Package 'optimizr'

May 9, 2026

Type Package

Title Further Numerical Optimization Algorithms

Version 1.0.1

Description A collection of numerical optimization algorithms.
One is a simple implementation of
the primitive grid search algorithm, the other is an extension of the
simulated annealing algorithm that can take custom boundaries into
account. The methodology for this bounded simulated annealing
algorithm is due to Haario and Saksman (1991), <doi:10.2307/1427681>.

Imports stats, progressr, future.apply

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0), doFuture

Config/testthat/edition 3

NeedsCompilation no

Author Lukas D Sauer [aut, cre]

Maintainer Lukas D Sauer <sauer@imbi.uni-heidelberg.de>

Repository CRAN

Date/Publication 2026-03-19 13:20:02 UTC

Contents

algorithm *Optimization algorithm*

Description

Optimization algorithm

Arguments

fn	The function to be optimized, taking a numeric vector of parameters and returning a numeric value.
lower	A numeric vector, lower boundary of the parameter vector.
upper	A numeric vector, upper boundary of the parameter vector.
control	A list of further control parameters for the algorithm.

Value

A list containing par, the parameter combination of the optimum, value, the optimal function value, and trace, a data.frame of parameter values visited during the algorithm's run.

genptry *Generate a candidate point in the neighborhood*

Description

Using a Gaussian kernel of standard deviation *scale* around the current point, this function generates a new point nearby.

Usage

```
genptry(p, scale)
```

Arguments

p	a numeric vector
scale	a real number, the radius of the Gaussian

Value

a numeric vector

get_par_names *Get names for the parameter vector*

Description

Get names for the parameter vector

Usage

```
get_par_names(x)
```

Arguments

x	a named object

Value

a character vector, either the names of x or strings of the form "Var1", "Var2", etc.

gridsearch *Grid Search Algorithm*

Description

The grid search algorithm searches the parameter space of a function fn for the optimal parameter values on a pre-defined grid.

Usage

```
gridsearch(
  fn,
  lower = NULL,
  upper = NULL,
  step = NULL,
  axes = mapply(seq, from = lower, to = upper, by = step, SIMPLIFY = FALSE),
  grid = expand.grid(axes),
  control = NULL
)
```

Arguments

fn	The function to be optimized, taking a numeric vector of parameters and returning a numeric value.
lower	A numeric vector, lower boundary of the parameter vector.
upper	A numeric vector, upper boundary of the parameter vector.
step	A numeric vector of step widths for each dimension of the parameter space.
axes	A list of numeric vectors, defining the axis values of the grid for each dimension of the parameter space.
grid	A data frame of all parameter combinations to be visited.
control	A list of further control parameters for the algorithm.

Details

The grid can be defined by either

1. lower and upper bounds of the parameter space together with a step width for each dimension,
2. axis vectors for each dimension of the parameter space, or
3. a "grid" containing every parameter combination to be visited.

Value

A list containing `par`, the parameter combination of the optimum, `value`, the optimal function value, and `trace`, a `data.frame` of parameter values visited during the algorithm's run.

Control parameters

The list `control` may contain the following objects:

- `fnscale`: A numeric, if it is non-negative or NULL, the algorithm searches for the minimum, if it is negative, it searches for the maximum.
- `REPORT`: An integer, if it is `NA_integer_` or negative, no trace is reported. If `>=0`, a trace is reported. If `>0`, status updates are sent to a `progressr` handler every step. By default, `REPORT = 0`,
- `use_future`: A logical, if `TRUE` the grid is searched using `future.apply::future_apply`, if `FALSE`, it is searched using apply. By default, `use_future = TRUE`. Note that for actually using parallelization, you still need to `future::plan()` the session.
- `future_apply_options`: A list of further options to be supplied to the `future_apply()` call. This is only used if `use_future` is `TRUE`. For example, use `future_apply_options = list((future.globals = structure(TRUE, add = c("globalvar"))))` will bring a global variable `globalvar` into the grid search call.

Examples

```
fn <- function(vec){
  return((-1)*dnorm(vec[["x"]], mean = 3.9)*dnorm(vec[["y"]], mean = 3.02))
}

# Define grid using lower and upper bounds and step widths
gridsearch(fn,
           lower = c(x = -10, y = -10),
           upper = c(x = 10, y = 10),
           step = c(x = 0.5, y = 0.5))
# Define grid using axes
gridsearch(fn,
           axes = list(x = (-10:10), y = (-10:10)))
# Custom grid, e.g. only the diagonal
grid <- data.frame(x = (-10:10), y = (-10:10))
gridsearch(fn,
           grid = grid)
# Diagnostics with progress bar
# Attention: Progress bar impedes performance!

progressr::handlers(global = TRUE)
fn <- function(vec){
  Sys.sleep(0.001)
  return(c(result =
  (-1)*dnorm(vec[["x"]], mean = 3.9)*dnorm(vec[["y"]], mean = 3.02)))
}
gridsearch(fn,
           lower = c(x = -10, y = -10),
           upper = c(x = 10, y = 10),
           step = c(x = 0.5, y = 0.5),
           control = list(REPORT = 1))

# Parallelized grid search using doFuture
library(doFuture)
plan(multisession)
gridsearch(fn,
           lower = c(x = -10, y = -10),
           upper = c(x = 10, y = 10),
           step = c(x = 0.5, y = 0.5))
```

---

phi *Transform an interval to the unit interval*

Description

A monotonously increasing bijection on the real line that maps the interval $[\text{lower}, \text{upper}]$ to the unit interval $[0, 1]$.

Usage

```
phi(x, lower, upper)
```

Arguments

x	any real number
lower	a real number, the lower bound of the interval
upper	a real number, the upper bound of the interval

Value

a real number

---

phiinv *Transform the unit interval to another interval*

Description

A monotonously increasing bijection on the real line that maps the unit interval $[0, 1]$ to the interval $[\text{lower}, \text{upper}]$.

Usage

```
phiinv(x, lower, upper)
```

Arguments

x	any real number
lower	a real number, the lower bound of the interval
upper	a real number, the upper bound of the interval

Value

a real number

simann *Simulated Annealing Algorithm*

Description

An implementation of the simulated annealing algorithm that is very similar to `stats::optim(method = "SANN")` with a few differences explained below.

Usage

```
simann(par, fn, lower = NULL, upper = NULL, control)
```

Arguments

`par`	A numeric vector, start values for the algorithm.
`fn`	The function to be optimized, taking a numeric vector of parameters and returning a numeric value.
`lower`	A numeric vector, lower boundary of the parameter vector.
`upper`	A numeric vector, upper boundary of the parameter vector.
`control`	A list of further control parameters for the algorithm.

Details

This is almost a precise re-write of the simulated annealing optimization algorithm `stats::optim(method = "SANN")` as implemented in the file `r-source/src/appl/optim.c`. In particular, temperature is updated step by step with respect to the formula

$$T^{(i)} = T_{\text{start}} / \log(((i-1)\%/\%t_{\max}) * t_{\max} + \exp(1))$$

and new candidate points are generated using a Gaussian kernel

$$p_j^{(i)} = p^{(i-1)} + \frac{T^{(i)}}{T_{\text{start}}} \cdot \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0,1)$.

However, there are three main differences:

- This function is currently implemented completely in R, for easier debugging.
- This function returns a trace of all visited values as a `data.frame`.
- This function allows for bounded parameter spaces by using the methodology from chapter 6 of (Haario and Saksman 1991).

The adaption for the case of bounded parameter spaces is as follows: W.l.o.g. assume that boundaries are $[0,1]^n$. Let p the current point and p^{try} the candidate generated by the Gauss kernel. Suppose that i-th component is not in boundary, i.e. $p_i^{\text{try}} \notin [0,1]$. Then division by 2 with remainder gives us $p_i^{\text{try}} \mod 2 \in [0,2)$, when identifying $\mathbb{R}/2\mathbb{Z} \cong [0,2)$ as sets. Then either $p_i^{\text{try}} \mod 2 \in [0,1]$ or $2 - p_i^{\text{try}} \mod 2 \in [0,1]$. Then, use this updated component of the candidate.

Value

A list containing `par`, the parameter combination of the optimum, `value`, the optimal function value, and `trace`, a `data.frame` of parameter values visited during the algorithm's run.

Control parameters

The list `control` may contain the following objects:

- `maxit`: The maximal number of temperature steps, defaults to `10000`,
- `temp`: The initial temperature, defaults to `10`.
- `tmax`: The maximal number of function evaluations at each temperature step, defaults to `10`.
- `parscale`: A scaling factor by which the parameter vector is scaled before plugging it into `fn`.
- `fnscale`: A scaling factor by which the value of `fn` is divided during optimization. If it is negative, it turns the problem into a maximization problem.
- `REPORT`: An integer determining that the algorithms current state is reported every `REPORT` steps. If `NA_integer_`, no trace is reported. Defaults to 100. Also determines how often the progress bar is updated.

References

Haario H, Saksman E. Simulated Annealing Process in General State Space. Advances in Applied Probability, Vol. 23, No. 4 (Dec., 1991), pp. 866-893, doi:10.2307/1427681.

Examples

```
# Minimization problem:
# "wild" function from the optim() examples, global minimum at about -15.81515
fw <- function (x)
  (10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80)
plot(fw, -50, 50, n = 1000, main = "The 'wild function' from the optim() examples")
res <- simann(par = 50, fn = fw,
              control = list(maxit = 20000,
                             temp = 20,
                             parscale = 20))
res$par
res$value

plot(res$trace$it, res$trace$fn)
plot(res$trace$it, res$trace$par1)

# Bounded maximization problem:
# The "wild" function has a local maximum around 46
resmx <- simann(par = 3, fn = fw,
                lower = 0, upper = 46,
                control = list(maxit = 20000,
                               temp = 20,
                               fnscale = -1,
                               parscale = 20))
resmx$par
```

```
resmx$value

# Diagnostics with progress bar and frequent trace reports
# Attention: Both progress bar and reports impede performance!

progressr::handlers(global = TRUE)
resdiag <- simann(par = 50, fn = fw,
                  control = list(maxit = 20000,
                                 temp = 20,
                                 parscale = 20,
                                 REPORT = 1))
```

Index