Package 'oeli'

August 18, 2025

Type Package

Title Some Utilities for Developing Data Science Software

Version 0.7.5

Description A collection of general-purpose helper functions that I (and maybe others) find useful when developing data science software. Includes tools for simulation, data transformation, input validation, and more.

License GPL (>= 3)

Encoding UTF-8

Config/testthat/edition 3

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports benchmarkme, checkmate, cli, cubature, dplyr, future, future.apply, ggplot2, glue, hexSticker, progressr, R6, Rcpp, SimMultiCorrData, stats, tibble, tools, utils

LinkingTo mvtnorm, Rcpp, RcppArmadillo, testthat

Suggests knitr, rmarkdown, testthat (>= 3.0.0), xml2

URL https://github.com/loelschlaeger/oeli, http://loelschlaeger.de/oeli/

BugReports https://github.com/loelschlaeger/oeli/issues

NeedsCompilation yes

Author Lennart Oelschläger [aut, cre]

Maintainer Lennart Oelschläger <oelschlaeger.lennart@gmail.com>

Repository CRAN

Date/Publication 2025-08-18 09:20:24 UTC

Contents

`check_correlation_matrix` *Check correlation matrix*

Description

These functions check whether the input fulfills the properties of a correlation matrix.

Usage

```
check_correlation_matrix(x, dim = NULL, tolerance = sqrt(.Machine$double.eps))

assert_correlation_matrix(
  x,
  dim = NULL,
  tolerance = sqrt(.Machine$double.eps),
  .var.name = checkmate::vname(x),
  add = NULL
)

test_correlation_matrix(x, dim = NULL, tolerance = sqrt(.Machine$double.eps))
```

Arguments

`x`	[any] Object to check.
`dim`	[`integer(1)`] The matrix dimension.
`tolerance`	[`numeric(1)`] A non-negative tolerance value.

.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.

Value

Same as documented in check_matrix.

See Also

Other matrix helpers: check_covariance_matrix(), check_transition_probability_matrix(), cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(), sample_correlation_matrix(), sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
M <- matrix(c(1,  0.9,  0.9, 0.9,  1,  -0.9, 0.9,  -0.9,  1), nrow = 3)
check_correlation_matrix(M)
test_correlation_matrix(M)
## Not run:
assert_correlation_matrix(M)

## End(Not run)
```

---

check_covariance_matrix

Check covariance matrix

Description

These functions check whether the input fulfills the properties of a covariance matrix.

Usage

```
check_covariance_matrix(x, dim = NULL, tolerance = sqrt(.Machine$double.eps))

assert_covariance_matrix(
  x,
  dim = NULL,
  tolerance = sqrt(.Machine$double.eps),
  .var.name = checkmate::vname(x),
  add = NULL
)

test_covariance_matrix(x, dim = NULL, tolerance = sqrt(.Machine$double.eps))
```

Arguments

x	[any] Object to check.
dim	[integer(1)] The matrix dimension.
tolerance	[numeric(1)] A non-negative tolerance value.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.

Value

Same as documented in check_matrix.

See Also

Other matrix helpers: check_correlation_matrix(), check_transition_probability_matrix(), cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(), sample_correlation_matrix(), sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
M <- matrix(c(1, 2, 3, 2, 1, 2, 3, 2, 1), nrow = 3)
check_covariance_matrix(M)
test_covariance_matrix(M)
## Not run:
assert_covariance_matrix(M)

## End(Not run)
```

---

check_list_of_lists *Check list of lists*

Description

These functions check whether the input is a list that contains list elements.

Usage

```
check_list_of_lists(x, len = NULL)

assert_list_of_lists(
  x,
  len = NULL,
  .var.name = checkmate::vname(x),
  add = NULL
)

test_list_of_lists(x, len = NULL)
```

Arguments

x	[any] Object to check.
len	[integer(1)] Exact expected length of x.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.

Value

Same as documented in check_list.

See Also

Other list helpers: merge_lists()

Examples

```
L <- list(list(1), list(2), 3)
check_list_of_lists(L)
test_list_of_lists(L)
## Not run:
assert_list_of_lists(L)

## End(Not run)
```

check_missing *Check missing formal argument*

Description

These functions check whether a value was specified as an argument to a function.

Usage

```
check_missing(x)

assert_missing(x)

test_missing(x)
```

Arguments

x	[any] A formal argument.

Value

Depending on the function prefix:

- If the check is successful, `assert_missing()` returns `x` invisibly, whereas `check_missing()` and `test_missing()` return `TRUE`.
- If the check is not successful, `assert_missing()` throws an error message, `test_missing()` returns `FALSE`, and `check_missing()` returns a string with the error message.

See Also

Other package helpers: `Dictionary`, `Storage`, `find_namespace_calls()`, `identical_structure()`, `input_check_response()`, `match_arg()`, `package_logo()`, `print_data.frame()`, `print_matrix()`, `system_information()`, `unexpected_error()`, `user_confirm()`

Examples

```
f <- function(x) {
  check_missing(x)
}
f()

g <- function(x) {
  test_missing(x)
}
g()

h <- function(x) {
  assert_missing(x)
```

```
}
## Not run:
h()

## End(Not run)
```

---

check_numeric_vector *Check numeric vector*

Description

These functions check whether the input is a numeric vector.

Usage

```
check_numeric_vector(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  typed.missing = FALSE,
  null.ok = FALSE
)

assert_numeric_vector(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  typed.missing = FALSE,
  null.ok = FALSE,
```

```
  .var.name = checkmate::vname(x),
  add = NULL
)

test_numeric_vector(
  x,
  lower = -Inf,
  upper = Inf,
  finite = FALSE,
  any.missing = TRUE,
  all.missing = TRUE,
  len = NULL,
  min.len = NULL,
  max.len = NULL,
  unique = FALSE,
  sorted = FALSE,
  names = NULL,
  typed.missing = FALSE,
  null.ok = FALSE
)
```

Arguments

x	[any] Object to check.
lower	[numeric(1)] Lower value all elements of x must be greater than or equal to.
upper	[numeric(1)] Upper value all elements of x must be lower than or equal to.
finite	[logical(1)] Check for only finite values? Default is FALSE.
any.missing	[logical(1)] Are vectors with missing values allowed? Default is TRUE.
all.missing	[logical(1)] Are vectors with no non-missing values allowed? Default is TRUE. Note that empty vectors do not have non-missing values.
len	[integer(1)] Exact expected length of x.
min.len	[integer(1)] Minimal length of x.
max.len	[integer(1)] Maximal length of x.
unique	[logical(1)] Must all values be unique? Default is FALSE.
sorted	[logical(1)] Elements must be sorted in ascending order. Missing values are ignored.

names	[character(1)] Check for names. See checkNamed for possible values. Default is "any" which performs no check at all. Note that you can use checkSubset to check for a specific set of names.
typed.missing	[logical(1)] If set to FALSE (default), all types of missing values (NA, NA_integer_, NA_real_, NA_character_ or NA_character_) as well as empty vectors are allowed while type-checking atomic input. Set to TRUE to enable strict type checking.
null.ok	[logical(1)] If set to TRUE, x may also be NULL. In this case only a type check of x is performed, all additional checks are disabled.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.

Value

Same as documented in check_numeric.

See Also

Other vector helpers: check_probability_vector(), chunk_vector(), equidistant_vectors(), insert_vector_entry(), map_indices(), match_numerics(), permutations(), split_vector_at(), subsets(), vector_occurrence()

Examples

```
x <- c(1, 2, "3")
check_numeric_vector(x)
test_numeric_vector(x)
## Not run:
assert_numeric_vector(x)

## End(Not run)
```

---

check_probability_vector

Check probability vector

Description

These functions check whether the input fulfills the properties of a probability matrix.

Usage

```
check_probability_vector(x, len = NULL, tolerance = sqrt(.Machine$double.eps))

assert_probability_vector(
  x,
  len = NULL,
  tolerance = sqrt(.Machine$double.eps),
  .var.name = checkmate::vname(x),
  add = NULL
)

test_probability_vector(x, len = NULL, tolerance = sqrt(.Machine$double.eps))
```

Arguments

x	[any] Object to check.
len	[integer(1)] Exact expected length of x.
tolerance	[numeric(1)] A non-negative tolerance value.
.var.name	[character(1)] Name of the checked object to print in assertions. Defaults to the heuristic implemented in vname.
add	[AssertCollection] Collection to store assertion messages. See AssertCollection.

Value

Same as documented in check_numeric.

See Also

Other vector helpers: check_numeric_vector(), chunk_vector(), equidistant_vectors(), insert_vector_entry(), map_indices(), match_numerics(), permutations(), split_vector_at(), subsets(), vector_occurrence()

Examples

```
p <- c(0.2, 0.3, 0.6)
check_probability_vector(p)
test_probability_vector(p)
## Not run:
assert_probability_vector(p)

## End(Not run)
```

`check_transition_probability_matrix` *Check transition probability matrix*

Description

These functions check whether the input is a transition probability matrix.

Usage

```
check_transition_probability_matrix(
  x,
  dim = NULL,
  tolerance = sqrt(.Machine$double.eps)
)

assert_transition_probability_matrix(
  x,
  dim = NULL,
  tolerance = sqrt(.Machine$double.eps),
  .var.name = checkmate::vname(x),
  add = NULL
)

test_transition_probability_matrix(
  x,
  dim = NULL,
  tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

`x`	[any] Object to check.
`dim`	[`integer(1)`] The matrix dimension.
`tolerance`	[`numeric(1)`] A non-negative tolerance value.
`.var.name`	[`character(1)`] Name of the checked object to print in assertions. Defaults to the heuristic implemented in `vname`.
`add`	[`AssertCollection`] Collection to store assertion messages. See `AssertCollection`.

Value

Same as documented in `check_matrix`.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(), sample_correlation_matr sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
T <- matrix(c(0.8,  0.2,  0.1, 0.1,  0.7,  0.4, 0.1,  0.1,  0.6), nrow = 3)
check_transition_probability_matrix(T)
test_transition_probability_matrix(T)
## Not run:
assert_transition_probability_matrix(T)

## End(Not run)
```

---

chunk_vector *Split a vector into chunks*

Description

This function either

- splits a vector into n chunks of equal size (type = 1),
- splits a vector into chunks of size n (type = 2).

Usage

```
chunk_vector(x, n, type = 1, strict = FALSE)
```

Arguments

x	[atomic()‘] A vector of elements.
n	[integer(1)] A number smaller or equal length(x).
type	[1 | 2] Either • 1 (default) to split x into n chunks of equal size, • or 2 to split x into chunks of size n.
strict	[logical(1)] Set to TRUE to fail if length(x) is not a multiple of n, or FALSE (default), else.

Value

A list.

See Also

Other vector helpers: check_numeric_vector(), check_probability_vector(), equidistant_vectors(), insert_vector_entry(), map_indices(), match_numerics(), permutations(), split_vector_at(), subsets(), vector_occurrence()

Examples

```
x <- 1:12
chunk_vector(x, n = 3, type = 1)
chunk_vector(x, n = 3, type = 2)
try(chunk_vector(x, n = 5, strict = TRUE))
```

---

correlated_regressors *Simulate correlated regressor values*

Description

This function simulates regressor values from various marginal distributions with custom correlations.

Usage

```
correlated_regressors(
  labels,
  n = 100,
  marginals = list(),
  correlation = diag(length(labels)),
  verbose = FALSE
)
```

Arguments

labels	[character()] Unique labels for the regressors.
n	[integer(1)] The number of values per regressor.
marginals	[list()] Optionally marginal distributions for regressors. If not specified, standard normal marginal distributions are used. Each list entry must be named according to a regressor label, and the following distributions are currently supported: **discrete distributions** • Poisson: list(type = "poisson", lambda = ...) • categorical: list(type = "categorical", p = c(...)) **continuous distributions** • normal: list(type = "normal", mean = ..., sd = ...) • uniform: list(type = "uniform", min = ..., max = ...)

correlation [matrix()]
A correlation matrix of dimension length(labels), where the (p, q)-th entry defines the correlation between regressor labels[p] and labels[q].

verbose [logical(1)]
Print information about the simulated regressors?

Value

A data.frame with n rows and length(labels) columns.

References

This function heavily depends on the {SimMultiCorrData} package.

See Also

Other simulation helpers: Simulator, ddirichlet_cpp(), dmixnorm_cpp(), dmvnorm_cpp(), dtnorm_cpp(), dwishart_cpp(), gaussian_tv(), simulate_markov_chain()

Examples

```
labels <- c("P", "C", "N1", "N2", "U")
n <- 100
marginals <- list(
  "P" = list(type = "poisson", lambda = 2),
  "C" = list(type = "categorical", p = c(0.3, 0.2, 0.5)),
  "N1" = list(type = "normal", mean = -1, sd = 2),
  "U" = list(type = "uniform", min = -2, max = -1)
)
correlation <- matrix(
  c(1, -0.3, -0.1, 0, 0.5,
    -0.3, 1, 0.3, -0.5, -0.7,
    -0.1, 0.3, 1, -0.3, -0.3,
    0, -0.5, -0.3, 1, 0.1,
    0.5, -0.7, -0.3, 0.1, 1),
  nrow = 5, ncol = 5
)
data <- correlated_regressors(
  labels = labels, n = n, marginals = marginals, correlation = correlation
)
head(data)
```

---

cov_to_chol *Cholesky root of covariance matrix*

Description

These functions compute the Cholesky root elements of a covariance matrix and, conversely, build a covariance matrix from its Cholesky root elements.

Usage

```
cov_to_chol(cov, unique = TRUE)

chol_to_cov(chol)

unique_chol(chol)
```

Arguments

cov	[matrix()] A covariance matrix. It can also be the zero matrix, in which case the Cholesky root is defined as the zero matrix.
unique	[logical(1)] Ensure that the Cholesky decomposition is unique by restricting the diagonal elements to be positive?
chol	[numeric()] Cholesky root elements.

Value

For cov_to_chol a numeric vector of Cholesky root elements.

For chol_to_cov a covariance matrix.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), check_transition_probability_
diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(), sample_correlation_matr
sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
cov <- sample_covariance_matrix(4)
chol <- cov_to_chol(cov)
all.equal(cov, chol_to_cov(chol))
```

---

ddirichlet_cpp *Dirichlet distribution*

Description

The function ddirichlet() computes the density of a Dirichlet distribution.

The function rdirichlet() samples from a Dirichlet distribution.

The functions with suffix _cpp perform no input checks, hence are faster.

Usage

```
ddirichlet_cpp(x, concentration, log = FALSE)

rdirichlet_cpp(concentration)

ddirichlet(x, concentration, log = FALSE)

rdirichlet(n = 1, concentration)
```

Arguments

x	[numeric()] A probability vector.
concentration	[numeric()] A concentration vector of the same length as x.
log	[logical(1)] Return the logarithm of the density value?
n	[integer(1)] The number of samples.

Value

For ddirichlet(): The density value.

For rdirichlet(): If n = 1 a vector of length p, else a matrix of dimension n times p with samples as rows.

See Also

Other simulation helpers: Simulator, correlated_regressors(), dmixnorm_cpp(), dmvnorm_cpp(), dtnorm_cpp(), dwishart_cpp(), gaussian_tv(), simulate_markov_chain()

Examples

```
x <- c(0.5, 0.3, 0.2)
concentration <- 1:3

# compute density
ddirichlet(x = x, concentration = concentration)
ddirichlet(x = x, concentration = concentration, log = TRUE)

# sample
rdirichlet(concentration = 1:3)
rdirichlet(n = 4, concentration = 1:2)
```

delete_columns_data.frame *Deleting* data.frame *columns*

Description

This function deletes columns of a data.frame by name.

Usage

```
delete_columns_data.frame(df, column_names)
```

Arguments

df	[data.frame] A data.frame.
column_names	[character()] The name(s) of column(s) of df to delete.

Value

The input df without the columns defined by column_names.

See Also

Other data.frame helpers: group_data.frame(), occurrence_info(), round_data.frame()

Examples

```
df <- data.frame("label" = c("A", "B"), "number" = 1:10)
delete_columns_data.frame(df = df, column_names = "label")
delete_columns_data.frame(df = df, column_names = "number")
delete_columns_data.frame(df = df, column_names = c("label", "number"))
```

Dictionary *Dictionary R6 Object*

Description

Provides a simple key-value interface based on R6.

Active bindings

keys [character()]
: Available keys.

alias [list()]
: Available keys per alias value.

Methods

Public methods:

- `Dictionary$new()`
- `Dictionary$add()`
- `Dictionary$get()`
- `Dictionary$remove()`
- `Dictionary$print()`

Method `new()`**:** Initializing a new `Dictionary` object.

Usage:

```
Dictionary$new(
  key_name,
  alias_name = NULL,
  value_names = character(),
  value_assert = alist(),
  allow_overwrite = TRUE,
  keys_reserved = character(),
  alias_choices = NULL,
  dictionary_name = NULL
)
```

Arguments:

`key_name [character(1)]`
: The name for the key variable.

`alias_name [NULL | character(1)]`
: Optionally the name for the alias variable.

`value_names [character(0)]`
: The names of the values connected to a key.

`value_assert [alist(1)]`
: For each element in `value_names`, `values_assert` *can* have an identically named element of the form `checkmate::assert*(...)`, where `...` can be any argument for the assertion function except for the `x` argument.

`allow_overwrite [logical(1)]`
: Allow overwriting existing keys with new values? Duplicate keys are never allowed.

`keys_reserved [character()]`
: Names that must not be used as keys.

`alias_choices [NULL or character()]`
: Optionally possible values for the alias. Can also be `NULL`, then all alias values are allowed.

`dictionary_name [NULL or character()]`
: Optionally the name for the dictionary.

Method `add()`**:** Adding an element to the dictionary.

Usage:

```
Dictionary$add(...)
```

Arguments:

... Values for
- the key variable key_name (must be a single character),
- the alias variable alias_name (optionally, must then be a character vector),
- all the variables specified for value_names (if any, they must comply to the value_assert checks).

Method `get()`**:** Getting elements from the dictionary.

Usage:

```
Dictionary$get(key, value = NULL)
```

Arguments:

`key [character(1)]`
A value for the key variable key_name. Use the $keys method for available keys.

`value [NULL | character(1)]`
One of the elements in value_names, selecting the required value. Can also be NULL (default) for all values connected to the key, returned as a list.

Method `remove()`**:** Removing elements from the dictionary (and associated alias, if any).

Usage:

```
Dictionary$remove(key)
```

Arguments:

`key [character(1)]`
A value for the key variable key_name. Use the $keys method for available keys.

Method `print()`**:** Printing details of the dictionary.

Usage:

```
Dictionary$print()
```

See Also

Other package helpers: Storage, check_missing(), find_namespace_calls(), identical_structure(), input_check_response(), match_arg(), package_logo(), print_data.frame(), print_matrix(), system_information(), unexpected_error(), user_confirm()

Examples

```
# Managing variable metadata for a dataset

meta_dict <- Dictionary$new(
  key_name = "var_name",
  alias_name = "category",
  value_names = c("label", "type"),
  value_assert = alist(
    label = checkmate::assert_string(),
    type = checkmate::assert_choice(choices = c("numeric", "factor", "character"))
  ),
  allow_overwrite = FALSE,
  keys_reserved = c("id"),
```

```
  alias_choices = c("demographics", "outcome", "other"),
  dictionary_name = "Variable Metadata"
)

# Add entries to the dictionary
meta_dict$add(
  var_name = "age",
  label = "Age of respondent",
  type = "numeric",
  category = "demographics"
)

meta_dict$add(
  var_name = "gender",
  label = "Gender identity",
  type = "factor",
  category = "demographics"
)

meta_dict$add(
  var_name = "income",
  label = "Annual income in USD",
  type = "numeric",
  category = c("demographics", "outcome")
)

# Print dictionary
meta_dict$print()

# Retrieve full metadata for a variable
meta_dict$get("income")

# Retrieve a specific piece of metadata
meta_dict$get("income", value = "label")

# Show variables by category
meta_dict$alias
```

`diff_cov` *Difference and un-difference covariance matrix*

Description

These functions difference and un-difference random vectors and covariance matrices.

Usage

```
diff_cov(cov, ref = 1)

undiff_cov(cov_diff, ref = 1)
```

```
delta(ref = 1, dim)

M(ranking = seq_len(dim), dim)
```

Arguments

cov, cov_diff	[matrix()] A (differenced) covariance matrix of dimension dim (or dim - 1, respectively).
ref	[integer(1)] The reference row between 1 and dim for differencing that maps cov to cov_diff, see details.
dim	[integer(1)] The matrix dimension.
ranking	[integer()] The integers 1 to dim in arbitrary order.

Details

Assume $x \sim N(0, \Sigma)$ is a multivariate normally distributed random vector of dimension n. We may want to consider the differenced vector

$$\tilde{x} = (x_1 - x_k, x_2 - x_k, \ldots, x_n - x_k)',$$

excluding the kth element (hence, $\tilde{x}$ is of dimension $(n-1) \times 1$). Formally, $\tilde{x} = \Delta_k x$, where Δ_k is a difference operator that depends on the reference row k. More precise, Δ_k is the identity matrix of dimension n without row k and with -1s in column k. The difference operator Δ_k can be computed via `delta(ref = k, dim = n)`.

Then, $\tilde{x} \sim N(0, \tilde{\Sigma})$, where

$$\tilde{\Sigma} = \Delta_k \Sigma \Delta_k'$$

is the differenced covariance matrix with respect to row $k = 1, \ldots, n$. The differenced covariance matrix $\tilde{\Sigma}$ can be computed via `diff_delta(Sigma, ref = k)`.

Since Δ_k is a non-bijective mapping, Σ cannot be uniquely restored from $\tilde{\Sigma}$. However, it is possible to compute a non-unique solution Σ_0, such that $\Delta_k \Sigma_0 \Delta_k = \tilde{\Sigma}$. For such a non-unique solution, we add a column and a row of zeros at column and row number k to $\tilde{\Sigma}$, respectively. An "un-differenced" covariance matrix Σ_0 can be computed via `undiff_delta(Sigma_diff, ref = k)`.

As a alternative to Δ_k, the function `M()` returns a matrix for taking differences such that the resulting vector is negative.

Value

A (differenced or un-differenced) covariance `matrix`.

See Also

Other matrix helpers: `check_correlation_matrix()`, `check_covariance_matrix()`, `check_transition_probability_matrix()`, `cov_to_chol()`, `insert_matrix_column()`, `matrix_diagonal_indices()`, `matrix_indices()`, `sample_correlation_matrix()`, `sample_covariance_matrix()`, `sample_transition_probability_matrix()`, `stationary_distribution()`

Examples

```
n <- 4
Sigma <- sample_covariance_matrix(dim = n)
k <- 2
x <- c(1, 3, 2, 4)

# build difference operator
delta_k <- delta(ref = k, dim = n)

# difference vector
delta_k %*% x

# difference Sigma
(Sigma_diff <- diff_cov(Sigma, ref = k))

# un-difference Sigma
(Sigma_0 <- undiff_cov(Sigma_diff, ref = k))

# difference again
Sigma_diff_2 <- diff_cov(Sigma_0, ref = k)
all.equal(Sigma_diff, Sigma_diff_2)

# difference such that the resulting vector is negative
M(ranking = order(x, decreasing = TRUE), dim = n) %*% x
```

---

dmixnorm_cpp *Mixture of normal distributions*

Description

The function dmixnorm() computes the density of a mixture of multivariate normal distribution.

The function pmixnorm() computes the cumulative distribution function of a mixture of multivariate normal distribution.

The function rmixnorm() samples from a mixture of multivariate normal distribution.

The functions with suffix _cpp perform no input checks, hence are faster.

The univariate normal mixture is available as the special case p = 1.

Usage

```
dmixnorm_cpp(x, mean, Sigma, proportions)

pmixnorm_cpp(x, mean, Sigma, proportions, abseps = 0.001)

rmixnorm_cpp(mean, Sigma, proportions)

dmixnorm(x, mean, Sigma, proportions)
```

```
pmixnorm(x, mean, Sigma, proportions, abseps = 0.001)

rmixnorm(n = 1, mean, Sigma, proportions)
```

Arguments

x	[numeric(p)] A quantile vector of length p, where p is the dimension.
mean	[matrix(nrow = p, ncol = c)] The mean vectors for each component in columns.
Sigma	[matrix(nrow = p^2, ncol = c)] The vectorized covariance matrices for each component in columns.
proportions	[numeric(c)] The non-negative mixing proportions for each components. If proportions do not sum to unity, they are rescaled to do so.
abseps	[numeric(1)] The absolute error tolerance.
n	[integer(1)] The number of requested samples.

Details

pmixnorm() is based on mvtnorm::pmvnorm with the randomized Quasi-Monte-Carlo procedure by Genz and Bretz. The argument abseps controls the accuracy of the Gaussian integral approximation.

Value

For dmixnorm(): The density value.

For pmixnorm(): The value of the distribution function.

For rmixnorm(): If n = 1 a vector of length p (note that it is a column vector for rmixnorm_cpp()), else a matrix of dimension n times p with samples as rows.

See Also

Other simulation helpers: Simulator, correlated_regressors(), ddirichlet_cpp(), dmvnorm_cpp(), dtnorm_cpp(), dwishart_cpp(), gaussian_tv(), simulate_markov_chain()

Examples

```
x <- c(0, 0)
mean <- matrix(c(-1, -1, 0, 0), ncol = 2)
Sigma <- matrix(c(diag(2), diag(2)), ncol = 2)
proportions <- c(0.7, 0.3)

# compute density
dmixnorm(x = x, mean = mean, Sigma = Sigma, proportions = proportions)
```

```
# compute CDF
pmixnorm(x = x, mean = mean, Sigma = Sigma, proportions = proportions)

# sample
rmixnorm(n = 3, mean = mean, Sigma = Sigma, proportions = proportions)
```

---

dmvnorm_cpp *Multivariate normal distribution*

Description

The function dmvnorm() computes the density of a multivariate normal distribution.

The function pmvnorm() computes the cumulative distribution function of a multivariate normal distribution.

The function rmvnorm() samples from a multivariate normal distribution.

The functions with suffix _cpp perform no input checks, hence are faster.

The univariate normal distribution is available as the special case p = 1.

Usage

```
dmvnorm_cpp(x, mean, Sigma, log = FALSE)

pmvnorm_cpp(x, mean, Sigma, abseps = 0.001)

rmvnorm_cpp(mean, Sigma, log = FALSE)

dmvnorm(x, mean, Sigma, log = FALSE)

pmvnorm(x, mean, Sigma, abseps = 0.001)

rmvnorm(n = 1, mean, Sigma, log = FALSE)
```

Arguments

x	[numeric()] A quantile vector of length p.
mean	[numeric()] The mean vector of length p. For the functions without suffix _cpp, it can also be of length 1 for convenience, then rep(mean, p) is considered.
Sigma	[matrix()] The covariance matrix of dimension p. For rmvnorm(), arbitrary dimensions (i.e., full rows and corresponding columns) of Sigma can be 0.

For the functions without suffix _cpp and if p = 1, it can also be a single numeric for convenience. Note that Sigma is this case is a variance, which is a different format than in stats::dnorm() or stats::rnorm, which require a standard deviation.

log [logical(1)]
Consider the log-normal distribution?

abseps [numeric(1)]
The absolute error tolerance.

n [integer(1)]
The number of requested samples.

Details

pmvnorm() just calls mvtnorm::pmvnorm with the randomized Quasi-Monte-Carlo procedure by Genz and Bretz. The argument abseps controls the accuracy of the Gaussian integral approximation.

Value

For dmvnorm(): The density value.

For pmvnorm(): The value of the distribution function.

For rmvnorm(): If n = 1 a vector of length p (note that it is a column vector for rmvnorm_cpp()), else a matrix of dimension n times p with samples as rows.

See Also

Other simulation helpers: Simulator, correlated_regressors(), ddirichlet_cpp(), dmixnorm_cpp(), dtnorm_cpp(), dwishart_cpp(), gaussian_tv(), simulate_markov_chain()

Examples

```
x <- c(0, 0)
mean <- c(0, 0)
Sigma <- diag(2)

# compute density
dmvnorm(x = x, mean = mean, Sigma = Sigma)
dmvnorm(x = x, mean = mean, Sigma = Sigma, log = TRUE)

# compute CDF
pmvnorm(x = x, mean = mean, Sigma = Sigma)

# sample
rmvnorm(n = 3, mean = mean, Sigma = Sigma)
rmvnorm(mean = mean, Sigma = Sigma, log = TRUE)
```

`do.call_timed` *Measure computation time*

Description

This function measures the computation time of a call.

Usage

```
do.call_timed(what, args, units = "secs")
```

Arguments

`what, args`	Passed to `do.call`.
`units`	Passed to `difftime`.

Details

This function is a wrapper for `do.call`.

Value

A list of the two elements `"result"` (the results of the `do.call` call) and `"time"` (the computation time).

See Also

Other function helpers: `function_arguments()`, `function_body()`, `function_defaults()`, `quiet()`, `timed()`, `try_silent()`, `variable_name()`

Examples

```
## Not run:
what <- function(s) {
  Sys.sleep(s)
  return(s)
}
args <- list(s = 1)
do.call_timed(what = what, args = args)

## End(Not run)
```

dtnorm_cpp *Truncated normal distribution*

Description

The function `dtnorm()` computes the density of a truncated normal distribution.

The function `rtnorm()` samples from a truncated normal distribution.

The function `dttnorm()` and `rttnorm()` compute the density and sample from a two-sided truncated normal distribution, respectively.

The functions with suffix _cpp perform no input checks, hence are faster.

Usage

```
dtnorm_cpp(x, mean, sd, point, above, log = FALSE)

dttnorm_cpp(x, mean, sd, lower, upper, log = FALSE)

rtnorm_cpp(mean, sd, point, above, log = FALSE)

rttnorm_cpp(mean, sd, lower, upper, log = FALSE)

dtnorm(x, mean, sd, point, above, log = FALSE)

dttnorm(x, mean, sd, lower, upper, log = FALSE)

rtnorm(mean, sd, point, above, log = FALSE)

rttnorm(mean, sd, lower, upper, log = FALSE)
```

Arguments

`x`	[`numeric(1)`] A quantile.
`mean`	[`numeric(1)`] The mean.
`sd`	[`numeric(1)`] The non-negative standard deviation.
`point`, `lower`, `upper`	[`numeric(1)`] The truncation point.
`above`	[`logical(1)`] Truncate from above? Else, from below.
`log`	[`logical(1)`] Return the logarithm of the density value?

Value

For `dtnorm()` and `dttnorm()`: The density value.

For `rtnorm()` and `rttnorm()`: The random draw

See Also

Other simulation helpers: `Simulator`, `correlated_regressors()`, `ddirichlet_cpp()`, `dmixnorm_cpp()`, `dmvnorm_cpp()`, `dwishart_cpp()`, `gaussian_tv()`, `simulate_markov_chain()`

Examples

```
x <- c(0, 0)
mean <- c(0, 0)
Sigma <- diag(2)

# compute density
dmvnorm(x = x, mean = mean, Sigma = Sigma)
dmvnorm(x = x, mean = mean, Sigma = Sigma, log = TRUE)

# sample
rmvnorm(n = 3, mean = mean, Sigma = Sigma)
rmvnorm(mean = mean, Sigma = Sigma, log = TRUE)
```

---

dwishart_cpp *Wishart distribution*

Description

The function `dwishart()` computes the density of a Wishart distribution.

The function `rwishart()` samples from a Wishart distribution.

The functions with suffix _cpp perform no input checks, hence are faster.

Usage

```
dwishart_cpp(x, df, scale, log = FALSE, inv = FALSE)

rwishart_cpp(df, scale, inv = FALSE)

dwishart(x, df, scale, log = FALSE, inv = FALSE)

rwishart(df, scale, inv = FALSE)
```

Arguments

x	[matrix()] A covariance matrix of dimension p.
df	[integer()] The degrees of freedom greater of equal p.
scale	[matrix()] The scale covariance matrix of dimension p.
log	[logical(1)] Return the logarithm of the density value?
inv	[logical(1)] Use this inverse Wishart distribution?

Value

For dwishart(): The density value.

For rwishart(): A matrix, the random draw.

See Also

Other simulation helpers: Simulator, correlated_regressors(), ddirichlet_cpp(), dmixnorm_cpp(), dmvnorm_cpp(), dtnorm_cpp(), gaussian_tv(), simulate_markov_chain()

Examples

```
x <- diag(2)
df <- 6
scale <- matrix(c(1, -0.3, -0.3, 0.8), ncol = 2)

# compute density
dwishart(x = x, df = df, scale = scale)
dwishart(x = x, df = df, scale = scale, log = TRUE)
dwishart(x = x, df = df, scale = scale, inv = TRUE)

# sample
rwishart(df = df, scale = scale)
rwishart(df = df, scale = scale, inv = TRUE)

# expectation of Wishart is df * scale
n <- 100
replicate(n, rwishart(df = df, scale = scale), simplify = FALSE) |>
  Reduce(f = "+") / n
df * scale

# expectation of inverse Wishart is scale / (df - p - 1)
n <- 100
replicate(n, rwishart(df = df, scale = scale, TRUE), simplify = FALSE) |>
  Reduce(f = "+") / n
scale / (df - 2 - 1)
```

`equidistant_vectors` *Generate equidistant vectors in Euclidean space*

Description

This function constructs the coordinates of vertices of a regular simplex in $\mathbb{R}^{\texttt{dim}}$ and returns the first `n` of them,

- scaled so that the pairwise Euclidean distance between any two vertices equals `dist`,
- and centered so their centroid is at `center`.

Usage

```
equidistant_vectors(dim, n = dim + 1, dist = 1, center = rep(0, dim))
```

Arguments

`dim`	[`integer(1)`] The dimension.
`n`	[`integer(1)`] The number of vertices to return. Cannot be larger than `dim + 1`.
`dist`	[`numeric(1)`] Desired pairwise Euclidean distance between any two vertices.
`center`	[`numeric(dim)`] Desired center.

Value

A matrix, where each column is a vertex of the simplex.

See Also

Other vector helpers: `check_numeric_vector()`, `check_probability_vector()`, `chunk_vector()`, `insert_vector_entry()`, `map_indices()`, `match_numerics()`, `permutations()`, `split_vector_at()`, `subsets()`, `vector_occurrence()`

Examples

```
dim <- n <- 3
(dist <- runif(1))
(center <- rnorm(dim))
(V <- equidistant_vectors(dim = dim, n = n, dist = dist, center = center))
rowMeans(V)
dist(t(V))
```

`find_namespace_calls` *Namespace calls*

Description

This function searches for namespace calls in `.R` files, i.e., code lines of the format `<package name>::<function name>`.

Usage

```
find_namespace_calls(path = "R", triple_colon = FALSE, as_list = FALSE)
```

Arguments

`path`	[`character(1)`] The path name to a folder. All `.R` files in this folder and sub-directories will be searched.
`triple_colon`	[`logical(1)`] Also search for `:::`?
`as_list`	[`logical(1)`] Simplify the output into a `list` of unique function names per package?

Value

A `data.frame`. If `as_list` = `TRUE`, a `list`.

See Also

Other package helpers: `Dictionary`, `Storage`, `check_missing()`, `identical_structure()`, `input_check_response()`, `match_arg()`, `package_logo()`, `print_data.frame()`, `print_matrix()`, `system_information()`, `unexpected_error()`, `user_confirm()`

Examples

```
## Not run:
find_namespace_calls()
find_namespace_calls(as_list = TRUE)

## End(Not run)
```

function_arguments *Get function arguments*

Description

This function returns the names of function arguments.

Usage

```
function_arguments(f, with_default = TRUE, with_ellipsis = TRUE)
```

Arguments

f	[function] A function.
with_default	[logical(1)] Include function arguments that have default values?
with_ellipsis	[logical(1)] Include the "..." argument if present?

Value

A character vector.

See Also

Other function helpers: do.call_timed(), function_body(), function_defaults(), quiet(), timed(), try_silent(), variable_name()

Examples

```
f <- function(a, b = 1, c = "", ...) { }
function_arguments(f)
function_arguments(f, with_default = FALSE)
function_arguments(f, with_ellipsis = FALSE)
```

function_body *Extract function body*

Description

This function extracts the body of a function as a single character.

Usage

```
function_body(fun, braces = FALSE, nchar = getOption("width") - 4)
```

Arguments

fun	[function] A function.
braces	[logical(1)] Remove "{" and "}" at start and end (if any)?
nchar	[integer(1)] The maximum number of characters before abbreviation, at least 3.

Value

A character, the body of f.

See Also

Other function helpers: do.call_timed(), function_arguments(), function_defaults(), quiet(), timed(), try_silent(), variable_name()

Examples

```
fun <- mean.default
function_body(fun)
function_body(fun, braces = TRUE)
function_body(fun, nchar = 30)
```

---

function_defaults *Get default function arguments*

Description

This function returns the default function arguments (if any).

Usage

```
function_defaults(f, exclude = NULL)
```

Arguments

f	[function] A function.
exclude	[NULL | character()] Argument names to exclude. Can be NULL (default) to not exclude any argument names.

Value

A named list.

See Also

Other function helpers: do.call_timed(), function_arguments(), function_body(), quiet(), timed(), try_silent(), variable_name()

Examples

```
f <- function(a, b = 1, c = "", ...) { }
function_defaults(f)
function_defaults(f, exclude = "b")
```

---

gaussian_tv *Gaussian total variation*

Description

Computes the total variation (TV) between two multivariate Gaussian distributions f_1, f_2:

$$\mathrm{TV}(f_1, f_2) = \tfrac{1}{2} \int_{\mathbb{R}^p} |f_1(x) - f_2(x)|\, dx.$$

The value ranges from 0 (identical distributions) to 1 (no overlap).

Usage

```
gaussian_tv(
  mean1,
  mean2,
  Sigma1,
  Sigma2,
  method = c("auto", "mc", "cubature"),
  n = 10000,
  tol_equal = 1e-06,
  eps = 1e-06
)
```

Arguments

mean1, mean2	[numeric(p)] The mean vectors.
Sigma1, Sigma2	[matrix(nrow = p, ncol = p)] The covariance matrices.
method	[character(1)] Computation method. One of: • "auto": use closed-form formula when covariances are equal, otherwise use "cubature" for $p \leq 2$ and "mc" for higher dimensions. • "mc": estimate via Monte Carlo importance sampling from the mixture $0.5(f_1 + f_2)$.

- `"cubature"`: compute overlap via adaptive cubature integration over a bounding box, then convert to TV. Exact but slow for $p \geq 2$.

`n` [integer(1)]
Number of Monte Carlo samples to draw.

`tol_equal` [numeric(1)]
Numerical tolerance used to decide whether `Sigma1` and `Sigma2` are considered equal (enabling the closed-form formula in `"auto"` mode).

`eps` [numeric(1)]
Only used when `method = "cubature"`. Specifies the total probability mass allowed to lie outside the integration hyper-rectangle across all dimensions. This determines the numerical integration bounds: the function chooses limits so that the probability of a point from either Gaussian falling outside the box is at most `eps`. The bound is split evenly across dimensions via a union bound, so the per-dimension tail probability is approximately `eps / p`. Smaller values produce wider bounds (slower but more accurate integration), while larger values yield narrower bounds (faster but potentially less accurate).

Value

The total variation in [0, 1].

See Also

Other simulation helpers: `Simulator`, `correlated_regressors()`, `ddirichlet_cpp()`, `dmixnorm_cpp()`, `dmvnorm_cpp()`, `dtnorm_cpp()`, `dwishart_cpp()`, `simulate_markov_chain()`

Examples

```
### univariate case
mean1 <- 0
mean2 <- 1
Sigma1 <- Sigma2 <- matrix(1)
gaussian_tv(mean1, mean2, Sigma1, Sigma2)

### bivariate case
mean1 <- c(0, 0)
mean2 <- c(1, 1)
Sigma1 <- matrix(c(1, 0.2, 0.2, 1), ncol = 2)
Sigma2 <- matrix(c(1.5, -0.3, -0.3, 1), ncol = 2)
gaussian_tv(mean1, mean2, Sigma1, Sigma2, method = "mc", n = 1e3)
```

---

group_data.frame *Grouping of a* data.frame

Description

This function groups a `data.frame` according to values of one column.

Usage

```
group_data.frame(df, by, keep_by = TRUE)
```

Arguments

df	[data.frame] A data.frame.
by	[character(1)] The name of a column of df to group by.
keep_by	[logical(1)] Keep the grouping column by?

Value

A list of data.frames, subsets of df.

See Also

Other data.frame helpers: delete_columns_data.frame(), occurrence_info(), round_data.frame()

Examples

```
df <- data.frame("label" = c("A", "B"), "number" = 1:10)
group_data.frame(df = df, by = "label")
group_data.frame(df = df, by = "label", keep_by = FALSE)
```

---

identical_structure *Check if two objects have identical structure*

Description

This function determines whether two objects have the same structure,

- which includes the mode, class and dimension
- but does *not* include concrete values or attributes.

Usage

```
identical_structure(x, y)
```

Arguments

x, y	[any] Two objects.

Value

Either TRUE if x and y have the same structure, and FALSE, else.

References

Inspired by https://stackoverflow.com/a/45548885/15157768.

See Also

Other package helpers: Dictionary, Storage, check_missing(), find_namespace_calls(), input_check_response(), match_arg(), package_logo(), print_data.frame(), print_matrix(), system_information(), unexpected_error(), user_confirm()

Examples

```
identical_structure(integer(1), 1L)
identical_structure(diag(2), matrix(rnorm(4), 2, 2))
identical_structure(diag(2), data.frame(diag(2)))
```

---

input_check_response *Standardized response to input check*

Description

This function provides a standardized response to input checks, ensuring consistency.

Usage

```
input_check_response(
  check,
  var_name = NULL,
  error = TRUE,
  prefix = "Input {.var {var_name}} is bad:"
)
```

Arguments

check	[TRUE | character(1) | list()] Matches the return value of the check* functions from the {checkmate} package, i.e., either TRUE if the check was successful, or a character (the error message) else. Can also be a list of multiple such values for alternative criteria, where at least one must be TRUE for a successful check.
var_name	[NULL | character(1)] Optionally specifies the name of the input being checked. This name will be used for the default value of the prefix argument.
error	[logical(1)] If check is not TRUE (or no element in check is TRUE, if check is a list), throw an error?
prefix	[character(1)] A prefix for the thrown error message, only relevant if error is TRUE.

Value

TRUE if check is TRUE (or any element in check is TRUE, if check is a list) . Else, depending on error:

- If error is TRUE, throws an error.
- If error is FALSE, returns FALSE.

See Also

Other package helpers: Dictionary, Storage, check_missing(), find_namespace_calls(), identical_structure(), match_arg(), package_logo(), print_data.frame(), print_matrix(), system_information(), unexpected_error(), user_confirm()

Examples

```
x <- "1"
y <- 1

### check is successful
input_check_response(
  check = checkmate::check_character(x),
  var_name = "x",
  error = TRUE
)

### alternative checks
input_check_response(
  check = list(
    checkmate::check_character(x),
    checkmate::check_character(y)
  ),
  var_name = "x",
  error = TRUE
)

### standardized check response
## Not run:
input_check_response(
  check = checkmate::check_character(y),
  var_name = "y",
  error = TRUE
)

input_check_response(
  check = list(
    checkmate::check_flag(x),
    checkmate::check_character(y)
  ),
  var_name = "y",
  error = TRUE
)
```

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

`insert_matrix_column` *Insert column in matrix*

Description

This function inserts a column into a matrix.

Usage

```
insert_matrix_column(A, x, p)
```

Arguments

`A`	[`matrix()`] A `matrix`.
`x`	[`atomic()`] The column to be added, of length `nrow(A)`. Can also be a single value.
`p`	[`integer()`)] The position(s) where to add the column, one or more of: • `p = 0` appends the column left • `p = ncol(A)` appends the column right • `p = n` inserts the column between the `n`-th and `(n + 1)`-th column of `A`.

Value

A `matrix`.

See Also

Other matrix helpers: `check_correlation_matrix()`, `check_covariance_matrix()`, `check_transition_probability_matrix()`, `cov_to_chol()`, `diff_cov()`, `matrix_diagonal_indices()`, `matrix_indices()`, `sample_correlation_matrix()`, `sample_covariance_matrix()`, `sample_transition_probability_matrix()`, `stationary_distribution()`

Examples

```
A <- diag(3)
x <- 1:3
insert_matrix_column(A, x, 0)
insert_matrix_column(A, x, 1)
insert_matrix_column(A, x, 2)
insert_matrix_column(A, x, 3)

### also single value
```

```
x <- 2
insert_matrix_column(A, x, 0)

### also multiple positions
insert_matrix_column(A, x, 0:3)

### also trivial case
insert_matrix_column(matrix(nrow = 0, ncol = 0), integer(), integer())
```

`insert_vector_entry` *Insert entry in vector*

Description

This function inserts a value into a vector.

Usage

```
insert_vector_entry(v, x, p)
```

Arguments

v	[atomic()] A vector.
x	[atomic(1)] The entry to be added.
p	[integer())] The position(s) where to add the value, one or more of: • p = 0 appends the value left • p = length(v) appends the value right • p = n inserts the value between the n-th and (n + 1)-th entry of v.

Value

A vector.

See Also

Other vector helpers: check_numeric_vector(), check_probability_vector(), chunk_vector(), equidistant_vectors(), map_indices(), match_numerics(), permutations(), split_vector_at(), subsets(), vector_occurrence()

Examples

```
v <- 1:3
x <- 0
insert_vector_entry(v, x, 0)
insert_vector_entry(v, x, 1)
insert_vector_entry(v, x, 2)
insert_vector_entry(v, x, 3)

### also multiple positions
insert_vector_entry(v, x, 0:3)

### also trivial case
insert_vector_entry(integer(), integer(), integer())
```

---

map_indices *Map indices*

Description

This function maps indices from an input vector to corresponding sequences of grouped indices. Each element from the input specifies a group to be mapped from the sequence, determined by the grouping size `n`.

Usage

```
map_indices(indices, n)
```

Arguments

`indices`	`[integer()]` An index vector, where each element specifies a group to be mapped from the sequence.
`n`	`[integer]` The size of each group of consecutive indices.

Details

This function is useful when working with indices arranged in fixed-size groups, where each group can be referenced by a single index. For example, if indices are structured in chunks of 3, calling this function with `n = 3` will map the corresponding groups of 3 consecutive indices for the given input indices, see the examples.

Value

An `integer vector`, containing the mapped indices according to the specified group size.

See Also

Other vector helpers: check_numeric_vector(), check_probability_vector(), chunk_vector(), equidistant_vectors(), insert_vector_entry(), match_numerics(), permutations(), split_vector_at(), subsets(), vector_occurrence()

Examples

```
# Example: Map indices based on groups of 3
map_indices(c(1, 3, 5), 3)
```

---

match_arg *Argument matching*

Description

This function matches function arguments and is a modified version of match.arg.

Usage

```
match_arg(arg, choices, several.ok = FALSE, none.ok = FALSE)
```

Arguments

arg	[character()] The function argument.
choices	[character()] Allowed values for arg.
several.ok	[logical(1)] Is arg allowed to have more than one element?
none.ok	[logical(1)] Is arg allowed to have zero elements?

Value

The un-abbreviated version of the exact or unique partial match if there is one. Otherwise, an error is signaled if several.ok is FALSE or none.ok is FALSE. When several.ok is TRUE and (at least) one element of arg has a match, all un-abbreviated versions of matches are returned. When none.ok is TRUE and arg has zero elements, character(0) is returned.

See Also

Other package helpers: Dictionary, Storage, check_missing(), find_namespace_calls(), identical_structure(), input_check_response(), package_logo(), print_data.frame(), print_matrix(), system_information(), unexpected_error(), user_confirm()

match_numerics *Best-possible match of two numeric vectors*

Description

This function matches the indices of two numeric vectors as good as possible (that means with the smallest possible sum of deviations).

Usage

```
match_numerics(x, y)
```

Arguments

x, y	[numeric()] Two vectors of the same length.

Value

An integer vector of length length(x) with the positions of y in x.

See Also

Other vector helpers: check_numeric_vector(), check_probability_vector(), chunk_vector(), equidistant_vectors(), insert_vector_entry(), map_indices(), permutations(), split_vector_at(), subsets(), vector_occurrence()

Examples

```
x <- c(-1, 0, 1)
y <- c(0.1, 1.5, -1.2)
match_numerics(x, y)
```

matrix_diagonal_indices *Get indices of matrix diagonal*

Description

This function returns the indices of the diagonal elements of a quadratic matrix.

Usage

```
matrix_diagonal_indices(n, triangular = NULL)
```

Arguments

n	[integer(1)] The matrix dimension.
triangular	[NULL or character(1)] If NULL (default), all elements of the matrix are considered. If "lower" ("upper"), only the lower- (upper-) triangular matrix is considered.

Value

An integer vector.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), check_transition_probability_matrix(), cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_indices(), sample_correlation_matrix(), sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
# indices of diagonal elements
n <- 3
matrix(1:n^2, n, n)
matrix_diagonal_indices(n)

# indices of diagonal elements of lower-triangular matrix
L <- matrix(0, n, n)
L[lower.tri(L, diag=TRUE)] <- 1:((n * (n + 1)) / 2)
L
matrix_diagonal_indices(n, triangular = "lower")

# indices of diagonal elements of upper-triangular matrix
U <- matrix(0, n, n)
U[upper.tri(U, diag=TRUE)] <- 1:((n * (n + 1)) / 2)
U
matrix_diagonal_indices(n, triangular = "upper")
```

---

matrix_indices *Get* matrix *indices*

Description

This function returns matrix indices as character.

Usage

```
matrix_indices(x, prefix = "", exclude_diagonal = FALSE)
```

Arguments

x	[matrix] A matrix.
prefix	[character(1)] A prefix for the indices.
exclude_diagonal	[logical(1)] Exclude indices where row equals column?

Value

A character vector.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), check_transition_probability_
cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), sample_correlation_matrix(
sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
M <- diag(3)
matrix_indices(M)
matrix_indices(M, "M_")
matrix_indices(M, "M_", TRUE)
```

---

merge_lists *Merge named lists*

Description

This function merges lists based on their element names. Elements are only included in the final output list, if no former list has contributed an element with the same name.

Usage

```
merge_lists(...)
```

Arguments

...	One or more named list(s).

Value

A list.

See Also

Other list helpers: check_list_of_lists()

Examples

```
merge_lists(list("a" = 1, "b" = 2), list("b" = 3, "c" = 4, "d" = NULL))
```

---

occurrence_info *Provide information about occurrences*

Description

This function provides verbose information about absolute or relative element occurrences in data.frame columns.

Usage

```
occurrence_info(x, relative = FALSE, named = FALSE)
```

Arguments

x	[data.frame] The object to check for occurrences.
relative	[logical(1)] The number of rows or columns to be printed, greater or equal 2.
named	[logical(1)] Prepend column names of x (if not NA)?

Value

A character().

See Also

Other data.frame helpers: delete_columns_data.frame(), group_data.frame(), round_data.frame()

Examples

```
occurrence_info(datasets::warpbreaks, relative = FALSE, named = TRUE)
```

package_logo *Creating a basic logo for an R package*

Description

This function creates a basic R package logo. The logo has a white background and the package name (with or without curly brackets) in the center. The font size for the package name is scaled such that it fits inside the logo. Type `?oeli` to see an example.

Usage

```
package_logo(
  package_name,
  brackets = FALSE,
  background = ggplot2::ggplot() + ggplot2::theme_void(),
  s_x = 1,
  s_y = 1,
  s_width = 1,
  s_height = 1,
  white_around_sticker = FALSE
)
```

Arguments

`package_name`	`[character(1)]` The package name.
`brackets`	`[logical(1)]` Curly brackets around the package name?
`background`	A `ggplot` object, the background of the sticker.
`s_x`, `s_y`, `s_width`, `s_height`, `white_around_sticker`	Passed on to `sticker`.

Value

A `ggplot` object.

References

- This function builds upon `sticker`.
- Use `use_logo` to set up the logo for a package.

See Also

Other package helpers: `Dictionary`, `Storage`, `check_missing()`, `find_namespace_calls()`, `identical_structure()`, `input_check_response()`, `match_arg()`, `print_data.frame()`, `print_matrix()`, `system_information()`, `unexpected_error()`, `user_confirm()`

Examples

```
print(package_logo("my_package", brackets = TRUE))
```

`permutations` *Build permutations*

Description

This function creates all permutations of a given `vector`.

Usage

```
permutations(x)
```

Arguments

`x`	[`atomic()`] Any `vector`.

Value

A `list` of all permutations of `x`.

References

Modified version of https://stackoverflow.com/a/20199902/15157768.

See Also

Other vector helpers: `check_numeric_vector()`, `check_probability_vector()`, `chunk_vector()`, `equidistant_vectors()`, `insert_vector_entry()`, `map_indices()`, `match_numerics()`, `split_vector_at()`, `subsets()`, `vector_occurrence()`

Examples

```
permutations(1:3)
permutations(LETTERS[1:3])
```

`print_data.frame` *Print (abbreviated)* `data.frame`

Description

This function prints a (possibly abbreviated) `data.frame`.

Usage

```
print_data.frame(
  x,
  rows = NULL,
  cols = NULL,
  digits = NULL,
  row.names = TRUE,
  col.names = TRUE
)
```

Arguments

`x`	[`data.frame`] A `data.frame`.
`rows, cols`	[`integer(1)`	`NULL`] The number of rows or columns to be printed, greater or equal `2`. Printing is abbreviated in the middle. Can be `NULL` to print everything.
`digits`	[`integer(1)`	`NULL`] The number of decimal places to be used. Negative values are allowed, resulting in rounding to a power of ten. Can be `NULL` to not round.
`row.names, col.names`	[`logical(1)`] Print row names or column names?

Value

Invisibly returns `x`.

See Also

Other package helpers: `Dictionary`, `Storage`, `check_missing()`, `find_namespace_calls()`, `identical_structure()`, `input_check_response()`, `match_arg()`, `package_logo()`, `print_matrix()`, `system_information()`, `unexpected_error()`, `user_confirm()`

Examples

```
x <- data.frame(1:10, LETTERS[1:10], stats::rnorm(10))
print_data.frame(x, rows = 7)
print_data.frame(x, rows = 7, cols = 2)
print_data.frame(x, rows = 7, cols = 2, digits = 1)
print_data.frame(x, rows = 7, cols = 2, digits = 1, row.names = FALSE)
print_data.frame(x, rows = 7, cols = 2, digits = 1, col.names = FALSE)
```

---

print_matrix *Print (abbreviated)* matrix

Description

This function prints a (possibly abbreviated) matrix.

Usage

```
print_matrix(
  x,
  rowdots = 4,
  coldots = 4,
  digits = 2,
  label = NULL,
  simplify = FALSE,
  details = !simplify
)
```

Arguments

x	[atomic() | matrix] The object to be printed.
rowdots	[integer(1)] The row number which is replaced by ... .
coldots	[integer(1)] The column number which is replaced by ... .
digits	[integer(1)] The number of printed decimal places if input x is numeric.
label	[character(1)] A label for x. Only printed if simplify = FALSE.
simplify	[logical(1)] Simplify the output?
details	[logical(1)] Print the type and dimension of x?

Value

Invisibly returns `x`.

References

This function is a modified version of `pprint()` from the `{ramify}` package.

See Also

Other package helpers: `Dictionary`, `Storage`, `check_missing()`, `find_namespace_calls()`, `identical_structure()`, `input_check_response()`, `match_arg()`, `package_logo()`, `print_data.frame()`, `system_information()`, `unexpected_error()`, `user_confirm()`

Examples

```
print_matrix(x = 1, label = "single numeric")
print_matrix(x = LETTERS[1:26], label = "letters")
print_matrix(x = 1:3, coldots = 2)
print_matrix(x = matrix(rnorm(99), ncol = 1), label = "single column matrix")
print_matrix(x = matrix(1:100, nrow = 1), label = "single row matrix")
print_matrix(x = matrix(LETTERS[1:24], ncol = 6), label = "big matrix")
print_matrix(x = diag(5), coldots = 2, rowdots = 2, simplify = TRUE)
```

---

quiet *Silence R code*

Description

This function silences warnings, messages and any `cat()` or `print()` output from R expressions or functions.

Usage

```
quiet(x, print_cat = TRUE, message = TRUE, warning = TRUE)
```

Arguments

`x`	[expression] Any function or expression or value assignment expression.
`print_cat`	[logical(1)] Silence `print()` and `cat()` outputs?
`message`	[logical(1)] Silence messages?
`warning`	[logical(1)] Silence warnings?

Value

Invisibly the expression x.

References

This function is a modified version of quiet.

See Also

Other function helpers: do.call_timed(), function_arguments(), function_body(), function_defaults(), timed(), try_silent(), variable_name()

Examples

```
f <- function() {
  warning("warning")
  message("message")
  cat("cat")
  print("print")
}
quiet(f())
```

---

round_data.frame *Round* numeric *columns of a* data.frame

Description

This function rounds (only) the numeric columns of a data.frame.

Usage

```
round_data.frame(df, digits = 0)
```

Arguments

df	[data.frame] A data.frame.
digits	[integer(1) | NULL] The number of decimal places to be used. Negative values are allowed, resulting in rounding to a power of ten. Can be NULL to not round.

Value

A data.frame.

See Also

Other data.frame helpers: delete_columns_data.frame(), group_data.frame(), occurrence_info()

Examples

```
df <- data.frame("label" = c("A", "B"), "number" = rnorm(10))
round_data.frame(df, digits = 1)
```

---

sample_correlation_matrix

Sample correlation matrix

Description

This function samples a correlation matrix by sampling a covariance matrix from an inverse Wishart distribution and transforming it to a correlation matrix.

Usage

```
sample_correlation_matrix(dim, df = dim, scale = diag(dim))
```

Arguments

dim	[integer(1)] The dimension.
df	[integer(1)] The degrees of freedom of the inverse Wishart distribution greater or equal dim.
scale	[matrix()] The scale covariance matrix of the inverse Wishart distribution of dimension dim.

Value

A correlation matrix.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), check_transition_probability_
cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(),
sample_covariance_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
sample_correlation_matrix(dim = 3)
```

sample_covariance_matrix *Sample covariance matrix*

Description

This function samples a covariance matrix from an inverse Wishart distribution.

Usage

```
sample_covariance_matrix(dim, df = dim, scale = diag(dim), diag = FALSE)
```

Arguments

dim	[integer(1)] The dimension.
df	[integer(1)] The degrees of freedom of the inverse Wishart distribution greater or equal dim.
scale	[matrix()] The scale covariance matrix of the inverse Wishart distribution of dimension dim.
diag	[logical(1)] Diagonal matrix?

Value

A covariance matrix.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), check_transition_probability_
cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(),
sample_correlation_matrix(), sample_transition_probability_matrix(), stationary_distribution()

Examples

```
sample_covariance_matrix(dim = 3)
```

sample_transition_probability_matrix *Sample transition probability matrices*

Description

This function returns a random, squared matrix of dimension `dim` that fulfills the properties of a transition probability matrix.

Usage

```
sample_transition_probability_matrix(dim, state_persistent = TRUE)
```

Arguments

`dim`	[integer(1)] The dimension.
`state_persistent`	[logical(1)] Put more probability on the diagonal?

Value

A transition probability `matrix`.

See Also

Other matrix helpers: `check_correlation_matrix()`, `check_covariance_matrix()`, `check_transition_probability_cov_to_chol()`, `diff_cov()`, `insert_matrix_column()`, `matrix_diagonal_indices()`, `matrix_indices()`, `sample_correlation_matrix()`, `sample_covariance_matrix()`, `stationary_distribution()`

Examples

```
sample_transition_probability_matrix(dim = 3)
```

simulate_markov_chain *Simulate Markov chain*

Description

This function simulates a Markov chain.

Usage

```
simulate_markov_chain(Gamma, T, delta = oeli::stationary_distribution(Gamma))
```

Arguments

Gamma	[matrix()] A transition probability matrix.
T	[integer(1)] The length of the Markov chain.
delta	[numeric()] A probability vector, the initial distribution. The stationary distribution is used by default.

Value

A numeric vector of length T with states.

See Also

Other simulation helpers: Simulator, correlated_regressors(), ddirichlet_cpp(), dmixnorm_cpp(), dmvnorm_cpp(), dtnorm_cpp(), dwishart_cpp(), gaussian_tv()

Examples

```
Gamma <- matrix(c(0.8, 0.2, 0.3, 0.7), byrow = TRUE, nrow = 2)
delta <- c(0.6, 0.4)
simulate_markov_chain(Gamma = Gamma, T = 20, delta = delta)
```

Simulator *Simulator R6 Object*

Description

Creates a simulation setup, where a function f is evaluated runs times, optionally at each combination of input values.

Provides some convenience (see below for more details):

- Simulation results can be restored from a backup if the R session crashes.
- More simulation runs can be conducted after the initial simulation, failed simulation cases can be re-run.
- Parallel computation and progress updates are supported.

Details

Backup:

Simulation results can be saved to disk, allowing you to restore the results if the R session is interrupted or crashes before the simulation completes. To enable backup, set backup = TRUE in the $go() method, which will create a backup directory at the location specified by path. To restore, use Simulator$initialize(use_backup = path).

More runs and re-run:

If additional simulation runs are needed, simply call the `$go()` method again. Any cases that were not successfully completed in previous runs will be attempted again.

Parallel computation:

By default, simulations run sequentially. But since they are independent, they can be parallelized to decrease computation time. To enable parallel computation, use the {future} framework. For example, run

```
future::plan(future::multisession, workers = 4)
```

in advance for computation in 4 parallel R sessions.

Progress updates:

Use the {progressr} framework to get progress updates. For example, run the following in advance:

```
progressr::handlers(global = TRUE)
progressr::handlers(
  progressr::handler_progress(format = ">> :percent, :eta to go :message")
)
```

Active bindings

`results` [tibble, read-only]
: The simulation results.

`cases` [tibble, read-only]
: The simulation cases.

Methods

Public methods:

- `Simulator$new()`
- `Simulator$print()`
- `Simulator$define()`
- `Simulator$go()`

Method `new()`**:** Initialize a `Simulator` object, either a new one or from backup.

Usage:

```
Simulator$new(
  use_backup = NULL,
  verbose = getOption("verbose", default = FALSE)
)
```

Arguments:

`use_backup` [`NULL` | `character(1)`]
: Optionally a path to a backup folder previously used in `$go()`.

`verbose` [`logical(1)`]
: Provide info? Does not include progress updates. For that, see details.

Method `print()`**:** Print method.

Usage:

```
Simulator$print()
```

Method `define()`**:** Define function and arguments for a new `Simulator` object.

Usage:

```
Simulator$define(f, ...)
```

Arguments:

`f` [`function`]
: A `function` to evaluate.

`...` Arguments for `f`. Each value must be

1. named after an argument of `f`, and
2. a `list`, where each element is a variant of that argument for `f`.

Method `go()`**:** Run simulations.

Usage:

```
Simulator$go(
  runs = 0,
  backup = FALSE,
  path = paste0("backup_", format(Sys.time(), "%Y-%m-%d-%H-%M-%S"))
)
```

Arguments:

`runs` [`integer(1)`]
: The number of (additional) simulation runs.
If `runs = 0`, only pending cases (if any) are solved.

`backup` [`logical(1)`]
: Create a backup under `path`?

`path` [`character(1)`]
: Only relevant, if `backup = TRUE`.
In this case, a path for a new folder, which does not yet exist and allows reading and writing.

See Also

Other simulation helpers: `correlated_regressors()`, `ddirichlet_cpp()`, `dmixnorm_cpp()`, `dmvnorm_cpp()`, `dtnorm_cpp()`, `dwishart_cpp()`, `gaussian_tv()`, `simulate_markov_chain()`

Examples

```
# 1. Initialize a new simulation setup:
object <- Simulator$new(verbose = TRUE)

# 2. Define function `f` and arguments (if any):
f <- function(x, y = 1) {
  Sys.sleep(runif(1)) # to see progress updates
  x + y
}
x_args <- list(1, 2)
```

```
object$define(f = f, x = x_args)
print(object)

# 3. Define 'future' and 'progress' (optional):
## Not run:
future::plan(future::sequential)
progressr::handlers(global = TRUE)
## End(Not run)

# 4. Evaluate `f` `runs` times at each parameter combination (backup is optional):
path <- file.path(tempdir(), paste0("backup_", format(Sys.time(), "%Y-%m-%d-%H-%M-%S")))
object$go(runs = 2, backup = TRUE, path = path)

# 5. Access the results:
object$results

# 6. Check if cases are pending or if an error occurred:
object$cases

# 7. Restore simulation results from backup:
object_restored <- Simulator$new(use_backup = path)
print(object_restored)
## Not run: all.equal(object, object_restored)

# 8. Run more simulations and pending simulations (if any):
object_restored$go(runs = 2)
```

`split_vector_at` *Split a vector at positions*

Description

This function splits a vector at specific positions.

Usage

```
split_vector_at(x, at)
```

Arguments

`x`	[atomic()‘] A vector of elements.
`at`	[`integer()`] Index position(s) just before to split. For example, `at = n` splits before the nth element of `x`.

Value

A `list`.

References

Based on https://stackoverflow.com/a/19274414.

See Also

Other vector helpers: check_numeric_vector(), check_probability_vector(), chunk_vector(), equidistant_vectors(), insert_vector_entry(), map_indices(), match_numerics(), permutations(), subsets(), vector_occurrence()

Examples

```
x <- 1:10
split_vector_at(x, c(2, 3, 5, 7))
```

---

stationary_distribution

Stationary distribution

Description

This function computes the stationary distribution corresponding to a transition probability matrix.

Usage

```
stationary_distribution(tpm, soft_fail = FALSE)
```

Arguments

tpm	[matrix()] A transition probability matrix.
soft_fail	[logical(1)] Return the discrete uniform distribution if the computation of the stationary distribution fails for some reason? Else, throw an error.

Value

A numeric vector.

See Also

Other matrix helpers: check_correlation_matrix(), check_covariance_matrix(), check_transition_probability_
cov_to_chol(), diff_cov(), insert_matrix_column(), matrix_diagonal_indices(), matrix_indices(), sample_correlation_matrix(), sample_covariance_matrix(), sample_transition_probability_matrix()

Examples

```
tpm <- matrix(0.05, nrow = 3, ncol = 3)
diag(tpm) <- 0.9
stationary_distribution(tpm)
```

`Storage` *Storage R6 Object*

Description

Provides a simple indexing interface for list elements based on R6. Basically, it allows to store items in a list and to regain them based on identifiers defined by the user.

Value

The output depends on the method:

- `$new()` returns a `Storage` object.
- `$add()`, `$remove()`, and `$print()` invisibly return the `Storage` object (to allow for method chaining)
- `$get()` returns the requested element(s)
- `$number()` returns an `integer`
- `$indices()` return an `integer vector`

Setting identifiers

An identifier is a `character`, typically a binary property. Identifiers can be negated by placing an exclamation mark (`"!"`) in front of them. Identifiers that have been assigned to other elements previously do not need to be specified again for new elements; instead, a default value can be used. This default value can be defined either globally for all cases (via the `$missing_identifier` field) or separately for each specific case (via the method argument).

User confirmation

If desired, the user can be asked for confirmation when adding, extracting, or removing elements using identifiers. This behavior can be set globally through the `$confirm` field or customized separately for each specific case via the method argument.

Active bindings

`identifier [character()]`
: The identifiers used.

`confirm [logical(1)]`
: The default value for confirmations.

`missing_identifier [logical(1)]`
: The default value for not specified identifiers.

`hide_warnings [logical(1)]`
: Hide warnings (for example if unknown identifiers are selected)?

Methods

Public methods:

- Storage$new()
- Storage$add()
- Storage$get()
- Storage$remove()
- Storage$number()
- Storage$indices()
- Storage$print()

Method `new()`**:** Initializing a `Storage` object.

Usage:

```
Storage$new()
```

Method `add()`**:** Adding an element.

Usage:

```
Storage$add(
  x,
  identifier,
  confirm = interactive() & self$confirm,
  missing_identifier = self$missing_identifier
)
```

Arguments:

`x` [any]
: An object to be saved.

`identifier` [character()]
: Pne or more identifiers (the identifier `"all"` is reserved to select all elements).

`confirm` [logical(1)]
: Prompted for confirmation?

`missing_identifier` [logical(1) | NA]
: The value for not specified identifiers.

Method `get()`**:** Getting elements.

Usage:

```
Storage$get(
  identifier = character(),
  ids = integer(),
  logical = "and",
  confirm = interactive() & self$confirm,
  missing_identifier = self$missing_identifier,
  id_names = FALSE
)
```

Arguments:

`identifier` [character()]
: Pne or more identifiers (the identifier `"all"` is reserved to select all elements).

`ids` [`integer()`]
: One or more ids.

`logical` [`character(1)`]
: In the case that multiple identifiers are selected, how should they be combined? Options are:
  - `"and"` (the default): the identifiers are combined with logical and (all identifiers must be `TRUE`)
  - `"or"`: the identifiers are combined with logical or (at least one identifier must be `TRUE`)

`confirm` [`logical(1)`]
: Prompted for confirmation?

`missing_identifier` [`logical(1)` | `NA`]
: The value for not specified identifiers.

`id_names` [`logical(1)`]
: Name the elements according to their ids?

Method `remove()`**:** removing elements

Usage:

```
Storage$remove(
  identifier = character(),
  ids = integer(),
  logical = "and",
  confirm = interactive() & self$confirm,
  missing_identifier = self$missing_identifier,
  shift_ids = TRUE
)
```

Arguments:

`identifier` [`character()`]
: Pne or more identifiers (the identifier `"all"` is reserved to select all elements).

`ids` [`integer()`]
: One or more ids.

`logical` [`character(1)`]
: In the case that multiple identifiers are selected, how should they be combined? Options are:
  - `"and"` (the default): the identifiers are combined with logical and (all identifiers must be `TRUE`)
  - `"or"`: the identifiers are combined with logical or (at least one identifier must be `TRUE`)

`confirm` [`logical(1)`]
: Prompted for confirmation?

`missing_identifier` [`logical(1)` | `NA`]
: The value for not specified identifiers.

`shift_ids` [`logical(1)`]
: Shift ids when in-between elements are removed?

Method `number()`**:** Computing the number of identified elements.

Usage:

```
Storage$number(
  identifier = "all",
  missing_identifier = self$missing_identifier,
  logical = "and",
  confirm = FALSE
)
```

Arguments:

`identifier [character()]`
: Pne or more identifiers (the identifier `"all"` is reserved to select all elements).

`missing_identifier [logical(1) | NA]`
: The value for not specified identifiers.

`logical [character(1)]`
: In the case that multiple identifiers are selected, how should they be combined? Options are:
  - `"and"` (the default): the identifiers are combined with logical and (all identifiers must be `TRUE`)
  - `"or"`: the identifiers are combined with logical or (at least one identifier must be `TRUE`)

`confirm [logical(1)]`
: Prompted for confirmation?

Method `indices()`**:** Returning indices based on defined identifiers.

Usage:

```
Storage$indices(
  identifier = "all",
  logical = "and",
  confirm = interactive() & self$confirm
)
```

Arguments:

`identifier [character()]`
: Pne or more identifiers (the identifier `"all"` is reserved to select all elements).

`logical [character(1)]`
: In the case that multiple identifiers are selected, how should they be combined? Options are:
  - `"and"` (the default): the identifiers are combined with logical and (all identifiers must be `TRUE`)
  - `"or"`: the identifiers are combined with logical or (at least one identifier must be `TRUE`)

`confirm [logical(1)]`
: Prompted for confirmation?

Method `print()`**:** Printing details of the saved elements.

Usage:

```
Storage$print(...)
```

Arguments:

`...` Currently not used.

See Also

Other package helpers: Dictionary, check_missing(), find_namespace_calls(), identical_structure(), input_check_response(), match_arg(), package_logo(), print_data.frame(), print_matrix(), system_information(), unexpected_error(), user_confirm()

Examples

```
### 1. Create a `Storage` object:
my_storage <- Storage$new()

# 2. Add elements along with identifiers:
my_storage$
  add(42, c("number", "rational"))$
  add(pi, c("number", "!rational"))$
  add("fear of black cats", c("text", "!rational"))$
  add("wearing a seat belt", c("text", "rational"))$
  add(mean, "function")

# 3. What elements are stored?
print(my_storage)

# 4. Extract elements based on identifiers:
my_storage$get("rational")
my_storage$get("!rational")
my_storage$get(c("text", "!rational"))
my_storage$get("all") # get all elements
my_storage$get(c("text", "!text"))
my_storage$get(c("text", "!text"), logical = "or")

# 5. Extract elements based on ids:
my_storage$get(ids = 4:5)
my_storage$get(ids = 4:5, id_names = TRUE) # add the ids as names
```

---

subsets *Generate vector subsets*

Description

This function generates subsets of a vector.

Usage

```
subsets(v, n = seq_along(v))
```

Arguments

v	[atomic()‘] A vector of elements.
n	[integer(1)‘] The requested subset sizes.

Value

A `list`, each element is a subset of `v`.

See Also

Other vector helpers: `check_numeric_vector()`, `check_probability_vector()`, `chunk_vector()`, `equidistant_vectors()`, `insert_vector_entry()`, `map_indices()`, `match_numerics()`, `permutations()`, `split_vector_at()`, `vector_occurrence()`

Examples

```
v <- 1:3
subsets(v)
subsets(v, c(1, 3)) # only subsets of length 1 or 3
subsets(integer())  # trivial case works
```

`system_information` *General system level information*

Description

This function returns a `list` of general system level information.

Usage

```
system_information()
```

Value

A `list` with elements:

- `maschine`, the model name of the device
- `cores`, the number of cores
- `ram`, the size of the RAM
- `os`, the operating system
- `rversion`, the R version used

See Also

Other package helpers: `Dictionary`, `Storage`, `check_missing()`, `find_namespace_calls()`, `identical_structure()`, `input_check_response()`, `match_arg()`, `package_logo()`, `print_data.frame()`, `print_matrix()`, `unexpected_error()`, `user_confirm()`

Examples

```
system_information()
```

`timed` *Interrupt long evaluations*

Description

This function interrupts an evaluation after a certain number of seconds. Note the limitations documented in `setTimeLimit`.

Usage

```
timed(expression, seconds = Inf, on_time_out = "silent")
```

Arguments

`expression`	[`expression`] An R expression to be evaluated.
`seconds`	[`numeric(1)`] The number of seconds.
`on_time_out`	[`character(1)`] Defines what action to take if the evaluation time exceeded, either: • `"error"` to throw an error exception • `"warning"` to return `NULL` along with a warning • `"silent"` (the default) to just return `NULL`

Value

The value of `expression` or, if the evaluation time exceeded, whatever is specified for `on_time_out`.

See Also

Other function helpers: `do.call_timed()`, `function_arguments()`, `function_body()`, `function_defaults()`, `quiet()`, `try_silent()`, `variable_name()`

Examples

```
foo <- function(x) {
  for (i in 1:10) Sys.sleep(x / 10)
  return(x)
}
timed(foo(0.5), 1)
timed(foo(1.5), 1)
```

`try_silent` *Try an expression silently*

Description

This function tries to execute `expr` and returns a string with the error message if the execution failed.

Usage

```
try_silent(expr)
```

Arguments

`expr`	[`expression`] An R expression to be evaluated.

Details

This function is a wrapper for `try`.

Value

Either the value of `expr` or in case of a failure an object of class `fail`, which contains the error message.

See Also

Other function helpers: `do.call_timed()`, `function_arguments()`, `function_body()`, `function_defaults()`, `quiet()`, `timed()`, `variable_name()`

Examples

```
## Not run:
try_silent(1 + 1)
try_silent(1 + "1")

## End(Not run)
```

unexpected_error *Handling of an unexpected error*

Description

This function reacts to an unexpected error by throwing an error and linking to an issue site with the request to submit an issue.

Usage

```
unexpected_error(
  msg = "Ups, an unexpected error occured.",
  issue_link = "https://github.com/loelschlaeger/oeli/issues"
)
```

Arguments

msg	[character(1)] An error message.
issue_link	[character(1)] The URL to an issues site.

Value

No return value, but it throws an error.

See Also

Other package helpers: Dictionary, Storage, check_missing(), find_namespace_calls(), identical_structure(), input_check_response(), match_arg(), package_logo(), print_data.frame(), print_matrix(), system_information(), user_confirm()

user_confirm *User confirmation*

Description

This function asks in an interactive question a binary question.

Usage

```
user_confirm(question = "Question?", default = FALSE)
```

Arguments

question	[character(1)] The binary question to ask. It should end with a question mark.
default	[logical(1)] The default decision.

Value

Either TRUE or FALSE.

See Also

Other package helpers: Dictionary, Storage, check_missing(), find_namespace_calls(), identical_structure(), input_check_response(), match_arg(), package_logo(), print_data.frame(), print_matrix(), system_information(), unexpected_error()

---

variable_name *Determine variable name*

Description

This function tries to determine the name of a variable passed to a function.

Usage

```
variable_name(variable, fallback = "unnamed")
```

Arguments

variable	[any] Any object.
fallback	[character(1)] A fallback name if for some reason the actual variable name (which must be a single character) cannot be determined.

Value

A character, the variable name.

See Also

Other function helpers: do.call_timed(), function_arguments(), function_body(), function_defaults(), quiet(), timed(), try_silent()

Examples

```
variable_name(a)
f <- function(x) variable_name(x)
f(x = a)
```

vector_occurrence *Find the positions of first or last occurrence of unique vector elements*

Description

This function finds the positions of first or last occurrence of unique vector elements.

Usage

```
vector_occurrence(x, type = "first")
```

Arguments

x	[atomic()] A vector.
type	[character(1)] Either "first" for the first or "last" for the last occurrence.

Value

An integer vector, the positions of the unique vector elements. The ordering corresponds to unique(x), i.e., the i-th element in the output is the (first or last) occurrence of the i-th element from unique(x).

See Also

Other vector helpers: check_numeric_vector(), check_probability_vector(), chunk_vector(), equidistant_vectors(), insert_vector_entry(), map_indices(), match_numerics(), permutations(), split_vector_at(), subsets()

Examples

```
x <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
unique(x)
vector_occurrence(x, "first")
vector_occurrence(x, "last")
```

Index