

Package: optimizeR (via r-universe)

October 27, 2024

Title Unified Framework for Numerical Optimizers

Version 1.1.1.9000

Description Provides a unified object-oriented framework for numerical optimizers in R. Allows for both minimization and maximization with any optimizer, optimization over more than one function argument, measuring of computation time, setting a time limit for long optimization tasks.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests knitr, ggplot2, pracma, rmarkdown, testthat

Config/testthat/edition 3

URL <https://loelschlaeger.de/optimizeR/>,
<https://github.com/loelschlaeger/optimizeR/>

BugReports <https://github.com/loelschlaeger/optimizeR/issues>

Depends R (>= 4.0.0)

Imports checkmate, cli, lbfgsb3c, oeli (>= 0.5.2), R6, stats,
TestFunctions, ucminf, utils

Repository <https://loelschlaeger.r-universe.dev>

RemoteUrl <https://github.com/loelschlaeger/optimizer>

RemoteRef HEAD

RemoteSha 3dff14e4f1c8b49d1b5b26f349da64fd747497b3

Contents

apply_optimizer	2
define_optimizer	3
Objective	5

Optimizer	8
optimizer_dictionary	14
ParameterSpaces	14
test_objective	17
test_optimizer	18

Index	20
--------------	-----------

apply_optimizer	<i>Apply optimizer object</i>
-----------------	-------------------------------

Description

This function performs numerical optimization using an optimizer object.

Usage

```
apply_optimizer(optimizer = optimizer_nlm(), objective, initial, ...)
```

Arguments

optimizer	An object of class optimizer.
objective	A function to be optimized, returning a single numeric. Its first argument must be a numeric vector of the same length as <code>initial</code> , followed by any other arguments specified by the <code>...</code> argument.
initial	A numeric vector with starting parameter values for the optimization.
...	Additional arguments to be passed to optimizer.

Value

A named list, containing at least these four elements:

`value` A numeric, the value of the estimated optimum of `objective`.

`parameter` A numeric vector, the parameter vector where the optimum of `objective` is obtained.

`seconds` A numeric, the total optimization time in seconds.

`initial` A numeric, the initial parameter values.

Appended are additional output elements of the optimizer (if not excluded by the `output_ignore` element via [define_optimizer](#)).

See Also

[define_optimizer\(\)](#) for creating an optimizer object.

Examples

```
apply_optimizer(optimizer_nlm(), function(x) x^4 + 3*x - 5, 2)
```

define_optimizer *Specify numerical optimizer*

Description

This function specifies the framework for a numerical optimizer.

Two wrappers for well-known optimizers are already available:

1. optimizer_nlm() for the `nlm` optimizer
2. optimizer_optim() for the `optim` optimizer

Usage

```
define_optimizer(
  .optimizer,
  .objective,
  .initial,
  .value,
  .parameter,
  .direction,
  ...,
  .output_ignore = character(0),
  .validate = FALSE,
  .validation_settings = list(objective_test = TestFunctions::TF_ackley, objective_add =
    list(), initial = round(stats::rnorm(2), 2), check_seconds = 10)
)

optimizer_nlm(
  ...,
  .output_ignore = character(0),
  .validate = FALSE,
  .validation_settings = list()
)

optimizer_optim(
  ...,
  .direction = "min",
  .output_ignore = character(0),
  .validate = FALSE,
  .validation_settings = list()
)
```

Arguments

- `.optimizer` A function, a numerical optimizer. Four conditions must be met:
1. It must have an input named `.objective` for a function, the objective function which is optimized over its first argument.

2. It must have an input named `.initial` for a numerical vector, the initial parameter vector.
3. It must have a `...` argument for additional parameters to the objective function.
4. The output must be a named list, including the optimal function value and the optimal parameter vector.

`.objective` A character, the name of the function input of optimizer.

`.initial` A character, the name of the starting parameter values input of optimizer.

`.value` A character, the name of the optimal function value in the output list of optimizer.

`.parameter` A character, the name of the optimal parameter vector in the output list of optimizer.

`.direction` A character, indicates whether the optimizer minimizes ("min") or maximizes ("max").

`...` Additional arguments to be passed to the optimizer. Without specifications, the default values of the optimizer are used.

`.output_ignore` A character vector of element names in the output of `.optimizer` that are not saved. The elements `.value` and `.parameter` are added automatically to `.output_ignore`, because they are saved separately, see the output documentation of [apply_optimizer](#).

`.validate` A logical, set to TRUE (FALSE) to (not) validate the optimizer object. By default, `.validate = FALSE`.

`.validation_settings`
Ignored if `.validate = FALSE`. Otherwise, a list of validation settings:

objective_test A function, the test function to be optimized. By default, it is the [Ackley function](#).

objective_add A list of additional arguments to `objective_test` (if any).
By default, `objective_add = list()`, because the default function for `objective_test` does not have additional arguments.

initial A numeric vector, the initial values for the optimization of `objective_test`.
By default, `initial = round(stats::rnorm(2), 2)`.

check_seconds An integer, the maximum number of seconds before the test is aborted. The test call is considered to be successful if no error occurred within `check_seconds` seconds. By default, `check_seconds = 10`.

Value

An optimizer object.

Format

An optimizer object is a list of six elements:

optimizer A function, the optimization algorithm.

optimizer_name A character, the name of optimizer.

optimizer_arguments A named list, where each element is an additional function argument for optimizer.

optimizer_direction Either "min" if the optimizer minimizes or "max" if the optimizer maximizes.

optimizer_labels A named list of four character:

objective the name of the function input of optimizer

initial the name of the starting parameter values input of optimizer

value the name of the optimal function value in the output list of optimizer

parameter the name of the optimal parameter vector in the output list of optimizer.

output_ignore A character vector of element names in the output list of optimizer that are ignored. The elements value and parameter are added automatically to output_ignore, because they are saved separately, see the output documentation of [apply_optimizer](#).

See Also

Use [apply_optimizer\(\)](#) to apply an optimizer object for numerical optimization.

Examples

```
define_optimizer(
  .optimizer = pracma::nelder_mead,      # optimization function
  .objective = "fn",                    # name of function input
  .initial = "x0",                      # name of initial input
  .value = "fmin",                      # name of value output
  .parameter = "xmin",                 # name of parameter output
  .direction = "min",                  # optimizer minimizes
  .output_ignore = c("restarts", "errmess"), # ignore some outputs
  tol = 1e-6,                          # additional optimizer argument
  .validate = TRUE                      # validate the object
)
```

Objective

Specify objective function

Description

The `Objective` object specifies the framework for an objective function for numerical optimization.

Value

An `Objective` object.

Active bindings

`objective_name` A character, a label for the objective function.

`fixed_arguments` A character, the names of the fixed arguments (if any).

`seconds` A numeric, a time limit in seconds. Computations are interrupted prematurely if seconds is exceeded.

No time limit if `seconds = Inf` (the default).

Note the limitations documented in [setTimeLimit](#).

`hide_warnings` Either TRUE to hide warnings when evaluating the objective function, or FALSE (default) if not.

`verbose` Either TRUE (default) to print status messages, or FALSE to hide those.

`npar` An integer vector, defining the length of each target argument.

`output_template` A template of the expected output value, used for the `validate` method.

Methods**Public methods:**

- [Objective\\$new\(\)](#)
- [Objective\\$set_argument\(\)](#)
- [Objective\\$get_argument\(\)](#)
- [Objective\\$remove_argument\(\)](#)
- [Objective\\$validate\(\)](#)
- [Objective\\$evaluate\(\)](#)
- [Objective\\$print\(\)](#)
- [Objective\\$clone\(\)](#)

Method `new()`: Creates a new Objective object.

Usage:

```
Objective$new(f, target = NULL, npar, ...)
```

Arguments:

`f` A function to be optimized.

It is expected that `f` has at least one numeric argument.

Further, it is expected that the return value of `f` is of the structure `numeric(1)`, i.e. a single numeric value (although this can be altered via the `output_template` field).

`target` A character, the argument name(s) of `f` that get optimized.

All target arguments must receive a numeric vector.

Can be NULL (default), then it is the first argument of `f`.

`npar` A integer of the same length as `target`, defining the length of the respective numeric vector argument.

`...` Optionally additional arguments to `f` that are fixed during the optimization.

Returns: A new Objective object.

Method `set_argument()`: Set a fixed function argument.

Usage:

```
Objective$set_argument(..., overwrite = TRUE, verbose = self$verbose)
```

Arguments:

... Optionally additional arguments to `f` that are fixed during the optimization.

`overwrite` Either TRUE (default) to allow overwriting, or FALSE if not.

`verbose` Either TRUE (default) to print status messages, or FALSE to hide those.

Returns: Invisibly the Objective object.

Method `get_argument()`: Get a fixed function argument.

Usage:

```
Objective$get_argument(argument_name, verbose = self$verbose)
```

Arguments:

`argument_name` A character, a name of an argument for `f`.

`verbose` Either TRUE (default) to print status messages, or FALSE to hide those.

Returns: The argument value.

Method `remove_argument()`: Remove a fixed function argument.

Usage:

```
Objective$remove_argument(argument_name, verbose = self$verbose)
```

Arguments:

`argument_name` A character, a name of an argument for `f`.

`verbose` Either TRUE (default) to print status messages, or FALSE to hide those.

Returns: Invisibly the Objective object.

Method `validate()`: Validate an Objective object.

Usage:

```
Objective$validate(.at)
```

Arguments:

`.at` A numeric of length `sum(self$npar)`, the values for the target arguments written in a single vector.

Returns: Invisibly the Objective object.

Method `evaluate()`: Evaluate the objective function.

Usage:

```
Objective$evaluate(.at, .negate = FALSE, ...)
```

Arguments:

`.at` A numeric of length `sum(self$npar)`, the values for the target arguments written in a single vector.

`.negate` Either TRUE to negate the numeric return value of `f`, or FALSE (default) else.

... Optionally additional arguments to `f` that are fixed during the optimization.

Returns: The objective value.

Method `print()`: Print details of the Objective object.

Usage:

```
Objective$print()
```

Returns: Invisibly the Objective object.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Objective$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### define log-likelihood function of Gaussian mixture model
llk <- function(mu, sd, lambda, data) {
  sd <- exp(sd)
  lambda <- plogis(lambda)
  cluster_1 <- lambda * dnorm(data, mu[1], sd[1])
  cluster_2 <- (1 - lambda) * dnorm(data, mu[2], sd[2])
  sum(log(cluster_1 + cluster_2))
}

### the log-likelihood function is supposed to be optimized over the first
### three arguments, the 'data' argument is constant
objective <- Objective$new(
  f = llk, target = c("mu", "sd", "lambda"), npar = c(2, 2, 1),
  data = faithful$eruptions
)

### evaluate the objective function at 1:5 (1:2 is passed to mu, 3:4 to sd,
### and 5 to lambda)
objective$evaluate(1:5)
```

Optimizer

Specify numerical optimizer as R6 object

Description

A `Optimizer` R6 object defines a numerical optimizer based on an optimization function implemented in R.

The main advantage of working with an `Optimizer` object instead of using the optimization function directly lies in the standardized inputs and outputs.

Any R function that fulfills the following four constraints can be defined as an `Optimizer` object:

1. It must have an input for a function, the objective function to be optimized.
2. It must have an input for a numeric vector, the initial values from where the optimizer starts.
3. It must have a `...` argument for additional parameters passed on to the objective function.
4. The output must be a named list, including the optimal function value and the optimal parameter vector.

Active bindings

`label` A character, the label for the optimizer.

`algorithm` A function, the optimization algorithm.

`arg_objective` A character, the argument name for the objective function in `algorithm`.

`arg_initial` A character, the argument name for the initial values in `algorithm`.

`out_value` A character, the element name for the optimal function value in the output list of `algorithm`.

`out_parameter` A character, the element name for the optimal parameters in the output list of `algorithm`.

`direction` Either "min" (if the optimizer minimizes) or "max" (if the optimizer maximizes).

`arguments` A named list of custom arguments for `algorithm`. Defaults are used for arguments that are not specified.

`seconds` A numeric, a time limit in seconds. Optimization is interrupted prematurely if `seconds` is exceeded.

No time limit if `seconds = Inf` (the default).

Note the limitations documented in [setTimeLimit](#).

`hide_warnings` Either TRUE to hide warnings during optimization, or FALSE (default) else.

`output_ignore` A character vector of elements to ignore in the optimization output.

Methods**Public methods:**

- [Optimizer\\$new\(\)](#)
- [Optimizer\\$definition\(\)](#)
- [Optimizer\\$set_arguments\(\)](#)
- [Optimizer\\$validate\(\)](#)
- [Optimizer\\$minimize\(\)](#)
- [Optimizer\\$maximize\(\)](#)
- [Optimizer\\$optimize\(\)](#)
- [Optimizer\\$print\(\)](#)
- [Optimizer\\$clone\(\)](#)

Method `new()`: Initializes a new `Optimizer` object.

Usage:

```
Optimizer$new(which, ...)
```

Arguments:

`which` A character, either one of `optimizer_dictionary$keys` or "custom" (in which case `$definition()` must be used to define the optimizer details).

`...` Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.

Returns: A new `Optimizer` object.

Method `definition()`: Defines an optimizer.

Usage:

```
Optimizer$definition(
  algorithm,
  arg_objective,
  arg_initial,
  out_value,
  out_parameter,
  direction
)
```

Arguments:

`algorithm` A function, the optimization algorithm.

`arg_objective` A character, the argument name for the objective function in `algorithm`.

`arg_initial` A character, the argument name for the initial values in `algorithm`.

`out_value` A character, the element name for the optimal function value in the output list of `algorithm`.

`out_parameter` A character, the element name for the optimal parameters in the output list of `algorithm`.

`direction` Either "min" (if the optimizer minimizes) or "max" (if the optimizer maximizes).

Returns: Invisibly the `Optimizer` object.

Method `set_arguments()`: Sets optimizer arguments.

Usage:

```
Optimizer$set_arguments(...)
```

Arguments:

`...` Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.

Returns: The `Optimizer` object.

Method `validate()`: Validates the `Optimizer` object. A time limit in seconds for the optimization can be set via the `$seconds` field.

Usage:

```
Optimizer$validate(
  objective = optimizeR::test_objective,
  initial = round(stats::rnorm(2)),
  ...,
  direction = "min"
)
```

Arguments:

`objective` A function to be optimized that

1. has at least one argument that receives a numeric vector
2. and returns a single numeric value.

Alternatively, it can also be a [Objective](#) object for more flexibility.

initial A numeric vector with starting parameter values for the optimization.
 ... Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.
direction Either "min" for minimization or "max" for maximization.
Returns: The Optimizer object.

Method minimize(): Performing minimization.

Usage:

```
Optimizer$minimize(objective, initial, ...)
```

Arguments:

objective A function to be optimized that

1. has at least one argument that receives a numeric vector
2. and returns a single numeric value.

Alternatively, it can also be a [Objective](#) object for more flexibility.

initial A numeric vector with starting parameter values for the optimization.

... Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.

Returns: A named list, containing at least these five elements:

value A numeric, the minimum function value.

parameter A numeric vector, the parameter vector where the minimum is obtained.

seconds A numeric, the optimization time in seconds.

initial A numeric, the initial parameter values.

error Either TRUE if an error occurred, or FALSE, else.

Appended are additional output elements of the optimizer.

If an error occurred, then the error message is also appended as element *error_message*.

If the time limit was exceeded, this also counts as an error. In addition, the flag *time_out* = TRUE is appended.

Examples:

```
Optimizer$new("stats::nlm")$
  minimize(objective = function(x) x^4 + 3*x - 5, initial = 2)
```

Method maximize(): Performing maximization.

Usage:

```
Optimizer$maximize(objective, initial, ...)
```

Arguments:

objective A function to be optimized that

1. has at least one argument that receives a numeric vector
2. and returns a single numeric value.

Alternatively, it can also be a [Objective](#) object for more flexibility.

initial A numeric vector with starting parameter values for the optimization.

... Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.

Returns: A named list, containing at least these five elements:

value A numeric, the maximum function value.

parameter A numeric vector, the parameter vector where the maximum is obtained.

seconds A numeric, the optimization time in seconds.

initial A numeric, the initial parameter values.

error Either TRUE if an error occurred, or FALSE, else.

Appended are additional output elements of the optimizer.

If an error occurred, then the error message is also appended as element `error_message`.

If the time limit was exceeded, this also counts as an error. In addition, the flag `time_out = TRUE` is appended.

Examples:

```
Optimizer$new("stats::nlm")$
  maximize(objective = function(x) -x^4 + 3*x - 5, initial = 2)
```

Method `optimize()`: Performing minimization or maximization.

Usage:

```
Optimizer$optimize(objective, initial, direction = "min", ...)
```

Arguments:

objective A function to be optimized that

1. has at least one argument that receives a numeric vector
2. and returns a single numeric value.

Alternatively, it can also be a [Objective](#) object for more flexibility.

initial A numeric vector with starting parameter values for the optimization.

direction Either "min" for minimization or "max" for maximization.

... Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.

Returns: A named list, containing at least these five elements:

value A numeric, the maximum function value.

parameter A numeric vector, the parameter vector where the maximum is obtained.

seconds A numeric, the optimization time in seconds.

initial A numeric, the initial parameter values.

error Either TRUE if an error occurred, or FALSE, else.

Appended are additional output elements of the optimizer.

If an error occurred, then the error message is also appended as element `error_message`.

If the time limit was exceeded, this also counts as an error. In addition, the flag `time_out = TRUE` is appended.

Examples:

```
objective <- function(x) -x^4 + 3*x - 5
optimizer <- Optimizer$new("stats::nlm")
optimizer$optimize(objective = objective, initial = 2, direction = "min")
optimizer$optimize(objective = objective, initial = 2, direction = "max")
```

Method `print()`: Prints the optimizer label.

Usage:

```
Optimizer$print(...)
```

Arguments:

... Optionally additional arguments to be passed to the optimizer algorithm. Without specifications, default values are used.

Returns: Invisibly the Optimizer object.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Optimizer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### Task: compare minimization with 'stats::nlm' and 'pracma::nelder_mead'

# 1. define objective function and initial values
objective <- TestFunctions::TF_ackley
initial <- c(3, 3)

# 2. get overview of optimizers in dictionary
optimizer_dictionary$keys

# 3. define 'nlm' optimizer
nlm <- Optimizer$new(which = "stats::nlm")

# 4. define the 'pracma::nelder_mead' optimizer (not contained in the dictionary)
nelder_mead <- Optimizer$new(which = "custom")
nelder_mead$definition(
  algorithm = pracma::nelder_mead, # the optimization function
  arg_objective = "fn",           # the argument name for the objective function
  arg_initial = "x0",            # the argument name for the initial values
  out_value = "fmin",           # the element for the optimal function value in the output
  out_parameter = "xmin",       # the element for the optimal parameters in the output
  direction = "min"             # the optimizer minimizes
)

# 5. compare the minimization results
nlm$minimize(objective, initial)
nelder_mead$minimize(objective, initial)

## -----
## Method `Optimizer$minimize`
## -----

Optimizer$new("stats::nlm")$
  minimize(objective = function(x) x^4 + 3*x - 5, initial = 2)
```

```
## -----
## Method `Optimizer$maximize`
## -----

Optimizer$new("stats::nlm")$
  maximize(objective = function(x) -x^4 + 3*x - 5, initial = 2)

## -----
## Method `Optimizer$optimize`
## -----

objective <- function(x) -x^4 + 3*x - 5
optimizer <- Optimizer$new("stats::nlm")
optimizer$optimize(objective = objective, initial = 2, direction = "min")
optimizer$optimize(objective = objective, initial = 2, direction = "max")
```

optimizer_dictionary *Dictionary of optimizer functions*

Description

The optimizer_dictionary object is a dictionary of currently implemented numerical optimizer functions.

Usage

```
optimizer_dictionary
```

Format

An R6 object of class [Dictionary](#).

ParameterSpaces *Switch Between Parameter Spaces*

Description

This R6 object manages two related parameter spaces: the Optimization Space (for optimization) and the Interpretation Space (for easier interpretation).

In the Optimization Space, parameters are stored as a numeric vector, the standard format for numerical optimizers. Parameters in this space are typically identified.

In the Interpretation Space, parameters are stored as a list and can take different formats (e.g., matrix). Parameters here do not need to be identified.

The user can define transformation functions (not necessarily bijective) to switch between these two spaces via the `$o2i()` and `$i2o()` methods.

Methods**Public methods:**

- `ParameterSpaces$new()`
- `ParameterSpaces$print()`
- `ParameterSpaces$switch()`
- `ParameterSpaces$o2i()`
- `ParameterSpaces$i2o()`
- `ParameterSpaces$clone()`

Method `new()`: Initializes a new `ParameterSpaces` object.

Usage:

```
ParameterSpaces$new(parameter_names, parameter_lengths_in_o_space)
```

Arguments:

`parameter_names` [`character()`]

Unique names for the parameters.

`parameter_lengths_in_o_space` [`integer()`]

The length of each parameter in the optimization space.

Returns: A new `ParameterSpaces` object.

Method `print()`: Print an overview of the parameter spaces.

Usage:

```
ParameterSpaces$print(show_transformer = FALSE)
```

Arguments:

`show_transformer` [`logical(1)`]

Show transformer functions in the output?

Method `switch()`: Switch between Optimization Space and Interpretation Space.

Usage:

```
ParameterSpaces$switch(x)
```

Arguments:

`x` [`numeric()` | `list()`]

The parameters, either as a `numeric` vector (will be switched to Interpretation Space), or as a `list()` (will be switched to Optimization Space).

Method `o2i()`: Define transformation functions when switching from Optimization Space to Interpretation Space.

Usage:

```
ParameterSpaces$o2i(...)
```

Arguments:

`...` [`function`]

One or more transformation functions, named according to the parameters.

Transformers from Optimization Space to Interpretation Space (`o2i`) **must receive** a `numeric`.

The default is the identity.

Method i2o(): Define transformation functions when switching from Interpretation Space to Optimization Space.

Usage:

```
ParameterSpaces%i2o(...)
```

Arguments:

```
... [function]
```

One or more transformers functions, named according to the parameters.

Transformers from Interpretation Space to Optimization Space (i2o) **must return** a numeric.

The default is `as.vector()`.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ParameterSpaces$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### Log-likelihood function of two-class Gaussian mixture model with
### parameter vector `theta` that consists of
### - `mu`, mean vector of length 2
### - `sd`, standard deviation vector of length 2, must be positive
### - `lambda`, class probability of length 1, must be between 0 and 1

normal_mixture_llk <- function(theta, data) {
  mu <- theta[1:2]
  sd <- exp(theta[3:4])
  lambda <- plogis(theta[5])
  c1 <- lambda * dnorm(data, mu[1], sd[1])
  c2 <- (1 - lambda) * dnorm(data, mu[2], sd[2])
  sum(log(c1 + c2))
}

### define parameter spaces
### - `mu` needs no transformation
### - `sd` needs to be real in optimization space and positive in
###   interpretation space
### - `lambda` needs to be real and of length `1` in optimization space, and
###   a probability vector of length `2` in interpretation space

normal_mixture_spaces <- ParameterSpaces$
  new(
    parameter_names = c("mu", "sd", "lambda"),
    parameter_lengths_in_o_space = c(2, 2, 1)
  )$
  o2i(
    "mu" = function(x) x,
    "sd" = function(x) exp(x),
    "lambda" = function(x) c(plogis(x), 1 - plogis(x))
  )
}
```



```

)$.
i2o(
  "mu" = function(x) x,
  "sd" = function(x) log(x),
  "lambda" = function(x) qlongis(x[1])
)

### switch between parameter spaces

par <- list(                                # parameters in interpretation space
  "mu" = c(2, 4),
  "sd" = c(0.5, 1),
  "lambda" = c(0.4, 0.6)
)
(x <- normal_mixture_spaces$switch(par)) # switch to optimization space
normal_mixture_llk(
  theta = x, data = datasets::faithful$eruptions
)
normal_mixture_spaces$switch(x)           # switch back

```

test_objective	<i>Test objective function</i>
----------------	--------------------------------

Description

This function is useful for testing or debugging the behavior of objective functions. It can throw a warning and / or an error on purpose.

Usage

```

test_objective(
  x,
  value = x,
  warning_prob = 0,
  error_prob = 0,
  warning_msg = "warning",
  error_msg = "error",
  call. = TRUE
)

```

Arguments

x	Any input.
value	The return value, any object.
warning_prob	The probability for throwing a warning.
error_prob	The probability for throwing an error.
warning_msg	The warning message.

error_msg The error message.
 call. Passed to `warning` or `stop`, respectively.

Value

The argument value.

test_optimizer	<i>Test optimization function</i>
----------------	-----------------------------------

Description

This function is useful for testing or debugging the behavior of optimization functions. It can throw a warning and / or an error on purpose.

Usage

```
test_optimizer(
  objective = test_objective,
  initial = 1,
  ...,
  parameter = 1,
  value = objective(parameter),
  seconds = 0,
  warning_prob = 0,
  error_prob = 0,
  warning_msg = "warning",
  error_msg = "error",
  call. = TRUE
)
```

Arguments

objective An objective function.
 initial The initial parameter vector.
 ... Optionally additional arguments to be passed to `objective`.
 parameter Defines the output parameter.
 value Defines the output value.
 seconds A delay in number of seconds.
 warning_prob The probability for throwing a warning.
 error_prob The probability for throwing an error.
 warning_msg The warning message.
 error_msg The error message.
 call. Passed to `warning` or `stop`, respectively.

Value

A list with elements parameter and value.

Index

* datasets

optimizer_dictionary, 14

apply_optimizer, 2, 4, 5

apply_optimizer(), 5

define_optimizer, 2, 3

define_optimizer(), 2

Dictionary, 14

nlm, 3

Objective, 5, 10–12

optim, 3

Optimizer, 8

optimizer_dictionary, 14

optimizer_nlm(define_optimizer), 3

optimizer_optim(define_optimizer), 3

ParameterSpaces, 14

setTimeLimit, 6, 9

stop, 18

test_objective, 17

test_optimizer, 18

warning, 18