_0\) latex representation of the null hypothesis
- \(H1\) latex representation of the alternative hypothesis
- \(\text{match.left}\) do the left value in the null and the alternative hypothesis match?
- \(\text{match.right}\) do the right value in the null and the alternative hypothesis match?
- \(\text{match.operator}\) 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

\[
\text{incomplete_table}(\text{tab}, n, \text{maxit} = 1000) \\
\text{cont_table_fill}(\text{tab}, n, \text{maxit} = 1000)
\]

Arguments

- \text{tab}  \hspace{1cm} \text{table: a contingency table}
- \text{n}  \hspace{1cm} \text{integer: number of missing values}
- \text{maxit}  \hspace{1cm} \text{integer: number of maximal iterations (default: 1000)}

Value

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

Examples

\[
\text{tab} <- \text{rbind(c(0.02, 0.04, 0.34), c(0.02, 0.28, 0.3))} \\
\text{incomplete_table(tab, 7)}
\]

inline  Text Knitting

Description

Knits \text{txt} within an R code chunk.

Usage

\[
\text{inline(txt)} \\
\text{txt_knit(txt)}
\]
is.prob

Arguments

- **txt**: character

Value

- **Output.**

Examples

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

---

**is.prob**  
*Interval Checker*

**Description**

Checks if \( x \) is in an opened or closed interval between \( \text{min} \) and \( \text{max} \). 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**

```r
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**

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

**Knitting a Text Argument**

**Description**

Selects a text argument and returns the knitted result.

**Usage**

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

```r
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**

```r
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**

```r
latexdef(name, body)
```

```r
answercol(n)
```

```r
add_answercol_def(n)
```
Arguments

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

Value

Nothing

Examples

answercol(2)

lcmval

<table>
<thead>
<tr>
<th>Least Common Multiple</th>
</tr>
</thead>
</table>

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
Description

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

Usage

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

```r
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
lmr_data

scale numeric(2): standard deviation for x and y data
...

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(sos)
n <- sample(5:10, 1)
lm1 <- lmr_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 \texttt{lm} with the additional components:

- \(x\) the generated \(x\) values
- \(y\) the generated \(y\) values
- \(\sum_{i=1}^n x_i\)
- \(\sum_{i=1}^n y_i\)
- \(\sum_{i=1}^n x_i^2\)
- \(\sum_{i=1}^n y_i^2\)
- \(\sum_{i=1}^n x_i y_i\)
- mean\(x\) the mean of \(x\): \(\frac{1}{n} \sum_{i=1}^n x_i\)
- mean\(y\) the mean of \(y\): \(\frac{1}{n} \sum_{i=1}^n y_i\)
- var\(x\) the variation of \(x\): \(\sum_{i=1}^n (x_i - \bar{x})^2\)
- var\(y\) the variation of \(y\): \(\sum_{i=1}^n (y_i - \bar{y})^2\)
- var\(xy\) the common variation of \(x\) and \(y\): \(\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})\)
- \(s_{xy}\) the covariance of \(x\) and \(y\)
- \(r_{xy}\) the correlation of \(x\) and \(y\)
- \(b_0\) the intercept of the linear regression
- \(b_1\) the slope of the linear regression
- \(r^2\) the coefficient of determination of the linear regression

Examples

```r
# 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)
```

\texttt{lsumprod} \hspace{1cm} \textit{Supporting Functions for Math \LaTeX\ Output}

Description

\texttt{1sumprod} creates a latex printout of \(\sum_i x_i y_i\) with brackets if \(x_i\) or \(y_i\) starts with a `-`.
\texttt{lsum} creates a latex printout of \(x\) as sum.
\texttt{lprod} creates a latex printout of \(x\) as product.
\texttt{lvec} creates a latex printout of \(x\) as vector.
\texttt{lmean} creates a latex printout as \(\frac{x_1 + \ldots + x_n}{n}\).
\texttt{lvar} creates a latex printout as \(\frac{(x_1 - \overline{x})^2 + \ldots + (x_n - \overline{x})^2}{n}\).
\texttt{lbr} creates a latex printout of \(x\) with brackets if \(x\) starts with a `-`.
\texttt{lsgn} creates a latex printout of \(x\) with a plus or minus at the beginning.
Usage

\texttt{lsumprod}(\ldots, \texttt{br} = "\(\)"

\texttt{lsum}(x)

\texttt{lprod}(x)

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

\texttt{lvar}(x, \texttt{mu} = \texttt{NULL}, \texttt{br} = "\(\)"

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

\texttt{lsgn}(x)

\texttt{latex\_sumprod}(\ldots, \texttt{br} = "\(\)"

\texttt{latex\_sum}(x)

\texttt{latex\_product}(x)

\texttt{latex\_mean}(x)

\texttt{latex\_var}(x, \texttt{mu} = \texttt{NULL}, \texttt{br} = "\(\)"

\texttt{latex\_bracket(}
  x,
  \texttt{br} = \texttt{c}("\(\", \"[\", \"\{\", \"|\", \"\<\", \"a\", \"c\", \"f\"),
  \texttt{subset} = \texttt{NULL}
)
\texttt{latex\_pmsign}(x)

Arguments

\ldots\quad \text{further input values}

\texttt{br, left, right} \quad \text{character: which brackets to use. The possibilities are:}

\begin{itemize}
  \item (default) uses \texttt{\left(} and \texttt{\right)},
  \item \texttt{[ use \left[} and \texttt{\right]},
  \item \texttt{\{} use \texttt{\left\{} and \texttt{\right\}.}
\end{itemize}
• use \left| and \right|.,
• || uses \left\| and \right\|.,
• <, a use \left\langle and \right\rangle, and
• 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, '(, ']')
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 NAs 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

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**

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

---

<table>
<thead>
<tr>
<th>monomial</th>
<th>Monomial</th>
</tr>
</thead>
</table>

**Description**

Creates a polynomial of the form \( c \cdot x^d \).

**Usage**

```r
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**

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

```r
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

```r
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**

```r
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**

```r
# 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"))
```

---

**nom.cc  

Association and Correlation**

**Description**

Computation of the following association and correlation measures:

- nom.cc (corrected) contingency coefficient
- nom.cramer Cramer’s V or Phi
- ord.spearman Spearman’s rank correlation
- ord.kendall Kendall’s rank correlation
Usage

nom.cc(tab, correct = FALSE)
nom.cramer(tab, ...)
ord.spearman(tab, ...)
ord.kendall(tab, ...)
cccoef(tab, correct = FALSE)
cramer_vf(tab, ...)
cramer_coef(tab, ...)
kendall_corr(tab, ...)
spearman_corr(tab, ...)
rs_corr(tab, ...)

Arguments

  tab              table: contingency table with absolute frequencies
  correct          logical: if a correction should be applied (default: FALSE)
  ...              further parameters

Value

  numeric

Examples

  tab <- matrix(round(10*runif(15)), ncol=5)
  nom.cc(tab)
  nom.cc(tab, correct=TRUE)
  nom.cramer(tab)
  ord.spearman(tab)
  ord.kendall(tab)


cleansanitization

Description

cleansanitization makes no sanitization on the strings.
Usage

nosanitize(str)

Arguments

str character: vector to sanitize

Value

A sanitized character vector.

Examples

nosanitize("Test")

now\hline Current Time \\
\hline

Description

Returns a time stamp based on the current time. \texttt{now} basically calls \texttt{gsub('.\', '', sprintf('%.20f', as.numeric(Sys.time())), fixed=TRUE)}. To ensure that at each call a different time stamp is delivered \texttt{now} may call \texttt{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

- $\text{round\_de}$, it returns either Runden Sie Ihr Ergebnis auf eine ganze Zahl, Runden Sie Ihr Ergebnis auf eine Stelle nach dem Komma
- $\text{schoice\_de}$ returns Es kann eine oder mehrere Antworten richtig sein. Es ist ausreichend, eine richtige Antwort zu geben.

Usage

nsprintf($n$, $...$)

round_de($n$)

dchoice_de()

print_de($n$, $...$)

Arguments

$n$ integer: number(s) to be used

$...$ character: format strings to be used

Value

$sprintf$ed strings

Examples

\begin{verbatim}
nsprintf(0, '0' = "keine Netzunterbrechung", '1' = "eine Netzunterbrechung",
        "%i Netzunterbrechungen")
nsprintf(0:3, `0` = "keine Netzunterbrechung", `1` = "eine Netzunterbrechung",
        "%i Netzunterbrechungen")
\end{verbatim}

\begin{center}
\textbf{num2str} \quad \textit{Number to String Conversion}
\end{center}

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

Description

num_result creates a list with the following elements:

• x the original values
• fx the rounded values with exams::fmt() as a character
• tolerance the tolerance
• digits the digits used for rounding

Note that x may contain more than one numeric value to determine the rounding and tolerance. Make sure that you use for numeric exercises ...$x[1].

If digits are not given and length(x)>1 then ceiling(-log10(min(diff(sort(x)), na.rm=TRUE))) is used. If digits are not given and length(x)==1 then 3+ceiling(-log10(abs(x))) is used. If no tolerance is given then tolmult*10^(1-digits) is used.

int_result can be used if the result is an integer number and calls num_result(x, 0, 0.1, 1, ...) with a tolerance of 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: rounded data
- **digits**: numeric: number of digits of rounding (default: NULL)
- **tolerance**: numeric: tolerance for rounded data (default: NULL)
- **tolmult**: numeric: multiplier for tolerance
- **...**: further parameters from exams::fmt()

Value

A list.

Examples

```r
# height for german man (in meter)
x <- rnorm(10, mean=1.8, sd =0.25)
um_result(c(mean(x), x), digits=2)
int_result(mean(x))
#
str(num_result(pi, 3))
str(num_result(pi, 6))
str(num_result(pi, 6, tolmult=5))
str(num_result(pi, 6, tolmult=5, tolerance=1e-6))
```

---

<table>
<thead>
<tr>
<th>num_solve</th>
<th>Target Variable Value</th>
</tr>
</thead>
</table>

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

```r
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**

```r
# 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),
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)
```

---

**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

\[
\text{pdensity}(x, \text{size} = 3, \text{power} = 1, \text{tol} = 1e-06)
\]

\[
\text{sample_density}(x, \text{size} = 3, \text{power} = 1, \text{tol} = 1e-06)
\]

Arguments

\(x\)
numeric: range of density with \(a = \text{min}(x, \text{na.rm} = \text{TRUE})\) and \(b = \text{max}(x, \text{na.rm} = \text{TRUE})\)

\(\text{size}\)
numeric: number of elements to be sampled (without replacement) from \(x\)

\(\text{power}\)
numeric: constant or linear density function

\(\text{tol}\)
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
- \(f_x\) the distribution function at \(x\)
- \(p_{\text{coeff}}\) a polynomial (intercept = first value)
- \(q_{\text{coeff}}\) indefinite integral of the polynomial (intercept = first value)
- \(p_{\text{int}}\) result of the integral\((p_{\text{coeff}}, c(a,b), 0:2)\)

Examples

\[
\text{pdensity}(-5:5)
\]

\[
\text{pdensity}(-5:5, \text{power}=1)
\]

---

**pearson_data**  
*Pearson Data*

Description

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

Usage

\[
\text{pearson_data}(r, n = 100, n_{\text{max}} = 6, \text{maxt} = 30, \text{xsos} = \text{NULL}, \text{ysos} = \text{NULL})
\]

\[
\text{dpearson}(r, n = 100, n_{\text{max}} = 6, \text{maxt} = 30, \text{xsos} = \text{NULL}, \text{ysos} = \text{NULL})
\]
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: $x_i, y_i, x_i - \bar{x}, y_i - \bar{y}, (x_i - \bar{x})^2, (y_i - \bar{y})^2$, and $(x_i - \bar{x})(y_i - \bar{y})$. In a final step, a vector with the row of sums is appended as a further column.

Examples

```r
data(sos)
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 $[\text{lower}, \text{upper}]$. The values and the interval borders of the polynomial $p$ are evaluated and the minimum value is returned.

Usage

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

```r
polynomial_minimum(  
p,  
```
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

\[
\begin{aligned}
p & \leftarrow \text{polynomial(c(-5, 3, -3, 1))} \\
pmin & \text{imum}(p, -3, 3)
\end{aligned}
\]

Description

Generates intervals based on powers of ten.

Usage

pos(pow)
neg(pow)
dbl(pow)
ibl(pow)
ipo(pos(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,
```r
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)))

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*x/sqrt(n), "v_o=\bar{x}+c\cdot\frac{s^2}{n},
            u~x-c*x/sqrt(n), "v_u=\bar{x}-c\cdot\frac{s^2}{n},
            e~c*x/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

```r
## 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

```r
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:

- \( P(A) = 1 - P(\bar{A}) \),
- \( P(A|B) = 1 - P(\bar{A}|B) \),
- \( P(A^B) = P(B^A) \),
- \( P(B) = P(A^B) + P(\bar{A}B) \),
- \( P(A|B) = P(A^B)/P(B) \), 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

```r
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 result 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

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

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 \( \pi_0 \) (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:

- \( \pi_0 \) 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

prop_test_data()
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: \( \text{sd}(x) \))
- \( \pi_0 \) true value of the proportion (default: 0.5)
- \( \text{alternative} \) a string specifying the alternative hypothesis (default: "two.sided"), otherwise "greater" or "less" can be used
- \( \alpha \) significance level (default: 0.05)
- \( \text{binom2norm} \) 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 \( p.\text{value} \) 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
q2norm

- 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-distribution.

Examples

```r
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}"
)
```

```r
to = deparse(substitute(x)),
    index = 1:length(x)
)}
```
replace_fmt

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

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

```r
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

```r
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

```r
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)
```
### skalenniveau

**Skalenniveau**

#### Description

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

#### Usage

```r
data(skalenniveau)
```

#### Format

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

#### Examples

```r
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

```r
solution(x, ...)
```

## Default S3 method:

```r
solution(x, ...)
```

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

```r
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)). 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, n=800, and n=1000.

**Usage**

data(sos)

sos200

sos400

sos800

sos1000

**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.

**Examples**

data(sos)

head(sos100)

rowSums(sos100^2, na.rm=TRUE)

rowSums(sos100, na.rm=TRUE)
spell

RMarkdown Spell Check

Description

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

Usage

```r
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")
)
```

```r
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

```r
# none
```
sqrtnp  Calculating Square Roots of np(1-p) Combinations

Description

Computes $\sqrt{np(1-p)}$ for all combinations of $n$ and $p$. If the result has only digits after the decimal point, then $n$, $p$, and $\sqrt{np(1-p)}$ 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 $|v - \text{round}(v, \text{digits})| < \text{tol}$ 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 $\text{snp}$ ($= \sqrt{np(1-p)}$).

Examples

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

sumofsquares  Sum of Squared Integers
sumofsquares

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 \( k \leq n_{\text{max}} \). 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 \( \text{maxt} \) seconds, which may result that not all possible solutions are found. To reduce computing time, \( \text{rbind}'s \) in the function are replaced by allocating matrices with size rows to fill in the results. Note that the following data sets are available:

- \( \text{sos100}=\text{sumofsquares}(100, 10, \text{zerosum=TRUE, maxt=Inf}) \),
- \( \text{sos200}=\text{sumofsquares}(200, 10, \text{zerosum=TRUE, maxt=Inf}) \),
- \( \text{sos400}=\text{sumofsquares}(400, 10, \text{zerosum=TRUE, maxt=Inf}) \),
- \( \text{sos800}=\text{sumofsquares}(800, 10, \text{zerosum=TRUE, maxt=Inf}) \), and
- \( \text{sos1000}=\text{sumofsquares}(1000, 10, \text{zerosum=TRUE, maxt=Inf}) \)

Usage

\[
\text{sumofsquares}(n, n_{\text{max}} = 10, \text{zerosum = FALSE, maxt = 30, size = 100000L})
\]

\[
\text{sum_sq}(n, n_{\text{max}} = 10, \text{zerosum = FALSE, maxt = 30, size = 100000L})
\]

Arguments

- \( n \): integer: number to decompose as sum of squares
- \( n_{\text{max}} \): integer: maximum number of squares in the sum
- \( \text{zerosum} \): logical: should the solution sum up to one (default: FALSE)
- \( \text{maxt} \): numeric: maximal number of seconds the routine should run
- \( \text{size} \): numeric: length of additional matrix size (default: 100000L)

Value

A matrix with \( n_{\text{max}} \) column with \( x_i \)'s. NA means number has not been used.

Examples

\[
\text{sos} \leftarrow \text{sumofsquares}(100, 6) \ # 23 \text{ solutions}
\]

\[
\text{head(sos)}
\]

\[
\text{table(rowSums(!is.na(sos))))}
\]

# one solution with one or two \( x_i \)
# five solutions with four \( x_i \)
# six solutions with five \( x_i \)
# ten solutions with six \( x_i \)

\[
\text{rowSums(sos^2, na.rm=TRUE)} \ # \text{all 100}
\]

\[
\text{sos} \leftarrow \text{sumofsquares}(100, 6, \text{zerosum=TRUE)}
\]

\[
\text{head(sos)}
\]

\[
\text{rowSums(sos^2, na.rm=TRUE)} \ # \text{all 100}
\]

\[
\text{rowSums(sos, na.rm=TRUE)} \ # \text{all 0}
\]
Description

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

- \( \text{sos100}=\text{sumofsquares}(100, 10, \text{zerosum}=\text{TRUE}, \text{maxt}=\text{Inf}) \).
- \( \text{sos200}=\text{sumofsquares}(200, 10, \text{zerosum}=\text{TRUE}, \text{maxt}=\text{Inf}) \).
- \( \text{sos400}=\text{sumofsquares}(400, 10, \text{zerosum}=\text{TRUE}, \text{maxt}=\text{Inf}) \).
- \( \text{sos800}=\text{sumofsquares}(800, 10, \text{zerosum}=\text{TRUE}, \text{maxt}=\text{Inf}) \). and
- \( \text{sos1000}=\text{sumofsquares}(100, 10, \text{zerosum}=\text{TRUE}, \text{maxt}=\text{Inf}) \)

Usage

\[
\text{sumofsquares1}(n^2, \text{nobs} = 10, n = 0, x = \text{runif(nobs)}, \text{maxit} = 1000)
\]

Arguments

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

Value

A integer vector of length \text{nobs}.

Examples

\[
\text{sumofsquares1}(100, 20)
\]
\[
\text{sumofsquares1}(100, 20)
\]
Description

Functions which deliver TRUE or FALSE if any approximation if possible. The approximation parameter c can be set directly, or it can be given via getOption. The approximation functions deliver TRUE in the following scenarios:

- t2norm: n>c with c=30
- binom2norm: if the type is "single" (default) then it checks size × prob × (1-prob)>c, or else it checks size × prob>c and size × (1-prob)>c with c=9
- clt2norm: n>c with c=30. Note that the existence of the expectation and variance, which are required by the Central Limit Theorem, cannot be checked.

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
- c numeric: approximation parameter (default: getOption("distribution.approxfun") or a default value)
- size integer: number of observations
- prob numeric: probability of success on each trial
- type character: approximation condition used
**Value**

logical if the approximation would be possible

**Examples**

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

---

**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 $2^{p_{ij}}5^{q_{ij}}$ is randomly chosen, ensuring that the sum of the entries is less than, or equal to $n$.

**Usage**

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

```r
dtable(
  nrow,
  ncol,
  unit = 10,
)
```

```r
dtable(
  nrow,
  ncol,
  unit = 10,
)
```

```r
dtable(
  nrow,
  ncol,
  unit = 10,
)
```
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

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)

Description

• toHTML returns an HTML representation of a matrix and, optionally, shows the result in the browser. If you decide to view the result in a browser then the HTML will be written to a temporary file and utils::browseURL() will be called
• toLatex returns a LaTeX representation of a matrix, but supports just a small subset of style options
• toHTMLorLatex returns an HTML or LaTeX representation of a matrix, depending if exams2pdf is in the call list or not

Usage

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

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

toHTMLorLatex(x, ...)
```

Arguments

- `x, object` html_matrix object
- `browser` logical: show the HTML in a browser (default: FALSE)
- `...` further parameters to utils::browseURL()

Value

character
Examples

library("tools")
m <- matrix(1:12, ncol=4)
hm <- html_matrix(m)
if (interactive()) html <- toHTML(hm, browser=TRUE)
toHTML(hm)
toLatex(hm)

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

```r
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

<table>
<thead>
<tr>
<th>Tooltip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Description

Adds a text tooltip to the HTML matrix.

Usage

```r
tooltip(x, tooltip = NULL)
```

```r
add_tooltip(x, tooltip = NULL)
```

Arguments

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

Value

An html_matrix object

Examples

```r
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**

**Description**

Conversion to R Markdown.

**Usage**

`toRMarkdown(txt)`

**Arguments**

- `txt`: character: vector with lines of Moodle Markdown

**Value**

Lines with RMarkdown

**Examples**

```r
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

```r
## 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 an absolute value of coefficient minus one is smaller than tol then coefficient is not kept
- **...**: unused parameters

Value

A character

Examples

```r
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)
```
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
transformif

Value

list with questions and solutions

Examples

```r
generate questions and solutions for the Transformatif.

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 \( \log(a + b \times x) \) if \( p=0 \) and \( (a + b \times x)^p \). Otherwise the transformation can be either applied to each element of `x`, or to all elements of `x`.

**Usage**

```r
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: \(-\text{abs}(\text{min}(x))\))
- `b`: numeric: scale (default: 1)
- `p`: numeric: power (default: 1)

**Value**

A transformed vector

**Examples**

```r
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**

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

```r
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

```r
# 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**

```r
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 \texttt{ts\_data} object with list elements:

- filter the filter used
- moving.\_average the computed moving average

Examples

# trend from a quadratic model
ts <- \texttt{ts\_data(12, trend.coeff=\texttt{\{sample(0:10, 1), sample(1+(1:10)/20, 1), 0.5\}})}
ts\_moving\_average(ts, 3)

\begin{flushleft}
\textbf{ts\_trend\_season} \hspace{1cm} \textit{Trend and Season Model}
\end{flushleft}

Description

Estimate a trend and season model from a \texttt{ts\_data} object.

Usage

\begin{verbatim}
\texttt{ts\_trend\_season(ts, trend = NULL, season = NULL)}
\texttt{ts\_ts(ts, trend = NULL, season = NULL)}
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{ts} \hspace{1cm} \texttt{ts\_data} object
\item \texttt{trend} \hspace{1cm} numeric or logical: if \texttt{trend} is \texttt{TRUE} then a linear trend will be estimated, otherwise an exponential trend. If \texttt{trend} is numeric this is considered as trend value
\item \texttt{season} \hspace{1cm} numeric or logical
\end{itemize}

Value

Returns an extended \texttt{ts\_data} object with the following list of elements:

- \texttt{t} the time points
- \texttt{s} the season for the time points
- \texttt{xt} the time series values
- \texttt{trend} the fitted trend values
- \texttt{trend.coeff} the trend coefficients
- \texttt{trend.linear} the trend type, if \texttt{NA} then it is unknown
- \texttt{season} the fitted season values
- \texttt{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.squared** the $R^2$ of the final model

Examples

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

### 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 result 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

```r
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:

```r
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")
)
```
**ttest_data**

**Value**

A list of ttest objects is returned

**Examples**

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

# 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**

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

# 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)
```
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 \(\text{two.sided}\) (default), \(\text{greater}\) or \(\text{less}\)
- **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:

- **\(\mu_0\)** 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 \(\text{two.sided}\), \(\text{greater}\) or \(\text{less}\))
- **altsd** alternative values usable for \(sd\) (if \(\text{use.sigma}=\text{TRUE}\)) or \(\sigma\) (if \(\text{use.sigma}=\text{FALSE}\))

Examples

```r
  ttest_data()
```

\[
\text{ttest\_num} \quad T\text{-tests}
\]

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: \(\text{numeric}(0)\))
- mean sample mean (default: \(\text{mean}(x)\))
- \(n\) sample size (default: \(\text{length}(x)\))
• 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= use t2norm' 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)
```
vec2mat

**Arguments**

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

**Value**

Logical

**Examples**

```r
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**

```r
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**

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

* datasets
  distributions, 33
  skalenniveau, 96
  sos100, 98

add_affix (affix), 7
add_anwercol_def (latexdef), 60
add_bracket (affix), 7
add_breaks (breaks), 16
add_cdata (affix), 7
add_data, 6
add_math (affix), 7
add_tooltip (tooltip), 108
affix, 7
all_different, 8
all_integer (divisor_25), 34
answercol (latexdef), 60
apply_grid (gapply), 41
approx_binom2norm (t2norm), 103
approx_clt2norm (t2norm), 103
approx_equal (equal), 35
approx_rational (fractions), 40
approx_t2norm (t2norm), 103
as_fraction (as_string), 11
as_obs, 5
as_obs (as_string), 11
as_res (as_result), 10
as_result, 10
as_string, 11
as_sum (as_string), 11
as_table, 12
as_ts, 14
assoc_data, 9
assoc_data(), 28
base::charmatch(), 40
base::sample, 44
base::sys.calls, 17
binom (combinatorics), 23
binom2norm (t2norm), 103
binom_param, 15
biv_discrete_prob (ddiscrete2), 28
bracket (affix), 7
breaks, 16
brkt (affix), 7
called_by (calledBy), 17
calledBy, 17
catif, 17
c_coef (nom.cc), 72
cdata (affix), 7
cdf (distribution), 31
choice_list (to_choice), 111
CImu_data, 19
CImulen_data, 18
CIpilen_data, 21
clt2norm (t2norm), 103
combination (combinatorics), 23
combinatorics, 23
combo (combinatorics), 23
combs (combinatorics), 23
compute_cdf (distribution), 31
compute_modes (mcval), 67
compute_pdf (distribution), 31
compute_probability (distribution), 31
condition_cat (catif), 17
cont_table_fill (incomplete_table), 58
cor_data, 24
cramer_coef (nom.cc), 72
cramer_vf (nom.cc), 72
dassoc (assoc_data), 9
data_n, 25
data_n25 (data_n), 25
data_nsq (data_n), 25
data_prob2, 26
dbinomtest (proptest_data), 88
dbl (pos), 81
dbreaks (breaks), 16
dcimu (CImu_data), 19
dcimulen (CImulen_data), 18
dcipilen (CIpilen_data), 21
dcorr (cor_data), 24
ddiscrete, 27
ddiscrete(), 27
ddiscrete2, 28
d3nif2, 30
denom_25 (divisor_25), 34
denominator_25 (divisor_25), 34
denom_25 (divisor_25), 34
dgrouped (grouped_data), 43
dhist (histdata), 45
dhistbreaks (histbreaks), 44
dhistwidth (histwidth), 46
dhistx (histx), 47
digits (as_result), 10
distribution, 31
distributions, 33
divisor_25, 34
dn (data_n), 25
dn25 (data_n), 25
dnsizelr (sample_size_freq), 94
dnsq (data_n), 25
dpearson (pearson_data), 79
dprob2 (data_prob2), 26
dt (ttest_data), 117
dtable (table_data), 104
dts (ts_data), 113
equal, 35
equations, 36
exams::forge (exams::forge-package), 4
exams::forge-package, 4
exams::2call, 37
exams::fmt(), 76, 77, 93
exercise, 37
exercise::data(exercise), 37
extremes, 38
fact (combinatorics), 23
factquot (combinatorics), 23
fcvt, 39
firstmatch, 40
fractions, 40
freq_table (table_data), 104
gapply, 41
gen mid (histx), 47
grade, 42
ggraphics::hist, 46
grouped_data, 43
grouped_stats (grouped_data), 43
gsimplify, 43
has digits (divisor_25), 34
histbreaks, 44
histdata, 45
histwidth, 46
histx, 47
hloop (hyperloop), 54
hm cell, 48
hm col (hm cell), 48
hm index (hm cell), 48
hm row (hm cell), 48
hm table (hm cell), 48
hm title (hm cell), 48
hm tr (hm cell), 48
html e2m, 50
html matrix, 51
html_matrix, 53
html mx (html matrix), 51
hunspell::dictionary, 99
hyper_param, 55
hyperloop, 54
hypothesis latex, 56
idbl (pos), 81
in range (is.prob), 59
incomplete_table, 58
ineg (pos), 81
inline, 58
int_res (num_result), 76
int_result (num_result), 76
ipos (pos), 81
is distribution (distribution), 31
is prob, 59
is distribution (distribution), 31
is prob (is.prob), 59
is prob interval (is.prob), 59
is term (divisor_25), 34
is terminal (divisor_25), 34
kendall corr (nom.cc), 72
knit select (knitif), 60
knitif, 60
latex_bracket (lsumprod), 64

latex_mean (lsumprod), 64
latex_pmsign (lsumprod), 64
latex_prob (prob_solve), 85
latex_product (lsumprod), 64
latex_sum (lsumprod), 64
latex_sumprod (lsumprod), 64
latex_var (lsumprod), 64
latexdef, 60
lbinom (combinatorics), 23
lbr (lsumprod), 64
lcm_vector (lcmval), 61
lfact (combinatorics), 23
lfactquot (combinatorics), 23
lfrac (as_string), 11
lhypo (hypothesis_latex), 56
lm1_data, 62
lm_regression_data (lmr_data), 63
lmath (affix), 7
lmatrix, 5
lmatrix (html_matrix_sk), 53
lmean (lsumprod), 64
lmr_data, 63
lobs (as_string), 11
lprob (prob_solve), 85
lprod (lsumprod), 64
lrv (rv), 94
lsgn (lsumprod), 64
lstring (as_string), 11
lsum (lsumprod), 64
lsumprod, 64
lvar (lsumprod), 64
lvec (lsumprod), 64
make_key (makekey), 66
makekey, 66
MASS::fractions (), 40
math (affix), 7
mchoice_moodle (moodle_m2s), 71
mcv (mcval), 67
mcval, 67
mean.histogram (histdata), 45
meanint_data, 68
means (means_choice), 68
means_choice, 68
median.histogram (histdata), 45
mime_image, 69
mime_img (mime_image), 69
mod_cell (hm_cell), 48
mod_col (hm_cell), 48
mod_ind (hm_cell), 48
mod_row (hm_cell), 48
mod_t (hm_cell), 48
mod_title (hm_cell), 48
mod_tr (hm_cell), 48
modify_cell (hm_cell), 48
modify_col (hm_cell), 48
modify_index (hm_cell), 48
modify_row (hm_cell), 48
modify_table (hm_cell), 48
modify_title (hm_cell), 48
modify_tr (hm_cell), 48
monom (monomial), 70
nom.cc, 72
nom.cramer (nom.cc), 72
nomanitize, 73
now, 74
nsprintf, 75
num2str, 75
num_res (num_result), 76
num_result, 76
num_solve, 77
only_digits (divisor_25), 34
ord.kendall (nom.cc), 72
ord.spearman (nom.cc), 72
pdensity, 78
pdunif2 (ddunif2), 30
pearson_data, 79
pearson_data (), 62
permutation (combinatorics), 23
pmdef (distribution), 31
pminimum, 80
point_probability (distribution), 31
polynomial_minimum (pminimum), 80
polynomial_probability (pprobability), 82
pos, 81
pprob (distribution), 31
pprobability, 82
pretty, 46
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>prime_numbers (divisor_25), 34</td>
<td></td>
</tr>
<tr>
<td>primes (divisor_25), 34</td>
<td></td>
</tr>
<tr>
<td>print.distribution (distribution), 31</td>
<td></td>
</tr>
<tr>
<td>print.equations, 84</td>
<td></td>
</tr>
<tr>
<td>print.html_matrix, 85</td>
<td></td>
</tr>
<tr>
<td>print.prob_solve (prob_solve), 85</td>
<td></td>
</tr>
<tr>
<td>print_de (nsprintf), 75</td>
<td></td>
</tr>
<tr>
<td>prob (distribution), 31</td>
<td></td>
</tr>
<tr>
<td>prob1 (distribution), 31</td>
<td></td>
</tr>
<tr>
<td>prob_mx (data_prob2), 26</td>
<td></td>
</tr>
<tr>
<td>prob_solve, 85</td>
<td></td>
</tr>
<tr>
<td>probability_solution (prob_solve), 85</td>
<td></td>
</tr>
<tr>
<td>prop_binomtest_data (propetest_data), 88</td>
<td></td>
</tr>
<tr>
<td>prop_binomtest_num (propetest_num), 90</td>
<td></td>
</tr>
<tr>
<td>propertest_data, 88</td>
<td></td>
</tr>
<tr>
<td>propertest_num, 90</td>
<td></td>
</tr>
<tr>
<td>propertest_num(), 88</td>
<td></td>
</tr>
<tr>
<td>proptests, 87</td>
<td></td>
</tr>
<tr>
<td>q2norm, 91</td>
<td></td>
</tr>
<tr>
<td>qdunif2 (ddunif2), 30</td>
<td></td>
</tr>
<tr>
<td>quantile.distribution (distribution), 31</td>
<td></td>
</tr>
<tr>
<td>quantile.histogram (histdata), 45</td>
<td></td>
</tr>
<tr>
<td>rand (random), 92</td>
<td></td>
</tr>
<tr>
<td>rand_breaks (histbreaks), 44</td>
<td></td>
</tr>
<tr>
<td>random, 92</td>
<td></td>
</tr>
<tr>
<td>rdunif2 (ddunif2), 30</td>
<td></td>
</tr>
<tr>
<td>refer, 92</td>
<td></td>
</tr>
<tr>
<td>refer2vector (refer), 92</td>
<td></td>
</tr>
<tr>
<td>remove_affix (affix), 7</td>
<td></td>
</tr>
<tr>
<td>remove_cdata (affix), 7</td>
<td></td>
</tr>
<tr>
<td>remove_quotes (affix), 7</td>
<td></td>
</tr>
<tr>
<td>reorder_association_data (assoc_data), 9</td>
<td></td>
</tr>
<tr>
<td>replace_fmt, 93</td>
<td></td>
</tr>
<tr>
<td>rm_spell_check (spell), 99</td>
<td></td>
</tr>
<tr>
<td>rmdFormatRV (rv), 94</td>
<td></td>
</tr>
<tr>
<td>rmdFormatRV, (rv), 94</td>
<td></td>
</tr>
<tr>
<td>round_25 (divisor_25), 34</td>
<td></td>
</tr>
<tr>
<td>round_de (nsprintf), 75</td>
<td></td>
</tr>
<tr>
<td>rounded (as_result), 10</td>
<td></td>
</tr>
<tr>
<td>rs_corr (nom.cc), 72</td>
<td></td>
</tr>
<tr>
<td>rv, 94</td>
<td></td>
</tr>
<tr>
<td>sample_density (pdensity), 78</td>
<td></td>
</tr>
<tr>
<td>sample_size_freq, 94</td>
<td></td>
</tr>
<tr>
<td>scale_to, 95</td>
<td></td>
</tr>
<tr>
<td>schoice_de (nsprintf), 75</td>
<td></td>
</tr>
<tr>
<td>sequation (num_solve), 77</td>
<td></td>
</tr>
<tr>
<td>simple_hloop (gsimplify), 43</td>
<td></td>
</tr>
<tr>
<td>simplify_hyperloop (gsimplify), 43</td>
<td></td>
</tr>
<tr>
<td>skalenniveau, 96</td>
<td></td>
</tr>
<tr>
<td>slr_data (lm1_data), 62</td>
<td></td>
</tr>
<tr>
<td>sol_ans (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_info (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_int (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_mc (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_mc_ans (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_mc_tf (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_meta (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_num (solution), 96</td>
<td></td>
</tr>
<tr>
<td>sol_tfl (solution), 96</td>
<td></td>
</tr>
<tr>
<td>solution, 96</td>
<td></td>
</tr>
<tr>
<td>sos100, 98</td>
<td></td>
</tr>
<tr>
<td>sos1000 (sos100), 98</td>
<td></td>
</tr>
<tr>
<td>sos200 (sos100), 98</td>
<td></td>
</tr>
<tr>
<td>sos400 (sos100), 98</td>
<td></td>
</tr>
<tr>
<td>sos800 (sos100), 98</td>
<td></td>
</tr>
<tr>
<td>spearman_corr (nom.cc), 72</td>
<td></td>
</tr>
<tr>
<td>spell, 99</td>
<td></td>
</tr>
<tr>
<td>spelling::spell_check_files(), 99</td>
<td></td>
</tr>
<tr>
<td>sprob (prob_solve), 85</td>
<td></td>
</tr>
<tr>
<td>sqrtntp, 100</td>
<td></td>
</tr>
<tr>
<td>stats::binom.test, 90</td>
<td></td>
</tr>
<tr>
<td>stats::binom.test(), 90</td>
<td></td>
</tr>
<tr>
<td>stats::cor(), 24, 25</td>
<td></td>
</tr>
<tr>
<td>stats::dhyper(), 55</td>
<td></td>
</tr>
<tr>
<td>stats::lm(), 63</td>
<td></td>
</tr>
<tr>
<td>stats::runif(), 47</td>
<td></td>
</tr>
<tr>
<td>stats::t.test(), 118, 119</td>
<td></td>
</tr>
<tr>
<td>stats::uniroot(), 77, 78</td>
<td></td>
</tr>
<tr>
<td>stringdist::stringdistmatrix, 72</td>
<td></td>
</tr>
<tr>
<td>sum_discrete_unif_cdf (ddunif2), 30</td>
<td></td>
</tr>
<tr>
<td>sum_discrete_unif_pmf (ddunif2), 30</td>
<td></td>
</tr>
<tr>
<td>sum_discrete_unif_quantile (ddunif2), 30</td>
<td></td>
</tr>
<tr>
<td>sum_discrete_unif_rand (ddunif2), 30</td>
<td></td>
</tr>
<tr>
<td>sum_sq (sumofsquares), 100</td>
<td></td>
</tr>
<tr>
<td>summary.distribution (distribution), 31</td>
<td></td>
</tr>
<tr>
<td>sumofsquares, 100</td>
<td></td>
</tr>
<tr>
<td>sumofsquares(), 79, 80</td>
<td></td>
</tr>
<tr>
<td>sumofsquares1, 102</td>
<td></td>
</tr>
<tr>
<td>t2norm, 103</td>
<td></td>
</tr>
<tr>
<td>table_data, 104</td>
<td></td>
</tr>
<tr>
<td>template, 105</td>
<td></td>
</tr>
<tr>
<td>to_choice, 111</td>
<td></td>
</tr>
<tr>
<td>to_mat (vec2mat), 121</td>
<td></td>
</tr>
<tr>
<td>toHTML.html_matrix, 106</td>
<td></td>
</tr>
</tbody>
</table>
toHTML/XML (html_e2m), 50

toHTMLorLatex (toHTML.html_matrix), 106
tol (as_result), 10

toLatex (toHTML.html_matrix), 106
toLatex.distribution (distribution), 31
toLatex.equation_solve (num_solve), 77
toLatex.polynomial, 107
toLatex.prob_solve (prob_solve), 85
tolerance (as_result), 10

tooltip, 108
toRMarkdown, 109
toString.polynomial, 109
toTable (as_table), 12

transformif, 112
ts_data, 5, 113
ts_ma (ts_moving_average), 114
ts_moving_average, 114
ts_trend_season, 115
t_s_ts (ts_trend_season), 115
ttest_data, 117
ttest_num, 118

ttest_num(), 116
ttests, 116
txt_knit (inline), 58

unaffix (affix), 7

uncdata (affix), 7

unique_elem, 120

unique_max, 120

unquote (affix), 7

utils::browseURL(), 106

val (as_result), 10

variables (equations), 36

variation (combinatorics), 23

vec2mat, 121

width_breaks (histwidth), 46