

Package: ao (via r-universe)

September 2, 2024

Title Alternating Optimization

Version 1.1.1

Description Alternating optimization is an iterative procedure that optimizes a function by alternately performing restricted optimization over individual parameter subsets. Instead of tackling joint optimization directly, it breaks the problem down into simpler 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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports checkmate, cli, future.apply, oeli (>= 0.5.2), progressr, R6, stats, utils

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

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 4.0.0), optimizeR (>= 1.1.1)

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

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

RemoteRef HEAD

RemoteSha 7f1f110f0ce41c25cf6573bf3729a078a6bde35b

Contents

ao	2
ao_input_check	6
Procedure	7

ao	<i>Alternating Optimization</i>
----	---------------------------------

Description

Alternating optimization is an iterative procedure 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,
  ...,
  partition = "sequential",
  new_block_probability = 0.3,
  minimum_block_number = 2,
  minimize = TRUE,
  lower = -Inf,
  upper = Inf,
  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$new("stats::optim", method = "L-BFGS-B"),
  verbose = FALSE,
  hide_warnings = TRUE
)
```

Arguments

f	<p>(function)</p> <p>A function to be optimized, returning a single numeric value.</p> <p>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.</p> <p>If f is to be optimized over an argument other than the first, or more than one argument, this has to be specified via the target argument.</p>
initial	<p>(numeric() or list())</p> <p>The starting parameter values for the target argument(s).</p> <p>This can also be a list of multiple starting parameter values, see details.</p>

target	(character() or NULL) The name(s) of the argument(s) over which <i>f</i> gets optimized. This can only be numeric arguments. Can be NULL (default), then it is the first argument of <i>f</i> .
npar	(integer()) The length of the target argument(s). Must be specified if more than two target arguments are specified via the target argument. Can be NULL if there is only one target argument, in which case <i>npar</i> is set to be <code>length(initial)</code> .
gradient	(function or NULL) A function that returns the gradient of <i>f</i> . The function call of <i>gradient</i> must be identical to <i>f</i> . Can be NULL, in which case a finite-difference approximation will be used.
...	Additional arguments to be passed to <i>f</i> (and <i>gradient</i>).
partition	(character(1) or list()) Defines the parameter partition, and can be either <ul style="list-style-type: none"> • "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 alternating optimization process. <p>This can also be a list of multiple partition definitions, see details.</p>
new_block_probability	(numeric(1)) Only relevant if <code>partition = "random"</code> . The probability for a new parameter block when creating a random partitions. 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 <code>partition = "random"</code> . The minimum number of blocks in random partitions.
minimize	(logical(1)) Whether to minimize during the alternating optimization process. If FALSE, maximization is performed.
lower, upper	(numeric()) Optionally lower and upper parameter bounds.
iteration_limit	(integer(1) or Inf) The maximum number of iterations through the parameter partition before the alternating optimization process is terminated. Can also be Inf for no iteration limit.

seconds_limit	(numeric(1)) The time limit in seconds before the alternating optimization process is terminated. Can also be Inf for no time limit. Note that this stopping criteria is only checked <i>after</i> a sub-problem is solved and not <i>within</i> solving a sub-problem, so the actual process time can exceed this limit.
tolerance_value	(numeric(1)) A non-negative tolerance value. The alternating optimization terminates if the absolute difference between the current function value and the one before tolerance_history iterations is smaller than tolerance_value. Can be 0 for no value threshold.
tolerance_parameter	(numeric(1)) A non-negative tolerance value. The alternating optimization terminates if the distance between the current estimate and the before tolerance_history iterations 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 the current estimate and the one from the last iteration. If the distance is smaller than tolerance_parameter, the procedure 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 or list()) An Optimizer object, which can be created via Optimizer . It numerically solves the sub-problems. By default, the <code>optim</code> optimizer is used. If another optimizer is specified, the arguments gradient, lower, and upper are ignored. This can also be a list of multiple base optimizers, see details.
verbose	(logical(1)) Whether to print tracing details during the alternating optimization process.
hide_warnings	(logical(1)) Whether to hide warnings during the alternating optimization process.

Details

Multiple threads:

Alternating optimization can suffer from local optima. To increase the likelihood of reaching the global 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 alternating optimization thread.

Output value:

In the case of multiple threads, the output changes slightly in comparison to the standard case. It is a list with the following elements:

- `estimate` is the optimal parameter vector over all threads.
- `estimates` is a list of optimal parameters in each thread.
- `value` is the optimal function value over all threads.
- `values` is a list of optimal function values in each thread.
- `details` combines details of the single threads and has an additional column `thread` with an index for the different threads.
- `seconds` gives the computation time in seconds for each thread.
- `stopping_reason` gives the termination message for each thread.
- `threads` give details how the different threads were specified.

Parallel computation:

By default, threads 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 threads, setting `verbose = TRUE` to print tracing details during alternating optimization is not supported. However, you can still track the progress of threads 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.
- `value` is the function value at termination.
- `details` is a `data.frame` with full information about the procedure: 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`)

- seconds is the overall computation time in seconds.
- stopping_reason is a message why the procedure has terminated.

In the case of multiple threads, 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 -----

# 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_llk <- 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_llk,
  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)
)
```

ao_input_check

Input checks

Description

This helper function checks the inputs for the `ao` function.

Usage

```

ao_input_check(
  argument_name,
  check_result,
  error_message = check_result,
  prefix = "Input {.var {argument_name}} is bad:"
)

```

Arguments

argument_name	(character(1))	The name of the argument that is checked.
check_result	(logical(1))	TRUE if the check was successful.
error_message	(character(1))	An error message to be printed.
prefix	(character(1))	A prefix before the error message.

Value

Either throws an error, or invisible TRUE.

Procedure

Procedure Object

Description

This object specifies alternating optimization procedure.

Active bindings

npar	(integer(1))	The length of the target argument.
partition	(character(1) or list())	Defines the parameter partition, and can be either <ul style="list-style-type: none"> • "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 alternating optimization process.
new_block_probability	(numeric(1))	Only relevant if partition = "random". The probability for a new parameter block when creating a random partitions. 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))
 Whether to print tracing details during the alternating optimization process.
- `minimize` (logical(1))
 Whether to minimize during the alternating optimization process. If FALSE, maximization is performed.
- `iteration_limit` (integer(1) or Inf)
 The maximum number of iterations through the parameter partition before the alternating optimization process is terminated. Can also be Inf for no iteration limit.
- `seconds_limit` (numeric(1))
 The time limit in seconds before the alternating optimization process is terminated. Can also be Inf for no time limit. Note that this stopping criteria 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 alternating optimization terminates if the absolute difference between the current function value and the one before `tolerance_history` iterations is smaller than `tolerance_value`.
 Can be 0 for no value threshold.
- `tolerance_parameter` (numeric(1))
 A non-negative tolerance value. The alternating optimization terminates if the distance between the current estimate and the before `tolerance_history` iterations 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 the current estimate and the one from the last iteration. If the distance is smaller than `tolerance_parameter`, the procedure 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.
- `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 alternating optimization procedure, 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 procedure: 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`)
- `seconds` is the overall computation time in seconds.
- `stopping_reason` is a message why the procedure has terminated.

Methods

Public methods:

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

Method `new()`: Creates a new object of this [R6](#) class.

Usage:

```
Procedure$new(
  npar = integer(),
  partition = "sequential",
  new_block_probability = 0.3,
  minimum_block_number = 2,
  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
)
```

Arguments:

`npar` (integer(1))

The (total) length of the target argument(s).

`partition` (character(1) or list())

Defines the parameter partition, and can be either

- "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 alternating optimization process.

`new_block_probability` (numeric(1))

Only relevant if `partition = "random"`. The probability for a new parameter block when creating a random partitions. 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))

Whether to print tracing details during the alternating optimization process.

`minimize` (logical(1))

Whether to minimize during the alternating optimization process. If FALSE, maximization is performed.

`iteration_limit` (integer(1) or Inf)

The maximum number of iterations through the parameter partition before the alternating optimization process is terminated. Can also be Inf for no iteration limit.

`seconds_limit` (numeric(1))

The time limit in seconds before the alternating optimization process is terminated. Can also be Inf for no time limit. Note that this stopping criteria 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 alternating optimization terminates if the absolute difference between the current function value and the one before `tolerance_history` iterations is smaller than `tolerance_value`.

Can be 0 for no value threshold.

`tolerance_parameter` (numeric(1))

A non-negative tolerance value. The alternating optimization terminates if the distance between the current estimate and the before `tolerance_history` iterations 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 the current estimate and the one from the last iteration. If the distance is smaller than `tolerance_parameter`, the procedure 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) $\sqrt{\text{sum}((x - y)^2)}$ is used.

`tolerance_history` (integer(1))

The number of iterations to look back to determine whether `tolerance_value` or `tolerance_parameter` has been reached.

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

Usage:

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

Arguments:

`message` (character(1))

The status message.

`message_type` (integer(1))

The type of the message, 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))

Whether to print tracing details during the alternating optimization process.

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

Usage:

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

Arguments:

`initial_parameter` (numeric())

The starting parameter values for the procedure.

`initial_value` (numeric(1))

The function value at the initial parameters.

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

Usage:

```
Procedure$update_details(
  value,
  parameter_block,
  seconds,
  error,
  block = self$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))
 Whether solving the sub-problem resulted in an error.

block (integer())
 The currently active parameter block, represented as parameter indices.

Method get_partition(): Get a parameter partition.

Usage:

```
Procedure$get_partition()
```

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

Usage:

```
Procedure$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) or 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 alternating optimization procedure.

Usage:

```
Procedure$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) or 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())

Whether to keep the column containing the information about the iteration in the output.

`keep_block_columns` (logical())

Whether to 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 alternating optimization procedure.

Usage:

```
Procedure$get_value_latest()
```

Method `get_value_best()`: Get the optimum function value in the alternating optimization procedure.

Usage:

```
Procedure$get_value_best()
```

Method `get_parameter()`: Get the parameter values in different steps of the alternating optimization procedure.

Usage:

```
Procedure$get_parameter(
  which_iteration = self$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) or 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())

Whether to keep the column containing the information about the iteration in the output.

`keep_block_columns (logical())`

Whether to 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 alternating optimization procedure.

Usage:

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

Arguments:

`parameter_type (character(1))`

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 alternating optimization procedure.

Usage:

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

Arguments:

`parameter_type (character(1))`

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 alternating optimization procedure.

Usage:

```
Procedure$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) or 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())

Whether to keep the column containing the information about the iteration in the output.

`keep_block_columns` (logical())

Whether to 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 alternating optimization procedure.

Usage:

`Procedure$get_seconds_total()`

Method `check_stopping()`: Checks if the alternating optimization procedure can be terminated.

Usage:

`Procedure$check_stopping()`

Index

ao, [2](#), [6](#)
ao_input_check, [6](#)

optim, [4](#)
Optimizer, [4](#)

Procedure, [7](#)

R6, [9](#)