Package: ao (via r-universe)

October 20, 2026

Title Alternating Optimization

Version 1.2.3

Description Implementation of an iterative process that optimizes a function by alternately performing restricted optimization over parameter subsets. Instead of solving one joint optimization problem, alternating optimization breaks it into smaller sub-problems. This approach can make optimization feasible when joint optimization is too difficult.

URL https://loelschlaeger.de/ao/, https://github.com/loelschlaeger/ao/

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

License GPL-3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports checkmate, cli, future.apply, oeli (>= 0.7.5), optimizeR (>= 1.2.1), progressr, R6, stats, utils

Suggests devtools, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 4.1.0)

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

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

Date/Publication 2026-05-11 14:45:03 UTC

RemoteUrl https://github.com/loelschlaeger/ao

RemoteRef HEAD

RemoteSha 13152394b936d379c72c3bf3ba3a1b7ad4fe6515

Contents

ao 2
generate_random_partition 7
merge_results 8
Process 9
split_by_target 17

Index **19**

---

ao *Alternating Optimization*

Description

Alternating optimization (AO) is an iterative process for optimizing a real-valued function jointly over all its parameters by alternating restricted optimization over parameter partitions.

Usage

```
ao(
  f,
  initial,
  target = NULL,
  npar = NULL,
  gradient = NULL,
  hessian = NULL,
  ...,
  partition = "sequential",
  new_block_probability = 0.3,
  minimum_block_number = 1,
  minimize = TRUE,
  lower = NULL,
  upper = NULL,
  iteration_limit = Inf,
  seconds_limit = Inf,
  tolerance_value = 1e-06,
  tolerance_parameter = 1e-06,
  tolerance_parameter_norm = function(x, y) sqrt(sum((x - y)^2)),
  tolerance_history = 1,
 base_optimizer = optimizeR::Optimizer$new("stats::optim", method = "L-BFGS-B", control
    = list(maxit = 10)),
  verbose = FALSE,
  hide_warnings = TRUE,
  add_details = TRUE
)
```

Arguments

`f` [function]
A function to be optimized, returning a single numeric value.
The first argument of f should be a numeric of the same length as initial, optionally followed by any other arguments specified by the ... argument.
If f is optimized over an argument other than the first, or over more than one argument, specify this with the target argument.

`initial` [numeric() | list()]
The starting parameter values for the target argument(s).
This can also be a list of multiple starting parameter values, see details.

`target` [character() | NULL]
The name(s) of the argument(s) over which f is optimized.
This can only be numeric arguments.
If NULL (default), the first argument of f is optimized.

`npar` [integer()]
The length(s) of the target argument(s).
Must be specified if more than one target argument is specified via the target argument.
Can be NULL if there is only one target argument, in which case npar is set to length(initial).

`gradient` [function | NULL]
An optional function that returns the gradient of f.
The function signature of gradient must be identical to f.
Ignored if base_optimizer does not support custom gradient.

`hessian` [function | NULL]
An optional function that returns the Hessian of f.
The function signature of hessian must be identical to f.
Ignored if base_optimizer does not support custom Hessian.

`...` Additional arguments to be passed to f (and gradient).

`partition` [character(1) | list()]
Defines the parameter partition. It can be

- "sequential" for treating each parameter separately,
- "random" for a random partition in each iteration,
- "none" for no partition (which is equivalent to joint optimization),
- or a list of vectors of parameter indices, specifying a custom partition for the AO process.

This can also be a list of multiple partition definitions, see details.

`new_block_probability` [numeric(1)]
Only relevant if partition = "random".
The probability of creating a new parameter block when creating a random partition.
Values close to 0 result in larger parameter blocks, values close to 1 result in smaller parameter blocks.

`minimum_block_number`
: [integer(1)]
Only relevant if `partition = "random"`.
The minimum number of blocks in random partitions.

`minimize`
: [logical(1)]
Minimize during the AO process?
If `FALSE`, maximization is performed.

`lower, upper`
: [numeric() | NULL]
Optional lower and upper parameter bounds.
Ignored if `base_optimizer` does not support parameter bounds.

`iteration_limit`
: [integer(1) | Inf]
The maximum number of iterations through the parameter partition before the AO process is terminated.
Can also be `Inf` for no iteration limit.

`seconds_limit`
: [numeric(1)]
The time limit in seconds before the AO process is terminated.
Can also be `Inf` for no time limit.
Note that this stopping criterion is only checked *after* a sub-problem is solved and not *within* solving a sub-problem, so the actual process time can exceed this limit.

`tolerance_value`
: [numeric(1)]
A non-negative tolerance value. The AO process terminates if the absolute difference between the current function value and the value from `tolerance_history` iterations earlier is smaller than `tolerance_value`.
Can be `0` for no value threshold.

`tolerance_parameter`
: [numeric(1)]
A non-negative tolerance value. The AO process terminates if the distance between the current estimate and the estimate from `tolerance_history` iterations earlier is smaller than `tolerance_parameter`.
Can be `0` for no parameter threshold.
By default, the distance is measured using the Euclidean norm, but another norm can be specified via the `tolerance_parameter_norm` argument.

`tolerance_parameter_norm`
: [function]
The norm that measures the distance between two estimates. If the distance is smaller than `tolerance_parameter`, the AO process is terminated.
It must be of the form `function(x, y)` for two vector inputs `x` and `y`, and return a single `numeric` value. By default, the Euclidean norm `function(x, y) sqrt(sum((x - y)^2))` is used.

`tolerance_history`
: [integer(1)]
The number of iterations to look back to determine whether `tolerance_value` or `tolerance_parameter` has been reached.

`base_optimizer` [`Optimizer | list()`]
An `Optimizer` object, which can be created via `Optimizer`. It numerically solves the sub-problems.
By default, the `optim` optimizer with `method = "L-BFGS-B"` is used. The default value for the number of iterations within each block is 10.
This can also be a `list` of multiple base optimizers, see details.

`verbose` [`logical(1)`]
Print tracing details during the AO process?
Not supported when using multiple processes, see details.

`hide_warnings` [`logical(1)`]
Hide warnings during the AO process?

`add_details` [`logical(1)`]
Add details about the AO process to the output?

Details

Multiple processes:

AO can suffer from local optima. To increase the likelihood of finding a better optimum, you can specify:

- multiple starting parameters
- multiple parameter partitions
- multiple base optimizers

Use the `initial`, `partition`, and/or `base_optimizer` arguments to provide a `list` of possible values for each parameter. Each combination of initial values, parameter partitions, and base optimizers will create a separate AO process.

Output value:

In the case of multiple processes, the output values refer to the best AO process with respect to function value.

If `add_details = TRUE`, the following elements are added:

- `estimates` is a `list` of optimal parameters in each process.
- `values` is a `list` of optimal function values in each process.
- `details` combines details of the single processes and has an additional column `process` with an index for the different processes.
- `seconds_each` gives the computation time in seconds for each process.
- `stopping_reasons` gives the termination message for each process.
- `processes` gives details on how the processes were specified.

Parallel computation:

By default, processes run sequentially. However, since they are independent, they can be parallelized. To enable parallel computation, use the `{future}` framework. For example, run the following *before* the `ao()` call:

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

Progress updates:

When using multiple processes, setting `verbose = TRUE` to print tracing details during AO is not supported. However, you can still track the progress using the `{progressr}` framework. For example, run the following *before* the `ao()` call:

```
progressr::handlers(global = TRUE)
progressr::handlers(
  progressr::handler_progress(":percent :eta :message")
)
```

Value

A `list` with the following elements:

- `estimate` is the parameter vector at termination.
- `estimate_split` is `estimate` split by `target` (only when applicable).
- `value` is the function value at termination.
- `details` is a `data.frame` with information about the AO process: For each iteration (column `iteration`) it contains the function value (column `value`), parameter values (columns starting with `p` followed by the parameter index), the active parameter block (columns starting with `b` followed by the parameter index, where `1` stands for a parameter contained in the active parameter block and `0` if not), and computation times in seconds (column `seconds`). Only available if `add_details = TRUE`.
- `seconds` is the overall computation time in seconds.
- `stopping_reason` is a message explaining why the AO process terminated.

In the case of multiple processes, the output changes slightly, see details.

Examples

```
# Example 1: Minimization of Himmelblau's function --------------------------

himmelblau <- function(x) (x[1]^2 + x[2] - 11)^2 + (x[1] + x[2]^2 - 7)^2
ao(f = himmelblau, initial = c(0, 0))

# Example 2: Maximization of 2-class Gaussian mixture log-likelihood --------

normal_mixture_loglik_uc = function(mu, logsd, eta, data) {
  sd <- exp(logsd)
  e <- exp(eta[1])
  den <- 1 + e
  q1 <- e / den
  q2 <- 1 / den
  l1 <- log(q1) + dnorm(data, mu[1], sd[1], log = TRUE)
  l2 <- log(q2) + dnorm(data, mu[2], sd[2], log = TRUE)
  m <- pmax(l1, l2)
  sum(m + log(exp(l1 - m) + exp(l2 - m)))
}

set.seed(123)

data <- datasets::faithful$eruptions

fit <- ao(
  f = normal_mixture_loglik_uc,
```

```
  initial = c(mean(data) + c(-1, 1), rep(log(sd(data)), 2), 0),
  target = c("mu", "logsd", "eta"),
  npar = c(2, 2, 1),
  data = data,
  partition = "random",
  base_optimizer = optimizeR::Optimizer$new("ucminf::ucminf"),
  minimize = FALSE,
  add_details = FALSE
)

(muhat <- fit$estimate_split$mu)
(sdhat <- exp(fit$estimate_split$logsd))
e <- exp(fit$estimate_split$eta)
den <- 1 + e
(qhat <- c(e / den, 1 / den))

# Example 3: Constrained Optimization in the Setting of Example 2 -----------

# target arguments:
# - class means mu (2, unrestricted)
# - class standard deviations sd (2, must be non-negative)
# - class proportion lambda (only 1 for identification, must be in [0, 1])

normal_mixture_loglik <- function(mu, sd, lambda, data) {
  c1 <- lambda * dnorm(data, mu[1], sd[1])
  c2 <- (1 - lambda) * dnorm(data, mu[2], sd[2])
  sum(log(c1 + c2))
}

set.seed(123)

ao(
  f = normal_mixture_loglik,
  initial = runif(5),
  target = c("mu", "sd", "lambda"),
  npar = c(2, 2, 1),
  data = datasets::faithful$eruptions,
  partition = list("sequential", "random", "none"),
  minimize = FALSE,
  lower = c(-Inf, -Inf, 0, 0, 0),
  upper = c(Inf, Inf, Inf, Inf, 1),
  add_details = FALSE
)
```

`generate_random_partition`

Generate random partition

Description

This helper function generates a random parameter partition, which is used for the randomized AO procedure.

Usage

```
generate_random_partition(x, p, min)
```

Arguments

x	[integer] The parameter indices.
p	[numeric(1)] The probability of generating a new block.
min	[integer(1)] The minimum number of blocks.

Value

A `list`, a random partition of `x`.

Author(s)

Siddhartha Chib

---

merge_results *Merge optimization results*

Description

This helper function merges the results of multiple AO processes.

Usage

```
merge_results(
  results,
  minimize = TRUE,
  add_details = TRUE,
  processes = data.frame()
)
```

Arguments

`results`	[list] A `list` of outputs from ao.
`minimize`	[logical(1)] Minimize during the AO process? If `FALSE`, maximization is performed.
`add_details`	[logical(1)] Add details about the AO process to the output?
`processes`	[data.frame] A `data.frame` describing how the different processes were specified.

Value

A `list`, see section "Output value" on the ao page.

`Process` *Process Object*

Description

This object specifies an AO process.

Active bindings

`npar [integer()]`
: The length(s) of the target argument(s).

`partition [character(1)|list()]`
: Defines the parameter partition. It can be

- `"sequential"` for treating each parameter separately,
- `"random"` for a random partition in each iteration,
- `"none"` for no partition (which is equivalent to joint optimization),
- or a `list` of vectors of parameter indices, specifying a custom partition for the AO process.

`new_block_probability [numeric(1)]`
: Only relevant if `partition = "random"`.

The probability of creating a new parameter block when creating a random partition.

Values close to 0 result in larger parameter blocks, values close to 1 result in smaller parameter blocks.

`minimum_block_number [integer(1)]`
: Only relevant if `partition = "random"`.

The minimum number of blocks in random partitions.

`verbose [logical(1)]`
: Print tracing details during the AO process?

`minimize` [`logical(1)`]
: Minimize during the AO process?

  If `FALSE`, maximization is performed.

`iteration_limit` [`integer(1)` | `Inf`]
: The maximum number of iterations through the parameter partition before the AO process is terminated.

  Can also be `Inf` for no iteration limit.

`seconds_limit` [`numeric(1)`]
: The time limit in seconds before the AO process is terminated.

  Can also be `Inf` for no time limit.

  Note that this stopping criterion is only checked *after* a sub-problem is solved and not *within* solving a sub-problem, so the actual process time can exceed this limit.

`tolerance_value` [`numeric(1)`]
: A non-negative tolerance value. The AO process terminates if the absolute difference between the current function value and the value from `tolerance_history` iterations earlier is smaller than `tolerance_value`.

  Can be `0` for no value threshold.

`tolerance_parameter` [`numeric(1)`]
: A non-negative tolerance value. The AO process terminates if the distance between the current estimate and the estimate from `tolerance_history` iterations earlier is smaller than `tolerance_parameter`.

  Can be `0` for no parameter threshold.

  By default, the distance is measured using the Euclidean norm, but another norm can be specified via the `tolerance_parameter_norm` field.

`tolerance_parameter_norm` [`function`]
: The norm that measures the distance between two estimates. If the distance is smaller than `tolerance_parameter`, the AO process is terminated.

  It must be of the form `function(x, y)` for two vector inputs `x` and `y`, and return a single `numeric` value. By default, the Euclidean norm `function(x, y) sqrt(sum((x - y)^2))` is used.

`tolerance_history` [`integer(1)`]
: The number of iterations to look back to determine whether `tolerance_value` or `tolerance_parameter` has been reached.

`add_details` [`logical(1)`]
: Add details about the AO process to the output?

`iteration` [`integer(1)`]
: The current iteration number.

`block` [`integer()`]
: The currently active parameter block, represented as parameter indices.

`output` [`list()`, read-only]
: The output of the AO process, which is a `list` with the following elements:

  - `estimate` is the parameter vector at termination.
  - `value` is the function value at termination.

- `details` is a `data.frame` with full information about the AO process. For each iteration (column `iteration`) it contains the function value (column `value`), parameter values (columns starting with `p` followed by the parameter index), the active parameter block (columns starting with `b` followed by the parameter index, where `1` stands for a parameter contained in the active parameter block and `0` if not), and computation times in seconds (column `seconds`). Only available if `add_details = TRUE`.
- `seconds` is the overall computation time in seconds.
- `stopping_reason` is a message explaining why the AO process terminated.

Methods

Public methods:

- `Process$new()`
- `Process$print_status()`
- `Process$initialize_details()`
- `Process$update_details()`
- `Process$get_partition()`
- `Process$get_details()`
- `Process$get_value()`
- `Process$get_value_latest()`
- `Process$get_value_best()`
- `Process$get_parameter()`
- `Process$get_parameter_latest()`
- `Process$get_parameter_best()`
- `Process$get_seconds()`
- `Process$get_seconds_total()`
- `Process$check_stopping()`

Method `new()`**:** Creates a new object of this R6 class.

Usage:

```
Process$new(
  target = NULL,
  npar = integer(),
  partition = "sequential",
  new_block_probability = 0.3,
  minimum_block_number = 1,
  verbose = FALSE,
  minimize = TRUE,
  iteration_limit = Inf,
  seconds_limit = Inf,
  tolerance_value = 1e-06,
  tolerance_parameter = 1e-06,
  tolerance_parameter_norm = function(x, y) sqrt(sum((x - y)^2)),
  tolerance_history = 1,
  add_details = TRUE
)
```

Arguments:

`target [character() | NULL]`
: The name(s) of the argument(s) over which `f` is optimized.
This can only be `numeric` arguments.
If `NULL` (default), the first argument of `f` is optimized.

`npar [integer()]`
: The length(s) of the target argument(s).

`partition [character(1) | list()]`
: Defines the parameter partition. It can be
- `"sequential"` for treating each parameter separately,
- `"random"` for a random partition in each iteration,
- `"none"` for no partition (which is equivalent to joint optimization),
- or a `list` of vectors of parameter indices, specifying a custom partition for the AO process.

`new_block_probability [numeric(1)]`
: Only relevant if `partition = "random"`.
The probability of creating a new parameter block when creating a random partition.
Values close to 0 result in larger parameter blocks, values close to 1 result in smaller parameter blocks.

`minimum_block_number [integer(1)]`
: Only relevant if `partition = "random"`.
The minimum number of blocks in random partitions.

`verbose [logical(1)]`
: Print tracing details during the AO process?

`minimize [logical(1)]`
: Minimize during the AO process?
If `FALSE`, maximization is performed.

`iteration_limit [integer(1) | Inf]`
: The maximum number of iterations through the parameter partition before the AO process is terminated.
Can also be `Inf` for no iteration limit.

`seconds_limit [numeric(1)]`
: The time limit in seconds before the AO process is terminated.
Can also be `Inf` for no time limit.
Note that this stopping criterion is only checked *after* a sub-problem is solved and not *within* solving a sub-problem, so the actual process time can exceed this limit.

`tolerance_value [numeric(1)]`
: A non-negative tolerance value. The AO process terminates if the absolute difference between the current function value and the value from `tolerance_history` iterations earlier is smaller than `tolerance_value`.
Can be `0` for no value threshold.

`tolerance_parameter [numeric(1)]`
: A non-negative tolerance value. The AO process terminates if the distance between the current estimate and the estimate from `tolerance_history` iterations earlier is smaller than `tolerance_parameter`.
Can be `0` for no parameter threshold.

By default, the distance is measured using the Euclidean norm, but another norm can be specified via the `tolerance_parameter_norm` field.

`tolerance_parameter_norm` [`function`]
The norm that measures the distance between two estimates. If the distance is smaller than `tolerance_parameter`, the AO process is terminated.
It must be of the form `function(x, y)` for two vector inputs `x` and `y`, and return a single `numeric` value. By default, the Euclidean norm `function(x, y) sqrt(sum((x - y)^2))` is used.

`tolerance_history` [`integer(1)`]
The number of iterations to look back to determine whether `tolerance_value` or `tolerance_parameter` has been reached.

`add_details` [`logical(1)`]
Add details about the AO process to the output?

Method `print_status()`**:** Prints a status message.

Usage:

```
Process$print_status(message, message_type = 8, verbose = self$verbose)
```

Arguments:

`message` [`character(1)`]
A status message.

`message_type` [`integer(1)`]
The message type, one of the following:

- `1` for `cli::cli_h1()`
- `2` for `cli::cli_h2()`
- `3` for `cli::cli_h3()`
- `4` for `cli::cli_alert_success()`
- `5` for `cli::cli_alert_info()`
- `6` for `cli::cli_alert_warning()`
- `7` for `cli::cli_alert_danger()`
- `8` for `cli::cat_line()`

`verbose` [`logical(1)`]
Print tracing details during the AO process?

Method `initialize_details()`**:** Initializes the `details` part of the output.

Usage:

```
Process$initialize_details(initial_parameter, initial_value)
```

Arguments:

`initial_parameter` [`numeric()`]
The starting parameter values for the AO process.

`initial_value` [`numeric(1)`]
The function value at the initial parameters.

Method `update_details()`**:** Updates the `details` part of the output.

Usage:

```
Process$update_details(
  value,
  parameter_block,
  seconds,
  error,
  error_message,
  block = self$block
)
```

Arguments:

`value` [`numeric(1)`]
: The updated function value.

`parameter_block` [`numeric()`]
: The updated parameter values for the active parameter block.

`seconds` [`numeric(1)`]
: The time in seconds for solving the sub-problem.

`error` [`logical(1)`]
: Did solving the sub-problem result in an error?

`error_message` [`character(1)`]
: An error message if `error = TRUE`.

`block` [`integer()`]
: The currently active parameter block, represented as parameter indices.

Method `get_partition()`**:** Get a parameter partition.

Usage:

```
Process$get_partition()
```

Method `get_details()`**:** Get the `details` part of the output.

Usage:

```
Process$get_details(
  which_iteration = NULL,
  which_block = NULL,
  which_column = c("iteration", "value", "parameter", "block", "seconds")
)
```

Arguments:

`which_iteration` [`integer()`]
: Selects the iteration(s).
Can also be `NULL` to select all iterations.

`which_block` [`character(1)` | `integer()`]
: Selects the parameter block in the partition and can be one of
  - `"first"` for the first parameter block,
  - `"last"` for the last parameter block,
  - an `integer` vector of parameter indices,
  - or `NULL` for all parameter blocks.

`which_column` [`character()`]
: Selects the columns in the `details` part of the output and can be one or more of

- "iteration",
- "value",
- "parameter",
- "block",
- and "seconds".

Method `get_value()`**:** Get the function value in different steps of the AO process.

Usage:

```
Process$get_value(
  which_iteration = NULL,
  which_block = NULL,
  keep_iteration_column = FALSE,
  keep_block_columns = FALSE
)
```

Arguments:

`which_iteration [integer()]`
: Selects the iteration(s).
Can also be `NULL` to select all iterations.

`which_block [character(1) | integer()]`
: Selects the parameter block in the partition and can be one of
- `"first"` for the first parameter block,
- `"last"` for the last parameter block,
- an `integer` vector of parameter indices,
- or `NULL` for all parameter blocks.

`keep_iteration_column [logical(1)]`
: Keep the column containing the information about the iteration in the output?

`keep_block_columns [logical(1)]`
: Keep the column containing the information about the active parameter block in the output?

Method `get_value_latest()`**:** Get the function value in the latest step of the AO process.

Usage:

```
Process$get_value_latest()
```

Method `get_value_best()`**:** Get the optimum function value in the AO process.

Usage:

```
Process$get_value_best()
```

Method `get_parameter()`**:** Get the parameter values in different steps of the AO process.

Usage:

```
Process$get_parameter(
  which_iteration = self$iteration,
  which_block = NULL,
  keep_iteration_column = FALSE,
  keep_block_columns = FALSE
)
```

Arguments:

`which_iteration [integer()]`
Selects the iteration(s).
Can also be `NULL` to select all iterations.

`which_block [character(1) | integer()]`
Selects the parameter block in the partition and can be one of

- `"first"` for the first parameter block,
- `"last"` for the last parameter block,
- an `integer` vector of parameter indices,
- or `NULL` for all parameter blocks.

`keep_iteration_column [logical(1)]`
Keep the column containing the information about the iteration in the output?

`keep_block_columns [logical(1)]`
Keep the column containing the information about the active parameter block in the output?

Method `get_parameter_latest()`**:** Get the parameter value in the latest step of the AO process.

Usage:

```
Process$get_parameter_latest(parameter_type = "full")
```

Arguments:

`parameter_type [character(1)]`
Selects the parameter type and can be one of

- `"full"` (default) to get the full parameter vector,
- `"block"` to get the parameter values for the current block, i.e., the parameters with the indices `self$block`
- `"fixed"` to get the parameter values which are currently fixed, i.e., all except for those with the indices `self$block`

Method `get_parameter_best()`**:** Get the optimum parameter value in the AO process.

Usage:

```
Process$get_parameter_best(parameter_type = "full")
```

Arguments:

`parameter_type [character(1)]`
Selects the parameter type and can be one of

- `"full"` (default) to get the full parameter vector,
- `"block"` to get the parameter values for the current block, i.e., the parameters with the indices `self$block`
- `"fixed"` to get the parameter values which are currently fixed, i.e., all except for those with the indices `self$block`

Method `get_seconds()`**:** Get the optimization time in seconds in different steps of the AO process.

Usage:

```
Process$get_seconds(
  which_iteration = NULL,
  which_block = NULL,
  keep_iteration_column = FALSE,
  keep_block_columns = FALSE
)
```

Arguments:

`which_iteration [integer()]`
: Selects the iteration(s).
Can also be `NULL` to select all iterations.

`which_block [character(1) | integer()]`
: Selects the parameter block in the partition and can be one of

- `"first"` for the first parameter block,
- `"last"` for the last parameter block,
- an `integer` vector of parameter indices,
- or `NULL` for all parameter blocks.

`keep_iteration_column [logical(1)]`
: Keep the column containing the information about the iteration in the output?

`keep_block_columns [logical(1)]`
: Keep the column containing the information about the active parameter block in the output?

Method `get_seconds_total()`**:** Get the total optimization time in seconds of the AO process.

Usage:

```
Process$get_seconds_total()
```

Method `check_stopping()`**:** Checks if the AO process can be terminated.

Usage:

```
Process$check_stopping()
```

`split_by_target` *Split estimate by target*

Description

This helper function splits the solution by target parameters (if provided), which is used for the output.

Usage

```
split_by_target(estimate, target = NULL, npar)
```

Arguments

estimate	[numeric] A parameter vector.
target	[character() | NULL] The name(s) of the argument(s) over which f is optimized. This can only be numeric arguments. If NULL (default), the first argument of f is optimized.
npar	[integer()] The length(s) of the target argument(s). Must be specified if more than one target argument is specified via the target argument. Can be NULL if there is only one target argument, in which case npar is set to length(initial).

Value

A (named) list, a partition of estimate according to npar.

Author(s)

Siddhartha Chib

Index

ao, 2, *9*

generate_random_partition, 7

merge_results, 8

optim, *5*
Optimizer, *5*

Process, 9

R6, *11*

split_by_target, 17