Package: ao (via r-universe)

November 21, 2026

Title Alternating Optimization

Version 1.3.0

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

Imports checkmate, cli, oeli (>= 0.7.5), optimizeR (>= 1.2.1), R6, stats, utils

Suggests devtools, future.apply, ggplot2, knitr, progressr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Lennart Oelschläger [aut, cre] (ORCID: <https://orcid.org/0000-0001-5421-9313>), Siddhartha Chib [ctb]

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

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

Date/Publication 2026-11-21 16:00:02 UTC

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

RemoteRef HEAD

RemoteSha 2850525205507f4fe60fbf730d579b2547e29c00

Contents

ao 2

Index **8**

---

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` and `hessian`, if specified).

`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. Parameter indices can be members of any number of blocks, where parameters that are not a member of any block stay at their initial value.

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, each of which can be a single number (a common bound for all parameters) or a vector of the same length as `initial`.
The values in `initial` must respect the 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 refers to the time for solving the sub-problems and 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, which requires the `{future.apply}` package to be installed. 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, which requires the {progressr} package to be installed. 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 with the best function value in the AO process.
- `estimate_split` is `estimate` split by `target` (only when applicable).
- `value` is the function value at `estimate`.
- `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 total time in seconds for solving the sub-problems.
- `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("stats::nlm"),
  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
)
```

Index

ao, 2

optim, 5
Optimizer, 5