Package 'ino'

July 4, 2026

Title Initialization of Numerical Optimization

Version 1.2.1

Description Analysis of the initialization for numerical optimization of real-valued functions, particularly likelihood functions of statistical models. See <https://loelschlaeger.de/ino/> for more details.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports checkmate, cli, dplyr, future.apply, ggplot2, normalize (>= 0.1.2), oeli (>= 0.7.4), optimizeR (>= 1.2.1), portion (>= 0.1.2), R6, tidyr, utils

Suggests knitr, progressr, renv, rmarkdown, scales, TestFunctions, testthat

Config/testthat/edition 3

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

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

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2026-07-04 14:20:02 UTC

Contents

`autoplot.Nop` *Plotting methods*

Description

- `autoplot.Nop()` plots the objective function
- `autoplot.Nop_results()` plots boxplots of optimization results
- `autoplot.Nop_optima()` plots a bar chart of the found optima
- `autoplot.Nop_deviation()` plots deviations per dimension from a reference

Usage

```
## S3 method for class 'Nop'
autoplot(object, xlim = NULL, xlim2 = NULL, ...)

## S3 method for class 'Nop_optima'
autoplot(object, ...)

## S3 method for class 'Nop_deviation'
autoplot(object, jitter = TRUE, ...)

## S3 method for class 'Nop_results'
autoplot(
  object,
  which_element = "seconds",
  group_by = NULL,
  relative = FALSE,
  ...
)
```

Arguments

`object`	Depends on the method: • for `autoplot.Nop()`, a `Nop` object • for `autoplot.Nop_results()`, the value `Nop$results` • for `autoplot.Nop_optima()`, the value `Nop$optima` • for `autoplot.Nop_deviation()`, the value `Nop$deviation`
`xlim, xlim2`	`[numeric(2)]` Ranges for the first and second parameter to plot. If `NULL`, they are derived from the specified initial values in `object`.

`...`	Other arguments passed to specific methods.
`jitter`	[`logical(1)`] Apply jitter to the points?
`which_element`	[`character(1)`] A column name of `object` to plot.
`group_by`	[`character(1)`] Selects how the plot is grouped. Either: • `NULL` to not group, • `"optimization"` to group by optimization label, • `"optimizer"` to group by optimizer label.
`relative`	[`logical(1)`] Plot values relative to the overall median?

Value

A `ggplot` object.

`Nop` *Nop Object*

Description

A `Nop` object defines a numerical optimization problem.

Getting started

Step 1: Create a `Nop` object:

Call `object <- Nop$new(f, target, npar, ...)` where

- `f` is the objective function,
- `target` are the names of the target arguments,
- `npar` specifies the lengths of the target arguments,
- and `...` are additional arguments for `f`.

You can now evaluate the objective function via the `$evaluate()` method.

Step 2: Specify numerical optimizers:

Call `object$set_optimizer(<optimizer object>)`, where `<optimizer object>` is an object of class `optimizer`, which can be created via the `{optimizeR}` package (please refer to the package homepage for details).

For example,

- `optimizeR::Optimizer$new(which = "stats::nlm")` defines the `nlm` optimizer,
- `optimizeR::Optimizer$new(which = "stats::optim")` defines the `optim` optimizer.

Step 3: Select initial values:

Call initialization methods to define starting values for the optimization, for example:

- `object$initialize_fixed()` for fixed initial values,
- `object$initialize_random()` for random initial values,
- `object$initialize_continue()` for initial values based on parameter estimates from previous optimization runs.

Step 4: Optimization:

Call `object$optimize()` for the optimization.

Step 5: Analyze the results:

- `$results` returns a `tibble` of the optimization results,
- `$optima()` lists all identified optima,
- `$minimum` and `$maximum` return the best minimizer and maximizer

Input validation and errors

Public methods validate selector arguments, parameter bounds, and initial values before they are passed to optimizers. Invalid user input aborts with an error that identifies the affected argument. Errors raised by optimizers are captured in `$results` via the `error` and `error_message` columns when supported by the selected `Optimizer`.

Progress during optimization

Displaying progress during multiple optimization runs via the `{progressr}` package is supported. To get started, run

```
progressr::handlers(global = TRUE)
```

and see `handlers` for details.

Parallel optimization

Parallel computation of multiple optimization runs via the `{future}` package is supported. To get started, run one of

```
future::plan(future::multisession)
```

and see `plan` for details.

Active bindings

`initial_values` [`list()`, read-only]
: The currently defined initial values.

  Use the `initialize_*()` methods to add, transform, and reset values.

`results` [`tibble`, read-only]
: Optimization results with identifiers:

  - `".optimization_label"` (identifies the optimization run)
  - `".optimizer_label"` (identifies the optimizer)
  - `".direction"` (identifies the optimization direction)

- ".original" (identifies results obtained on the original problem)

The output has an associated autoplot method.

minimum [list(2), read-only]
: Best value and parameter across all (original) minimizations.

maximum [list(2), read-only]
: Best value and parameter across all (original) maximizations.

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

verbose [logical(1)]
: Print progress and details?

fresh_label [character(1), read-only]
: An optimization label that has not been used yet.

Methods

Public methods:

- Nop$new()
- Nop$fixed_argument()
- Nop$reduce_argument()
- Nop$standardize_argument()
- Nop$print()
- Nop$evaluate()
- Nop$set_optimizer()
- Nop$initialize_fixed()
- Nop$initialize_random()
- Nop$initialize_grid()
- Nop$initialize_custom()
- Nop$initialize_continue()
- Nop$initialize_filter()
- Nop$initialize_promising()
- Nop$initialize_transform()
- Nop$initialize_reset()
- Nop$optimize()
- Nop$optima()
- Nop$deviation()
- Nop$clone()

Nop$new(): Creates a new Nop object.

The output has an associated autoplot method.

Usage:

```
Nop$new(f, target = NULL, npar, gradient = NULL, hessian = NULL, ...)
```

Arguments:

`f` [`function`]
: A `function` to be optimized (the so-called objective function).
It is expected that

1. `f` has at least one `numeric` argument,
2. the return value of `f` is of the structure `numeric(1)`.

`target` [`character()`]
: The argument name(s) that get optimized (the so-called target arguments).
All target arguments must be `numeric`.
Can be `NULL` (default), then the first function argument is selected.

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

`gradient` [`function` | `NULL`]
: Optionally a `function` that returns the gradient of `f`.
The function call of `gradient` must be identical to `f`.
Ignored for optimizers that do not support user-supplied gradient.

`hessian` [`function` | `NULL`]
: Optionally a `function` that returns the Hessian of `f`.
The function call of `hessian` must be identical to `f`.
Ignored for optimizers that do not support user-supplied Hessian.

`...` Optionally additional function arguments passed to `f` (and `gradient` and `hessian`, if specified) that are fixed during the optimization.

`Nop$fixed_argument()`: Manages fixed arguments for the objective function.

Usage:

```
Nop$fixed_argument(action, ...)
```

Arguments:

`action` [`character(1)`]
: One of:

- `"set"` to set an argument,
- `"get"` to extract an argument value,
- `"remove"` to remove an argument,
- `"reset"` to reset an argument to its original value,
- `"modify"` to modify an argument value.

Note that `"set"` overrides an argument value, while `"modify"` preserves the original value, which can be recovered via `"reset"`.

`...` Additional parameters depending on `action`:

- named arguments if `action` = `"set"` or `"modify"`,
- a single argument name if `action` = `"get"`, `"remove"`, or `"reset"`.

`Nop$reduce_argument()`: Reduces a fixed argument for the objective function.

Usage:

```
Nop$reduce_argument(
  argument_name,
  proportion = 0.5,
```

```
  how = "random",
  centers = 2L,
  byrow = TRUE,
  ignore = integer()
)
```

Arguments:

`argument_name [character(1)]`
: The name of a fixed argument for the objective function.

`proportion, how, centers, byrow, ignore` Passed on to `portion`.

`Nop$standardize_argument()`: Standardizes a fixed argument for the objective function.

Usage:

```
Nop$standardize_argument(
  argument_name,
  center = TRUE,
  scale = TRUE,
  byrow = FALSE,
  ignore = integer(),
  jointly = list()
)
```

Arguments:

`argument_name [character(1)]`
: The name of a fixed argument for the objective function.

`center, scale, byrow, ignore, jointly` Passed on to `normalize`.

`Nop$print()`: Prints details of the `Nop` object.

Usage:

```
Nop$print(...)
```

Arguments:

`...` Currently not used.

`Nop$evaluate()`: Evaluates the objective function.

Usage:

```
Nop$evaluate(
  at = rep(0, sum(self$npar)),
  .gradient_as_attribute = FALSE,
  .hessian_as_attribute = FALSE
)
```

Arguments:

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

`.gradient_as_attribute, .hessian_as_attribute [logical(1)]`
: Add gradient and / or Hessian value as attributes?
If gradient and / or Hessian function is not specified, numerical approximation is used.

`Nop$set_optimizer()`: Specifies a numerical optimizer.

Usage:

```
Nop$set_optimizer(optimizer, optimizer_label = optimizer$label)
```

Arguments:

`optimizer` [`Optimizer`]
: An `Optimizer` object, which can be created via `Optimizer`.

`optimizer_label` [`character(1)`]
: A (unique) label for the optimizer.

`Nop$initialize_fixed()`: Defines fixed initial values for the optimization.

Usage:

```
Nop$initialize_fixed(at)
```

Arguments:

`at` [`numeric(sum(self$npar))` | `list()`]
: The fixed initial parameter vector.
It can also be a `list` of such vectors.

`Nop$initialize_random()`: Defines random initial values for the optimization.

Usage:

```
Nop$initialize_random(
  runs = 1L,
  sampler = function() stats::rnorm(sum(self$npar))
)
```

Arguments:

`runs` [`integer(1)`]
: The number of optimization runs.

`sampler` [`function`]
: A `function` without any arguments that returns a `numeric` vector of length `sum(self$npar)`.

`Nop$initialize_grid()`: Defines a grid of initial values for the optimization.

Usage:

```
Nop$initialize_grid(lower = 0, upper = 1, breaks = 3, jitter = FALSE, ...)
```

Arguments:

`lower`, `upper` [`numeric(1)` | `numeric(sum(self$npar))`]
: Lower and upper grid bounds for each parameter dimension.

`breaks` [`integer(1)` | `integer(sum(self$npar))`]
: The number of breaks for each parameter dimension.

`jitter` Add noise to the grid points for a random grid layout?

`...` Optional parameters passed to `jitter`.

`Nop$initialize_custom()`: Defines custom initial values for the optimization.

Usage:

```
Nop$initialize_custom(at, seconds = rep(0, length(at)), type = "custom")
```

Arguments:

`at` [`list()`]
: A `list` of initial parameter vectors.

`seconds` [`numeric(length(at))`]
: The number of seconds it took to obtain each initial value in `at`, which is added to the overall optimization time.

`type` [`character(1)`]
: The type of the initial values.

`Nop$initialize_continue()`: Defines initial values based on results from previous optimizations.

Usage:

```
Nop$initialize_continue(optimization_label)
```

Arguments:

`optimization_label` [`character(1)`]
: Label of optimization runs from which to select.

`Nop$initialize_filter()`: Filters initial values from the defined initial values.

Usage:

```
Nop$initialize_filter(condition)
```

Arguments:

`condition` [`character(1)`]
: Defines the condition on which the initial values are filtered, one of:
  - `"gradient_negative"` for points where the gradient is negative,
  - `"gradient_positive"` for points where the gradient is positive,
  - `"hessian_negative"` for points where the Hessian is negative definite,
  - `"hessian_positive"` for points where the Hessian is positive definite.

`Nop$initialize_promising()`: Selects promising initial values from the defined initial values.

Usage:

```
Nop$initialize_promising(proportion, condition)
```

Arguments:

`proportion` [`numeric(1)`]
: The proportion of selected from the defined initial values.

`condition` [`character(1)`]
: Defines the condition on which the initial values are selected, one of:
  - `"value_small"` for points where the function value is smallest,
  - `"value_large"` for points where the function value is largest,
  - `"gradient_small"` for points where the gradient norm is smallest,
  - `"gradient_large"` for points where the gradient norm is largest,
  - `"condition_small"` for points where the Hessian condition is smallest,
  - `"condition_large"` for points where the Hessian condition is largest.

`Nop$initialize_transform()`: Transforms the currently defined initial values.

Usage:

```
Nop$initialize_transform(transformer = function(x) x)
```

Arguments:

`transformer` [`function`]
: A `function` that receives and returns a `numeric()` of length `sum(self$npar)`.

`Nop$initialize_reset()`: Resets the currently defined initial values.

Usage:

```
Nop$initialize_reset()
```

`Nop$optimize()`: Optimizes the target function.

Usage:

```
Nop$optimize(
  optimization_label = self$fresh_label,
  which_optimizer = "all",
  which_direction = "min",
  lower = NULL,
  upper = NULL,
  seconds = Inf,
  hide_warnings = TRUE,
  reset_initial_afterwards = TRUE
)
```

Arguments:

`optimization_label` [`character(1)`]
: A label for the optimization to distinguish optimization runs.
Setting a label is useful when using the `$initialize_continue()` method.

`which_optimizer` [`character()` | `integer()`]
: Selects numerical optimizers. Either:
- `"all"` for all specified optimizers,
- specific optimizer labels,
- specified optimizer ids as defined in the `print()` output.

`which_direction` [`character()`]
: Selects the direction of optimization. One or both of:
- `"min"` for minimization,
- `"max"` for maximization.

`lower`, `upper` [`numeric()` | `NULL`]
: Optionally lower and upper parameter bounds.
Ignored for optimizers that do not support parameter bounds.

`seconds` [`numeric(1)`]
: A time limit in seconds.
Optimization is interrupted prematurely if `seconds` is exceeded.
Note the limitations documented in `setTimeLimit`.

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

`reset_initial_afterwards [logical(1)]`
Reset the initial values after the optimization?

Details: Supports:

- Parallel computation of multiple optimization runs via `{future}`
- Progress messages via `{progressr}`

`Nop$optima()`: Lists all identified optima.

The output has an associated `autoplot` method.

Usage:

```
Nop$optima(
  which_direction = "min",
  only_original = TRUE,
  group_by = NULL,
  sort_by_value = FALSE,
  digits = getOption("digits", default = 7)
)
```

Arguments:

`which_direction [character()]`
Selects the direction of optimization. One or both of:

- `"min"` for minimization,
- `"max"` for maximization.

`only_original [logical(1)]`
Include only optima obtained on the original problem?

`group_by [character(1)]`
Selects how the output is grouped. Either:

- `NULL` to not group,
- `"optimization"` to group by optimization label,
- `"optimizer"` to group by optimizer label.

`sort_by_value [logical(1)]`
Sort by value? Else, sort by frequency.

`digits [integer(1)]`
The number of decimal places.

`Nop$deviation()`: Compute deviations with respect to a reference parameter.

The output has an associated `autoplot` method.

Usage:

```
Nop$deviation(
  reference = rep(0, sum(self$npar)),
  which_element = "initial",
  which_direction = "min",
  which_optimizer = "all",
  only_original = TRUE,
  parameter_labels = paste0("x", seq_len(sum(self$npar)))
)
```

Arguments:

`reference` [`numeric()`]
: The reference vector of length `sum(self$npar)`.

`which_element` [`character(1)`]
: Either
  - `"initial"` for deviations with respect to the initial values, or
  - `"parameter"` for deviations with respect to the estimated parameters.

`which_direction` [`character()`]
: Selects the direction of optimization. One or both of:
  - `"min"` for minimization,
  - `"max"` for maximization.

`which_optimizer` [`character()` | `integer()`]
: Selects numerical optimizers. Either:
  - `"all"` for all specified optimizers,
  - specific optimizer labels,
  - specified optimizer ids as defined in the `print()` output.

`only_original` [`logical(1)`]
: Include only optima obtained on the original problem?

`parameter_labels` [`character()`]
: Labels for the parameters of length `sum(self$npar)`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### define objective function, optimizer and initial values
Nop_ackley <- Nop$new(f = TestFunctions::TF_ackley, npar = 2)$
  set_optimizer(optimizeR::Optimizer$new(which = "stats::nlm"))$
  initialize_random(runs = 20)

### plot function surface and initial values
Nop_ackley |> ggplot2::autoplot()

### minimize objective function
Nop_ackley$optimize(which_direction = "min")

### show optima
Nop_ackley$optima(digits = 0)

### show best value and parameter across all minimizations
Nop_ackley$minimum
```

Index