Package: optimizeR (via r-universe)

November 20, 2026

Title Unified Framework for Numerical Optimizers

Version 1.4.0

Description Provides a unified object-oriented framework for numerical optimizers in R. Supports minimization and maximization with any optimizer, optimization over more than one function argument, computation time measurement, and time limits for long optimization tasks.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.0.0)

Imports checkmate, cli, methods, numDeriv, oeli (>= 0.7.2), R6, stats

Suggests ao, datasets, DEoptim, GenSA, ggplot2, lbfgsb3c, nloptr, optimx, pracma, rmarkdown, testthat, ucminf, vntrs

Config/testthat/edition 3

URL https://loelschlaeger.de/optimizeR/

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

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Lennart Oelschläger [aut, cre] (ORCID: <https://orcid.org/0000-0001-5421-9313>), Marius Ötting [ctb] (ORCID: <https://orcid.org/0000-0002-9373-0365>)

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

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 15:30:02 UTC

RemoteUrl https://github.com/cran/optimizeR

RemoteRef HEAD

RemoteSha ebabd83c1485f7e52f468224f1c6f7ffb82c287d

Contents

constructors 2
Objective 3
Optimizer 9
optimizer_dictionary 18
ParameterSpaces 19

Index **23**

`constructors` *Create optimizer and objective objects*

Description

The function `optimizer()` creates an `Optimizer` object and is equivalent to `Optimizer$new()`.

The function `objective()` creates an `Objective` object and is equivalent to `Objective$new()`.

Usage

```
optimizer(which, ...)

objective(f, target = NULL, npar, ...)
```

Arguments

`which`	[`character(1)`] Either: • one of `optimizer_dictionary$keys` (the package of the optimizer must be installed) • or `"custom"`, in which case `$definition()` must be used to define the optimizer details.
`...`	For `optimizer()`, named arguments to pass to the optimizer algorithm. Additionally, `.verbose = FALSE` can be set to suppress the status message for `which = "custom"`. For `objective()`, optionally additional function arguments that are fixed during the optimization.
`f`	[`function`] A `function` to be optimized.
`target`	[`character()`] The argument name(s) to optimize. If `NULL` (default), the first function argument is selected.
`npar`	[`integer()`] The length of each target argument.

Value

For `optimizer()`: An `Optimizer` object.

For `objective()`: An `Objective` object.

Examples

```
### the Rosenbrock function
rosenbrock <- function(x) 100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2

### minimize with 'stats::nlm'
nlm <- optimizer("stats::nlm", iterlim = 200)
nlm$minimize(rosenbrock, initial = c(-1.2, 1))

### evaluate the objective
rosenbrock_objective <- objective(rosenbrock, npar = 2)
rosenbrock_objective$evaluate(c(1, 1))
```

`Objective` *Specify objective function object*

Description

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

Value

An `Objective` object.

Active bindings

`objective_name [character(1)]`
: The label for the objective function.

`fixed_arguments [character(), read-only]`
: The name(s) of the fixed argument(s) (if any).

`seconds [numeric(1)]`
: A time limit in seconds for one function evaluation. Computations are interrupted prematurely if `seconds` is exceeded, in which case `$evaluate()` returns the string `"time limit reached"`.

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

Note the limitations documented in `setTimeLimit`.

`hide_warnings [logical(1)]`
: Hide warnings when evaluating the objective function?

`verbose [logical(1)]`
: Print status messages?

`npar [integer(), read-only]`
: The length of each target argument.

`target` [`character()`, read-only]
The argument name(s) to optimize.

`gradient_specified` [`logical(1)`, read-only]
Whether a gradient function has been specified via `$set_gradient()`.

`hessian_specified` [`logical(1)`, read-only]
Whether a Hessian function has been specified via `$set_hessian()`.

Methods

Public methods:

- `Objective$new()`
- `Objective$set_argument()`
- `Objective$get_argument()`
- `Objective$remove_argument()`
- `Objective$set_gradient()`
- `Objective$set_hessian()`
- `Objective$evaluate()`
- `Objective$evaluate_gradient()`
- `Objective$evaluate_gradient_numeric()`
- `Objective$evaluate_hessian()`
- `Objective$evaluate_hessian_numeric()`
- `Objective$print()`
- `Objective$clone()`

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

Usage:

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

Arguments:

`f` [`function`]
A `function` to be optimized.
It is expected that `f` has at least one `numeric` argument.
Also, the return value of `f` is expected to have the structure `numeric(1)`, i.e. a single `numeric` value.

`target` [`character()`]
The argument name(s) to optimize.
All target arguments must be `numeric`.
If `NULL` (default), the first function argument is selected.

`npar` [`integer()`]
The length of each target argument, i.e., the length(s) of the `numeric vector` argument(s) specified by `target`.

`...` Optionally additional function arguments that are fixed during the optimization.

`Objective$set_argument()`: Set a function argument that remains fixed during optimization. A target argument cannot be fixed.

Usage:

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

Arguments:

`...` Optionally additional function arguments that are fixed during the optimization.

`.overwrite [logical(1)]`
: Overwrite existing values?

`.verbose [logical(1)]`
: Print status messages?

`Objective$get_argument()`**:** Get a fixed function argument.

Usage:

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

Arguments:

`argument_name [character(1)]`
: A function argument name.

`.verbose [logical(1)]`
: Print status messages?

`Objective$remove_argument()`**:** Remove a fixed function argument.

Usage:

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

Arguments:

`argument_name [character(1)]`
: A function argument name.

`.verbose [logical(1)]`
: Print status messages?

`Objective$set_gradient()`**:** Set a gradient function.

Usage:

```
Objective$set_gradient(
  gradient,
  target = self$target,
  npar = self$npar,
  ...,
  .verbose = self$verbose
)
```

Arguments:

`gradient [function]`
: A `function` that computes the gradient of the objective function `f`.
It is expected that `gradient` has the same target argument(s) as `f`, and that `gradient` returns a `numeric vector` of length `sum(self$npar)`.
The fixed arguments of `f` are only passed on to `gradient` if it has them as arguments.

`target [character()]`
: The argument name(s) to optimize.
All target arguments must be `numeric`.
If `NULL` (default), the first function argument is selected.

`npar` [integer()]
: The length of each target argument, i.e., the length(s) of the `numeric` `vector` argument(s) specified by `target`.

`...` Optionally additional arguments that are fixed for `gradient`.

`.verbose` [logical(1)]
: Print status messages?

`Objective$set_hessian()`: Set a Hessian function.

Usage:

```
Objective$set_hessian(
  hessian,
  target = self$target,
  npar = self$npar,
  ...,
  .verbose = self$verbose
)
```

Arguments:

`hessian` [function]
: A `function` that computes the Hessian of the objective function `f`.
It is expected that `hessian` has the same target argument(s) as `f`, and that `hessian` returns a `numeric` `matrix` of dimension `sum(self$npar)` times `sum(self$npar)`.
The fixed arguments of `f` are only passed on to `hessian` if it has them as arguments.

`target` [character()]
: The argument name(s) to optimize.
All target arguments must be `numeric`.
If `NULL` (default), the first function argument is selected.

`npar` [integer()]
: The length of each target argument, i.e., the length(s) of the `numeric` `vector` argument(s) specified by `target`.

`...` Optionally additional arguments that are fixed for `hessian`.

`.verbose` [logical(1)]
: Print status messages?

`Objective$evaluate()`: Evaluate the objective function.

Usage:

```
Objective$evaluate(
  .at,
  .negate = FALSE,
  .gradient_as_attribute = FALSE,
  .gradient_attribute_name = "gradient",
  .gradient_numeric = FALSE,
  .hessian_as_attribute = FALSE,
  .hessian_attribute_name = "hessian",
  .hessian_numeric = FALSE,
  ...
)
```

Arguments:

`.at` [`numeric()`]
: The values for the target argument(s), written in a single vector.
Must be of length `sum(self$npar)`.

`.negate` [`logical(1)`]
: Negate the function return value?
In that case, derivatives that are attributes of the function return value (with the names `.gradient_attribute_name` and `.hessian_attribute_name`) are negated as well.

`.gradient_as_attribute` [`logical(1)`]
: Add the value of the gradient function as an attribute to the output?
The attribute name is defined via the `.gradient_attribute_name` argument.
Ignored if `$gradient_specified` and `.gradient_numeric` are `FALSE`.

`.gradient_attribute_name` [`character(1)`]
: Only relevant if `.gradient_as_attribute = TRUE`.
In that case, the attribute name for the gradient (if available).

`.gradient_numeric` [`logical(1)`]
: Calculate the gradient via the numerical approximation `grad`?

`.hessian_as_attribute` [`logical(1)`]
: Add the value of the Hessian function as an attribute to the output?
The attribute name is defined via the `.hessian_attribute_name` argument.
Ignored if `$hessian_specified` and `.hessian_numeric` are `FALSE`.

`.hessian_attribute_name` [`character(1)`]
: Only relevant if `.hessian_as_attribute = TRUE`.
In that case, the attribute name for the Hessian (if available).

`.hessian_numeric` [`logical(1)`]
: Calculate the Hessian via the numerical approximation `hessian`?

`...` Optionally additional function arguments for this evaluation, which have priority over fixed arguments of the same name.

Returns: The function value, or the string `"time limit reached"` if the time limit `$seconds` is exceeded.

`Objective$evaluate_gradient()`**:** Evaluate the gradient function.

Returns a `numeric` vector of `NA_real_` values if the evaluation fails or returns an unexpected result.

Usage:

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

Arguments:

`.at` [`numeric()`]
: The values for the target argument(s), written in a single vector.
Must be of length `sum(self$npar)`.

`.negate` [`logical(1)`]
: Negate the function return value?

`...` Optionally additional function arguments for this evaluation.

`Objective$evaluate_gradient_numeric()`**:** Evaluate the numerical gradient `grad`.

Returns a `numeric` vector of `NA_real_` values if the calculation fails or returns an unexpected result.

Usage:

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

Arguments:

`.at` [`numeric()`]
: The values for the target argument(s), written in a single vector. Must be of length `sum(self$npar)`.

`.negate` [`logical(1)`]
: Negate the function return value?

`...` Optionally additional function arguments for this evaluation.

`Objective$evaluate_hessian()`**:** Evaluate the Hessian function.

Returns an `NA_real_` value or matrix if the evaluation fails or returns an unexpected result.

The Hessian function can also return a single number if there is only one parameter, which is then returned as such.

Usage:

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

Arguments:

`.at` [`numeric()`]
: The values for the target argument(s), written in a single vector. Must be of length `sum(self$npar)`.

`.negate` [`logical(1)`]
: Negate the function return value?

`...` Optionally additional function arguments for this evaluation.

`Objective$evaluate_hessian_numeric()`**:** Evaluate the numerical Hessian `hessian`.

Returns an `NA_real_` matrix if the calculation fails or returns an unexpected result.

Usage:

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

Arguments:

`.at` [`numeric()`]
: The values for the target argument(s), written in a single vector. Must be of length `sum(self$npar)`.

`.negate` [`logical(1)`]
: Negate the function return value?

`...` Optionally additional function arguments for this evaluation.

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

Usage:

```
Objective$print(...)
```

Arguments:

`...` Currently not used.

Returns: Invisibly the `Objective` object.

`Objective$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))
}

### optimize 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 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 object*

Description

The `Optimizer` object defines a numerical optimizer based on any optimization algorithm 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 two constraints can be defined as an `Optimizer` object:

1. It must have an input for a `function`, the objective function to be optimized.
2. Its output must include the optimal function value and the optimal parameter vector.

Active bindings

`label [character(1)]`
: The label for the optimizer.

`algorithm [function]`
: The optimization algorithm.

`arg_objective [character(1)]`
: The argument name for the objective function in `algorithm`.

`arg_initial` [`character(1)` | `NA`]
: The argument name for the initial values in `algorithm`.
Can be `NA` if `algorithm` does not take initial values.

`arg_lower` [`character(1)` | `NA`]
: Optionally the argument name for the lower parameter bound in `algorithm`.
Can be `NA` if not available.

`arg_upper` [`character(1)` | `NA`]
: Optionally the argument name for the upper parameter bound in `algorithm`.
Can be `NA` if not available.

`arg_gradient` [`character(1)` | `NA`]
: Optionally the argument name for the gradient function in `algorithm`.
Can be `NA` if not available.

`arg_hessian` [`character(1)` | `NA`]
: Optionally the argument name for the Hessian function in `algorithm`.
Can be `NA` if not available.

`gradient_as_attribute` [`logical(1)`]
: Only relevant if `arg_gradient` is not `NA`.
In that case, does `algorithm` expect that the gradient is an attribute of the objective function output (as for example in `nlm`)? In that case, `arg_gradient` defines the attribute name.

`hessian_as_attribute` [`logical(1)`]
: Only relevant if `arg_hessian` is not `NA`.
In that case, does `algorithm` expect that the Hessian is an attribute of the objective function output (as for example in `nlm`)? In that case, `arg_hessian` defines the attribute name.

`out_value` [`character()` | `function`]
: The element name for the optimal function value in the output of `algorithm`.
Can be a vector of names if the output is nested.
Alternatively, it can also be a `function` that receives the output of `algorithm` and returns the optimal function value.

`out_parameter` [`character()` | `function`]
: The element name for the optimal parameters in the output of `algorithm`.
Can be a vector of names if the output is nested.
Alternatively, it can also be a `function` that receives the output of `algorithm` and returns the optimal parameters.

`direction` [`character(1)`]
: Either `"min"` (if the optimizer minimizes) or `"max"` (if the optimizer maximizes).

`arguments` [`list()`]
: Custom arguments for `algorithm`.
Defaults are used for arguments that are not specified.

`seconds` [`numeric(1)`]
: 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`.
The time limit refers to elapsed time and is also checked each time the optimizer evaluates the objective function.

`hide_warnings` [`logical(1)`]
Hide warnings during optimization?

`output_ignore` [`character()`]
Elements to ignore (not include) in the optimization output.

Methods

Public methods:

- `Optimizer$new()`
- `Optimizer$definition()`
- `Optimizer$set_arguments()`
- `Optimizer$minimize()`
- `Optimizer$maximize()`
- `Optimizer$optimize()`
- `Optimizer$print()`
- `Optimizer$clone()`

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

Usage:

```
Optimizer$new(which, ..., .verbose = TRUE)
```

Arguments:

`which` [`character(1)`]
Either:

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

`...` [`any`]
In `Optimizer$new()` and `$set_arguments()`, named arguments to pass to the optimizer algorithm. In `$minimize()`, `$maximize()`, and `$optimize()`, named arguments to pass to the objective function.

`.verbose` [`logical(1)`]
Print status messages?

Returns: A new `Optimizer` object.

`Optimizer$definition()`: Defines an optimizer.

Usage:

```
Optimizer$definition(
  algorithm,
  arg_objective,
  arg_initial,
  arg_lower = NA,
  arg_upper = NA,
  arg_gradient = NA,
  arg_hessian = NA,
  gradient_as_attribute = FALSE,
  hessian_as_attribute = FALSE,
```

```
  out_value,
  out_parameter,
  direction
)
```

Arguments:

`algorithm` [`function`]
: The optimization algorithm.

`arg_objective` [`character(1)`]
: The argument name for the objective function in `algorithm`.

`arg_initial` [`character(1)` | `NA`]
: The argument name for the initial values in `algorithm`.
Can be `NA` if `algorithm` does not take initial values.

`arg_lower` [`character(1)` | `NA`]
: Optionally the argument name for the lower parameter bound in `algorithm`.
Can be `NA` if not available.

`arg_upper` [`character(1)` | `NA`]
: Optionally the argument name for the upper parameter bound in `algorithm`.
Can be `NA` if not available.

`arg_gradient` [`character(1)` | `NA`]
: Optionally the argument name for the gradient function in `algorithm`.
Can be `NA` if not available.

`arg_hessian` [`character(1)` | `NA`]
: Optionally the argument name for the Hessian function in `algorithm`.
Can be `NA` if not available.

`gradient_as_attribute` [`logical(1)`]
: Only relevant if `arg_gradient` is not `NA`.
In that case, does `algorithm` expect that the gradient is an attribute of the objective function output (as for example in `nlm`)? In that case, `arg_gradient` defines the attribute name.

`hessian_as_attribute` [`logical(1)`]
: Only relevant if `arg_hessian` is not `NA`.
In that case, does `algorithm` expect that the Hessian is an attribute of the objective function output (as for example in `nlm`)? In that case, `arg_hessian` defines the attribute name.

`out_value` [`character()` | `function`]
: The element name for the optimal function value in the output of `algorithm`.
Can be a vector of names if the output is nested (e.g., `c("optim", "bestval")` for the element `bestval` of the element `optim`). If the output is an S4 object, the names refer to its slots.
Alternatively, it can also be a `function` that receives the output of `algorithm` and returns the optimal function value.

`out_parameter` [`character()` | `function`]
: The element name for the optimal parameters in the output of `algorithm`.
Can be a vector of names if the output is nested (e.g., `c("optim", "bestmem")` for the element `bestmem` of the element `optim`). If the output is an S4 object, the names refer to its slots.
Alternatively, it can also be a `function` that receives the output of `algorithm` and returns the optimal parameters.

`direction` [`character(1)`]
Either `"min"` (if the optimizer minimizes) or `"max"` (if the optimizer maximizes).

Returns: Invisibly the `Optimizer` object.

Examples:

```
### define 'stats::optimize', which does not take initial values
optimize <- Optimizer$new(which = "custom", .verbose = FALSE)
optimize$definition(
  algorithm = stats::optimize,
  arg_objective = "f",
  arg_initial = NA,
  arg_lower = "lower",
  arg_upper = "upper",
  out_value = "objective",
  out_parameter = "minimum",
  direction = "min"
)
optimize$minimize(
  objective = function(x) x^4 + 3*x - 5, initial = 2, lower = -5,
  upper = 5
)
```

`Optimizer$set_arguments()`: Sets optimizer arguments.

The arguments for the objective function and the initial values cannot be set here, since they are specified when calling `$minimize()`, `$maximize()`, or `$optimize()`.

Usage:

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

Arguments:

`...` [any]
In `Optimizer$new()` and `$set_arguments()`, named arguments to pass to the optimizer algorithm. In `$minimize()`, `$maximize()`, and `$optimize()`, named arguments to pass to the objective function.

Returns: Invisibly the `Optimizer` object.

`Optimizer$minimize()`: Perform minimization.

Usage:

```
Optimizer$minimize(objective, initial, lower = NA, upper = NA, ...)
```

Arguments:

`objective` [`function` | `Objective`]
A `function` to be optimized that

1. receives a `numeric` `vector` as its first argument (the so-called target argument)
2. and returns a single `numeric` value.

Alternatively, it can also be an `Objective` object for more flexibility (e.g., for another or more than one target argument, or for a gradient and Hessian function).

`initial` [`numeric()`]
Starting parameter values for the optimization (without missing values).
If the optimizer does not take initial values, they only define the number of parameters.

`lower` [`NA` | `numeric()` | `numeric(1)`]
Lower bounds on the parameters.
If a single number, this will be applied to all parameters.
Can be `NA` to not define any bounds.
Ignored (with a warning) for optimizers that do not support lower bounds.

`upper` [`NA` | `numeric()` | `numeric(1)`]
Upper bounds on the parameters.
If a single number, this will be applied to all parameters.
Can be `NA` to not define any bounds.
Ignored (with a warning) for optimizers that do not support upper bounds.

`...` [any]
In `Optimizer$new()` and `$set_arguments()`, named arguments to pass to the optimizer algorithm. In `$minimize()`, `$maximize()`, and `$optimize()`, named arguments to pass to the objective function.

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` A `logical` flag: `TRUE` if an error occurred, otherwise `FALSE`.

Additional output elements from the optimizer are appended. If the optimizer output is not a named `list`, it is appended as element `output`.
If an error occurred, the error message is also appended as element `error_message`.
If the time limit was exceeded, this 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)
```

`Optimizer$maximize()`: Perform maximization.

Usage:

```
Optimizer$maximize(objective, initial, lower = NA, upper = NA, ...)
```

Arguments:

`objective` [`function` | `Objective`]
A `function` to be optimized that

1. receives a `numeric` `vector` as its first argument (the so-called target argument)
2. and returns a single `numeric` value.

Alternatively, it can also be an `Objective` object for more flexibility (e.g., for another or more than one target argument, or for a gradient and Hessian function).

`initial` [`numeric()`]
Starting parameter values for the optimization (without missing values).
If the optimizer does not take initial values, they only define the number of parameters.

`lower` [`NA` | `numeric()` | `numeric(1)`]
Lower bounds on the parameters.
If a single number, this will be applied to all parameters.

Can be NA to not define any bounds.
Ignored (with a warning) for optimizers that do not support lower bounds.

`upper` [NA | numeric() | numeric(1)]
Upper bounds on the parameters.
If a single number, this will be applied to all parameters.
Can be NA to not define any bounds.
Ignored (with a warning) for optimizers that do not support upper bounds.

`...` [any]
In `Optimizer$new()` and `$set_arguments()`, named arguments to pass to the optimizer algorithm. In `$minimize()`, `$maximize()`, and `$optimize()`, named arguments to pass to the objective function.

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` A `logical` flag: `TRUE` if an error occurred, otherwise `FALSE`.

Additional output elements from the optimizer are appended. If the optimizer output is not a named `list`, it is appended as element `output`.
If an error occurred, 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)
```

`Optimizer$optimize()`: Perform minimization or maximization.

Usage:

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

Arguments:

`objective` [`function` | `Objective`]
A `function` to be optimized that

1. receives a `numeric` `vector` as its first argument (the so-called target argument)
2. and returns a single `numeric` value.

Alternatively, it can also be an `Objective` object for more flexibility (e.g., for another or more than one target argument, or for a gradient and Hessian function).

`initial` [`numeric()`]
Starting parameter values for the optimization (without missing values).
If the optimizer does not take initial values, they only define the number of parameters.

`lower` [`NA` | `numeric()` | `numeric(1)`]
Lower bounds on the parameters.
If a single number, this will be applied to all parameters.
Can be `NA` to not define any bounds.
Ignored (with a warning) for optimizers that do not support lower bounds.

`upper` [`NA` | `numeric()` | `numeric(1)`]
Upper bounds on the parameters.
If a single number, this will be applied to all parameters.
Can be `NA` to not define any bounds.
Ignored (with a warning) for optimizers that do not support upper bounds.

`direction` [`character(1)`]
Either `"min"` for minimization or `"max"` for maximization.

`...` [any]
In `Optimizer$new()` and `$set_arguments()`, named arguments to pass to the optimizer algorithm. In `$minimize()`, `$maximize()`, and `$optimize()`, named arguments to pass to the objective function.

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

`value` A `numeric`, the optimized function value.

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

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

`initial` A `numeric`, the initial parameter values.

`error` A `logical` flag: `TRUE` if an error occurred, otherwise `FALSE`.

Additional output elements from the optimizer are appended. If the optimizer output is not a named `list`, it is appended as element `output`.
If an error occurred, 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) sin(x)
nlm <- Optimizer$new("stats::nlm")
nlm$optimize(objective = objective, initial = 2, direction = "min")
nlm$optimize(objective = objective, initial = 2, direction = "max")
```

`Optimizer$print()`**:** Prints the optimizer label.

Usage:

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

Arguments:

`...` [any]
In `Optimizer$new()` and `$set_arguments()`, named arguments to pass to the optimizer algorithm. In `$minimize()`, `$maximize()`, and `$optimize()`, named arguments to pass to the objective function.

Returns: Invisibly the `Optimizer` object.

`Optimizer$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 (the Ackley function) and initial values
objective <- function(x) {
  -20 * exp(-0.2 * sqrt(mean(x^2))) - exp(mean(cos(2 * pi * x))) +
    20 + exp(1)
}
initial <- c(3, 3)

# 2. get overview of optimizers available 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, # optimization function
  arg_objective = "fn",            # objective function argument name
  arg_initial = "x0",              # argument name for the initial values
  out_value = "fmin",              # element for the optimal function value
  out_parameter = "xmin",          # element for the optimal parameters
  direction = "min"                # optimizer minimizes
)

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


## ------------------------------------------------
## Method `Optimizer$definition()`
## ------------------------------------------------

### define 'stats::optimize', which does not take initial values
optimize <- Optimizer$new(which = "custom", .verbose = FALSE)
optimize$definition(
  algorithm = stats::optimize,
  arg_objective = "f",
  arg_initial = NA,
  arg_lower = "lower",
  arg_upper = "upper",
  out_value = "objective",
  out_parameter = "minimum",
```

```
  direction = "min"
)
optimize$minimize(
  objective = function(x) x^4 + 3*x - 5, initial = 2, lower = -5,
  upper = 5
)

## ------------------------------------------------
## 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) sin(x)
nlm <- Optimizer$new("stats::nlm")
nlm$optimize(objective = objective, initial = 2, direction = "min")
nlm$optimize(objective = objective, initial = 2, direction = "max")
```

`optimizer_dictionary` *Dictionary of optimizer functions*

Description

A dictionary of currently included numerical optimizer functions in the `{optimizeR}` package, which can be selected via their label in `Optimizer$new()` and `optimizer()`.

Any other optimizer function can be defined via the method `$definition()` of an `Optimizer` object.

Usage

```
optimizer_dictionary
```

Format

An `R6` object of class `Dictionary`.

Adding optimizers

Further optimizers can be added to the dictionary via its method `$add()`. It requires the same elements as the entries of the dictionary (see the example below), i.e., a `label`, the `algorithm` (either the optimization function itself or its name in the form `"package::function"`, in which case the package is only required when an `Optimizer` object is created), and the arguments of the method `$definition()` of an `Optimizer` object. Note that the dictionary is built when `{optimizeR}` is loaded, such that added optimizers are available until the end of the R session.

Examples

```
print(optimizer_dictionary)

### the definition of an optimizer
str(optimizer_dictionary$get("stats::nlm"))

### add an optimizer to the dictionary
if (requireNamespace("pracma", quietly = TRUE)) {
  optimizer_dictionary$add(
    "label" = "pracma::nelder_mead",
    "algorithm" = "pracma::nelder_mead",
    "arg_objective" = "fn",
    "arg_initial" = "x0",
    "arg_lower" = NA,
    "arg_upper" = NA,
    "arg_gradient" = NA,
    "arg_hessian" = NA,
    "gradient_as_attribute" = FALSE,
    "hessian_as_attribute" = FALSE,
    "out_value" = "fmin",
    "out_parameter" = "xmin",
    "direction" = "min"
  )
  nelder_mead <- Optimizer$new("pracma::nelder_mead")
  nelder_mead$minimize(function(x) sum((x - 1)^2), initial = c(0, 0))
}
```

ParameterSpaces *Switch between parameter spaces*

Description

The `ParameterSpaces` object manages two related parameter spaces:

- the Optimization Space (for optimization)
- the Interpretation Space (for easier interpretation).

In the Optimization Space, parameters are stored as a `numeric vector`, the standard format for numerical optimizers.

In the Interpretation Space, parameters are stored as a `list` and can take different formats (e.g., `matrix`).

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

Methods

Public methods:

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

`ParameterSpaces$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.

`ParameterSpaces$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?

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

Usage:

```
ParameterSpaces$switch(x, to = NULL)
```

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).

`to` [`character(1)` | `NULL`]
Explicitly switch to a specific space, either

- `"o"`: Optimization Space
- `"i"`: Interpretation Space

If NULL, the function will switch to the other space.

`ParameterSpaces$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.

`ParameterSpaces$i2o()`: Define transformation functions when switching from Interpretation Space to Optimization Space.

Usage:

```
ParameterSpaces$i2o(...)
```

Arguments:

`...` [function]
One or more transformation functions, named according to the parameters.
Transformers from Interpretation Space to Optimization Space (i2o) **must return** a `numeric`.
The default is `as.vector()`.

`ParameterSpaces$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) qlogis(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
```

Index

constructors, 2

Dictionary, *18*

grad, *7*

hessian, *7*, *8*

nlm, *10*, *12*

Objective, *2*, 3, *3*, *13–15*
objective *(*constructors*)*, 2
Optimizer, *2*, *3*, 9, *18*, *19*
optimizer, *18*
optimizer *(*constructors*)*, 2
optimizer_dictionary, 18

ParameterSpaces, 19

setTimeLimit, *3*, *10*