Package 'oeli'

May 22, 2024

Type Package

Title My Utilities for Developing Data Science Software

Version 0.5.0

Description Some general helper functions that I and maybe others find useful when developing data science software. Functionality includes argument validation, density calculation, sampling, matrix printing, user interaction, storage helpers and more. The vignettes illustrate use cases.

License GPL (>= 3)

Encoding UTF-8

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

Config/testthat/edition 3

RoxygenNote 7.3.1

Imports benchmarkme, checkmate, cli, ggplot2, hexSticker, latex2exp, Rcpp, rprojroot, showtext, stats, sysfonts, usethis, utils

LinkingTo Rcpp, RcppArmadillo, testthat

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

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

VignetteBuilder knitr

NeedsCompilation yes

Author Lennart Oelschläger [aut, cre]

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

Repository CRAN

Date/Publication 2024-05-22 10:30:08 UTC

R topics documented:

`check_correlation_matrix`
Check if an argument is a correlation matrix

Description

This function checks whether the input is a symmetric, real matrix that fulfills the correlation matrix properties.

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`	Object to check.
`dim`	An `integer`, the matrix dimension.
`tolerance`	A non-negative `numeric` 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

Compare to `check_matrix`.

check_covariance_matrix

Check if an argument is a covariance matrix

Description

This function checks whether the input is a symmetric, real matrix that fulfills the covariance matrix properties.

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	Object to check.
dim	An integer, the matrix dimension.
tolerance	A non-negative numeric 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

Compare to check_matrix.

check_date *Check date format*

Description

This function checks if the input `date` has the format `"YYYY-MM-DD"`.

Usage

```
check_date(date)
```

Arguments

`date`	A `character`, specifying a date in format `"YYYY-MM-DD"`.

Value

`as.Date(date)` if `date` has the format `"YYYY-MM-DD"`. Otherwise, the function throws an error.

Examples

```
check_date(date = "2000-01-01")
```

check_list_of_lists *Check if an argument is a list of lists*

Description

This function checks 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	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

Compare to check_list.

---

check_numeric_vector *Check if an argument is a numeric vector*

Description

This function checks 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	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

Compare to check_numeric.

---

check_probability_vector

Check if an argument is a probability vector

Description

This function checks whether the input is a real vector with non-negative entries that add up to one.

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	Object to check.
len	[integer(1)] Exact expected length of x.
tolerance	A non-negative numeric 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

Compare to check_numeric.

---

check_transition_probability_matrix

Check if an argument is a transition probability matrix

Description

This function checks whether the input is a quadratic, real matrix with elements between 0 and 1 and row sums equal to 1.

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	Object to check.
dim	An integer, the matrix dimension.
tolerance	A non-negative numeric 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

Compare to check_matrix.

---

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	A vector.
n	An integer smaller or equal length(x).
type	Either 1 (default) to split x into n chunks of equal size or 2 to split x into chunks of size n.
strict	Set to TRUE to fail if length(x) is not a multiple of n, or FALSE (default), else.

Value

A `list`.

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))
```

`cov_2_chol` *Get Cholesky root elements and build 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_2_chol(cov, unique = TRUE)

chol_2_cov(chol)

unique_chol(chol)
```

Arguments

`cov`	A covariance `matrix` of dimension `dim`.
`unique`	Set to `TRUE` to ensure that the Cholesky decomposition is unique by restricting the diagonal elements to be positive.
`chol`	A `numeric vector` of Cholesky root elements.

Value

For `cov_2_chol` a `numeric vector` of Cholesky root elements. For `chol_2_cov` a covariance `matrix`.

Examples

```
cov <- sample_covariance_matrix(4)
chol <- cov_2_chol(cov)
all.equal(cov, chol_2_cov(chol))
```

ddirichlet *Compute density of Dirichlet distribution*

Description

This function computes the density of a Dirichlet distribution.

Usage

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

Arguments

x	A numeric, a weight vector of length p. Each vector element must be between 0 and 1. The sum of the vector elements must be 1.
concentration	A numeric, the concentration vector of length p.
log	A logical, if TRUE the logarithm of the density value is returned. By default, log = FALSE.

Value

A numeric, the density value.

Examples

```
x <- c(0.5, 0.3, 0.2)
concentration <- 1:3
ddirichlet(x = x, concentration = concentration)
ddirichlet(x = x, concentration = concentration, log = TRUE)
```

delete_data_frame_columns *Deleting* data.frame *columns*

Description

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

Usage

```
delete_data_frame_columns(df, column_names)
```

Arguments

df	A data.frame.
column_names	A character (vector), the name(s) of a column of df to group by.

Value

The input `df` without the columns defined by `column_names`.

Examples

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

`Dictionary` *Dictionary R6 Object*

Description

Provides a simple key-value interface based on R6.

Active bindings

`keys` A `character vector` of available keys.

`alias` A `list` of available keys per alias value.

Methods

Public methods:

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

Method `new()`**:** initializing a `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` A single `character`, the name for the key variable.

`alias_name` Optionally a single `character`, the name for the alias variable. Can also be `NULL` (default) for no alias.

`value_names` A `character` (`vector`), the names of the values connected to a key.

`value_assert` A `alist` with check functions for supplied values. For each element in `value_names`, `values_assert` *can* have an identically named element of the form `checkmate::assert_*(...)`, where `...` can be any arguments for the assertion function except for the `x` argument.

`allow_overwrite` Either `TRUE` (default) to allow overwriting existing keys with new values, or `FALSE` else. Duplicate keys are never allowed.

`keys_reserved` A `character` (`vector`) of names that must not be used as keys.

`alias_choices` Optionally a `character` `vector` of possible values for the alias. Can also be `NULL`, then all alias values are allowed.

`dictionary_name` Optionally a single `character`, a name for the dictionary.

Returns: a new `Dictionary` object

Method `add()`**:** adding an element

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).

Returns: invisibly the `Dictionary` object

Method `get()`**:** getting elements

Usage:

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

Arguments:

`key` A single `character`, a value for the key variable `key_name`. Use the `$keys` method for available keys.

`value` Optionally a single `character`, 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`.

Returns: the selected value

Method `remove()`**:** removing elements (and associated alias, if any)

Usage:

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

Arguments:

`key` A single `character`, a value for the key variable `key_name`. Use the `$keys` method for available keys.

Returns: invisibly the `Dictionary` object

Method `print()`**:** printing details of the dictionary

Usage:

```
Dictionary$print()
```

Returns: invisibly the `Dictionary` object

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

Description

These functions difference and un-difference a covariance matrix with respect to row `ref`.

Usage

```
diff_cov(cov, ref = 1)

undiff_cov(cov_diff, ref = 1)

delta(ref = 1, dim)
```

Arguments

`cov, cov_diff`	A (differenced) covariance `matrix` of dimension `dim` (or `dim` - 1, respectively).
`ref`	An `integer` between `1` and `dim`, the reference row for differencing that maps `cov` to `cov_diff`, see details. By default, `ref = 1`.
`dim`	An `integer`, the dimension.

Details

For differencing: Let Σ be a covariance matrix of dimension n. Then

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

is the differenced covariance matrix with respect to row $k = 1, \ldots, n$, where Δ_k is a difference operator that depends on the reference row k. More precise, Δ_k the identity matrix of dimension n without row k and with -1s in column k. It can be computed with `delta(ref = k, dim = n)`.

For un-differencing: The "un-differenced" covariance matrix Σ cannot be uniquely computed from $\tilde{\Sigma}$. For a non-unique solution, we add a column and a row of zeros at column and row number k to $\tilde{\Sigma}$, respectively, and add 1 to each matrix entry to make the result a proper covariance matrix.

Value

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

Examples

```
n <- 3
Sigma <- sample_covariance_matrix(dim = n)
k <- 2

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

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

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

---

dmvnorm *Compute density of multivariate normal distribution*

Description

This function computes the density of a multivariate normal distribution.

Usage

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

Arguments

x	A numeric, a quantile vector of length p.
mean	A numeric, the mean vector of length p.
Sigma	A matrix, the covariance matrix of dimension p x p.
log	A logical, if TRUE the logarithm of the density value is returned. By default, log = FALSE.

Value

A numeric, the density value.

Examples

```
x <- c(0, 0)
mean <- c(0, 0)
Sigma <- diag(2)
dmvnorm(x = x, mean = mean, Sigma = Sigma)
dmvnorm(x = x, mean = mean, Sigma = Sigma, log = TRUE)
```

`do.call_timed` *Measure computation time*

Description

This function measures the computation time of a `do.call` call.

Usage

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

Arguments

`what`	Passed to `do.call`.
`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).

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)
```

dwishart *Compute density of (Inverse-) Wishart distribution*

Description

This function computes the density of the (Inverse-) Wishart distribution.

Usage

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

Arguments

x	A matrix, a covariance matrix of dimension p x p.
df	An integer, the degrees of freedom. Must be greater or equal p.
scale	A matrix, the scale matrix of dimension p x p. Must be a covariance matrix.
log	A logical, if TRUE the logarithm of the density value is returned. By default, log = FALSE.
inv	A logical, if TRUE the density of the Inverse-Wishart distribution is returned. By default, inv = FALSE.

Value

A numeric, the density value.

Examples

```
x <- diag(2)
df <- 4
scale <- diag(2)
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)
```

find_closest_year *Find the closest year to a given date*

Description

This function takes a date as input and returns the closest year.

Usage

```
find_closest_year(date)
```

Arguments

date	A date in the format of "YYYY-MM-DD".

Value

An integer, the closest year to the input date.

Examples

```
find_closest_year(as.Date("2022-07-15"))
find_closest_year(as.Date("2022-01-01"))
```

---

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	A function.
with_default	Either TRUE to include function arguments that have default values, or FALSE else.
with_ellipsis	Either TRUE to include the "..." argument if present, or FALSE else.

Value

A character vector.

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`	A `function`.
`braces`	Either `FALSE` (default) to remove `"{"` and `"}"` at start and end (if any), or `TRUE` if not.
`nchar`	An `integer`, the maximum number of characters before abbreviation. Must be at least `3`. By default, `nchar = getOption("width") - 4`.

Value

A `character`, the body of `f`.

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`	A `function`.
`exclude`	A `character` of argument names to exclude. Can be `NULL` (default) to not exclude any argument names.

Value

A named list.

Examples

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

---

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	A data.frame.
by	A character, the name of a column of df to group by.
keep_by	Either TRUE (default) to keep the grouping column by, or FALSE, else.

Value

A list of data.frames, subsets of df.

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)
```

`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`	A `matrix`.
`x`	A `vector` of length `nrow(A)`, the column to be added. Can also be a single value.
`p`	An `integer`, the position where to add the column: • `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`. Can also be a `vector` of multiple positions.

Value

A `matrix`.

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`	A `vector`.
`x`	A single value (i.e., an `atomic` `vector` of length 1), the entry to be added.
`p`	An `integer`, the position where to add the entry: • `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`. Can also be a `vector` of multiple positions.

Value

A `vector`.

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())
```

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`	A `character` (vector), the function argument.
`choices`	A `character` (vector) of allowed values for `arg`.
`several.ok`	Either `TRUE` if `arg` is allowed to have more than one element, or `FALSE` else.
`none.ok`	Either `TRUE` if `arg` is allowed to have zero elements, or `FALSE` else.

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.

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`	A `numeric` vector.
`y`	Another `numeric` vector of the same length as `x`.

Value

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

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`	An `integer`, the matrix dimension.
`triangular`	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`.

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	A matrix.
prefix	A character as prefix for the indices.
exclude_diagonal	Either TRUE to exclude indices where row equals column, or FALSE to include those.

Value

A character vector.

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.

Examples

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

---

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.

The function optionally calls use_logo if use_logo = TRUE to set up the logo for a package.

Usage

```
package_logo(package_name, brackets = TRUE, use_logo = FALSE)
```

Arguments

package_name	A character, the package name.
brackets	Set to TRUE (default) to have curly brackets around the package name.
use_logo	Set to TRUE to run use_logo in the end.

Value

A ggplot object.

Examples

```
package_logo("my_package", brackets = TRUE, use_logo = FALSE)
```

permutations *Build permutations*

Description

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

Usage

```
permutations(x)
```

Arguments

`x`	Any `vector`.

Value

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

References

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

Examples

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

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	A numeric or character (vector or matrix).
rowdots	An integer, the row number which is replaced by .... By default, rowdots = 4.
coldots	An integer, the column number which is replaced by .... By default, coldots = 4.
digits	An integer, the number of printed decimal places. Only relevant if input x is numeric. By default, digits = 2.
label	A character, a label for x. Only printed if simplify = FALSE. By default, label = NULL, i.e., no label.
simplify	A logical, set to TRUE to simplify the output. By default, simplify = FALSE.
details	A logical, set to TRUE to print the type and dimension of x. By default, details = !simplify.

Value

Invisibly returns x.

References

This function is a modified version of ramify::pprint().

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)
```

---

rdirichlet *Draw from Dirichlet distribution*

Description

This function draws from a Dirichlet distribution.

Usage

```
rdirichlet(n = 1, concentration)
```

Arguments

n	An integer, the number of samples.
concentration	The non-negative concentration vector of length p.

Value

If n = 1 a vector of length p, else a matrix of dimension n times p with samples as rows.

Examples

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

---

renv_development_packages

Using development packages when working with {renv}

Description

This function creates a file that loads development packages so that {renv} can detect and write them to the lockfile.

Usage

```
renv_development_packages(
  packages = c("covr", "devtools", "DT", "markdown", "R.utils", "yaml"),
  file_name = "development_packages"
)
```

Arguments

packages	A character vector of package names.
file_name	A single character, the name for the .R file.

Value

No return value, but it writes a file to the project root and adds an entry to the .Rbuildignore file.

rmvnorm *Draw from multivariate normal distribution*

Description

This function draws from a multivariate normal distribution.

Usage

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

Arguments

n	An integer, the number of samples.
mean	A numeric, the mean vector of length p.
Sigma	A matrix, the covariance matrix of dimension p x p.
log	A logical, if TRUE the draw is taken from the log-normal distribution. By default, log = FALSE.

Value

If n = 1 a vector of length p, else a matrix of dimension n times p with samples as rows.

Examples

```
mean <- c(0, 0)
Sigma <- diag(2)
rmvnorm(n = 3, mean = mean, Sigma = Sigma)
rmvnorm(mean = mean, Sigma = Sigma, log = TRUE)
```

rwishart *Draw from Wishart distribution*

Description

This function draws from a Wishart or Inverse-Wishart distribution.

Usage

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

Arguments

df	An integer, the degrees of freedom. Must be greater or equal p.
scale	A matrix, the scale matrix of dimension p x p. Must be a covariance matrix.
inv	A logical, if TRUE the density of the Inverse-Wishart distribution is returned. By default, inv = FALSE.

Value

A matrix, the random draw.

Examples

```
df <- 4
scale <- diag(2)
rwishart(df = df, scale = scale)
rwishart(df = df, scale = scale, inv = TRUE)
```

---

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	An integer, the dimension.
df	An integer greater or equal dim, the degrees of freedom of the inverse Wishart distribution.
scale	A covariance matrix of dimension dim, the scale matrix of the inverse Wishart distribution.
diag	Set to TRUE for a diagonal matrix.

Value

A covariance matrix.

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`	An `integer`, the matrix dimension.
`state_persistent`	Set to `TRUE` (default) to put more probability on the diagonal.

Value

A transition probability `matrix`.

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 = stationary_distribution(Gamma))
```

Arguments

`Gamma`	A transition probability `matrix`.
`T`	An `integer`, the length of the Markov chain.
`delta`	A `numeric` probability vector, the initial distribution. If not specified, `delta` is set to the stationary distribution of `Gamma`.

Value

A numeric vector of length T with states.

Examples

```
Gamma <- sample_transition_probability_matrix(dim = 3)
simulate_markov_chain(Gamma = Gamma, T = 10)
```

`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	A transition probability matrix.
soft_fail	Either TRUE to return the discrete uniform distribution if the computation of the stationary distribution fails for some reason, or FALSE to throw an error.

Value

A numeric vector.

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` a `character vector`, the identifiers used

`confirm` setting the default value for confirmations (either `TRUE` or `FALSE`)

`missing_identifier` setting the default value for not specified identifiers (either `TRUE`, `FALSE`, or `NA`)

`hide_warnings` either `TRUE` to hide warnings (for example if unknown identifiers are selected) or `FALSE` (default), else

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()
```

Returns: a new `Storage` object

Method `add()`**:** adding an element

Usage:

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

Arguments:

`x` any object to be saved

`identifier` a `character` `vector` with one or more identifiers (the identifier `"all"` is reserved to select all elements)

`confirm` either `TRUE` to be prompted for confirmation, or `FALSE` else

`missing_identifier` the `logical` value for not specified identifiers (either `NA`, `TRUE`, or `FALSE`)

Returns: invisibly the `Storage` object

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` a `character` `vector` with one or more identifiers (the identifier `"all"` is reserved to select all elements)

`ids` an `integer` `vector` of one or more ids

`logical` 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` either `TRUE` to be prompted for confirmation, or `FALSE` else

`missing_identifier` the `logical` value for not specified identifiers (either `NA`, `TRUE`, or `FALSE`)

`id_names` either `TRUE` to name the elements according to their ids or `FALSE` if not

Returns: the selected object(s)

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` a `character` `vector` with one or more identifiers (the identifier `"all"` is reserved to select all elements)

`ids` an `integer` `vector` of one or more ids

`logical` 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` either `TRUE` to be prompted for confirmation, or `FALSE` else

`missing_identifier` the `logical` value for not specified identifiers (either `NA`, `TRUE`, or `FALSE`)

`shift_ids` either `TRUE` to shift ids when in-between elements are removed, or `TRUE` to keep the ids

Returns: invisibly the `Storage` object

Method `number()`**:** computing the number of identified elements

Usage:

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

Arguments:

`identifier` a `character vector` with one or more identifiers (the identifier `"all"` is reserved to select all elements)

`missing_identifier` the `logical` value for not specified identifiers (either `NA`, `TRUE`, or `FALSE`)

`logical` 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` either `TRUE` to be prompted for confirmation, or `FALSE` else

Returns: an `integer`

Method `indices()`**:** returning indices based on defined identifiers

Usage:

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

Arguments:

`identifier` a `character vector` with one or more identifiers (the identifier `"all"` is reserved to select all elements)

`logical` 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` either `TRUE` to be prompted for confirmation, or `FALSE` else

Returns: an `integer vector`

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

Usage:

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

Arguments:

`...` currently not used

Returns: invisibly the `Storage` object

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`	A `vector` (`numeric` or `character`).
`n`	An `integer` vector with the required subset sizes. By default, `n = seq_along(v)`, i.e., all subsets of `v` are generated.

Value

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

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

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`	An R expression to be evaluated.
`seconds`	The number of seconds.
`on_time_out`	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`.

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`	An R expression to try.

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.

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 a GitHub issues site with the request to submit an issue.

Usage

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

Arguments

msg	A character, an error message.
issue_link	A character, the URL to a GitHub issues site.

Value

No return value, but it throws an error.

user_confirm *User confirmation*

Description

This function asks in an interactive question a binary question.

Usage

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

Arguments

question	A character, the binary question to ask. It should end with a question mark.
default	Either TRUE or FALSE (default), the default decision.

Value

Either TRUE or FALSE.

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 object.
fallback	A character, a fallback if for some reason the actual variable name (which must be a single character) cannot be determined.

Value

A character, the variable name.

Examples

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

Index