

# Package ‘optimizr’

March 19, 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](https://doi.org/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](mailto:sauer@imbi.uni-heidelberg.de)>

**Repository** CRAN

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

## Contents

algorithm . . . . .	2
genptry . . . . .	2
get_par_names . . . . .	3
gridsearch . . . . .	3
phi . . . . .	5
phiinv . . . . .	6
simann . . . . .	7

<b>Index</b>	<b>10</b>
--------------	-----------

---

algorithm	<i>Optimization algorithm</i>
-----------	-------------------------------

---

**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	<i>Generate a candidate point in the neighborhood</i>
---------	---

---

**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	<i>Get names for the parameter vector</i>
---------------	---

---

**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	<i>Grid Search Algorithm</i>
------------	------------------------------

---

**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**

<code>fn</code>	The function to be optimized, taking a numeric vector of parameters and returning a numeric value.
<code>lower</code>	A numeric vector, lower boundary of the parameter vector.
<code>upper</code>	A numeric vector, upper boundary of the parameter vector.
<code>step</code>	A numeric vector of step widths for each dimension of the parameter space.
<code>axes</code>	A list of numeric vectors, defining the axis values of the grid for each dimension of the parameter space.
<code>grid</code>	A data frame of all parameter combinations to be visited.
<code>control</code>	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  $\geq 0$ , 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::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))

```

**Description**

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

**Usage**

`phi(x, lower, upper)`

**Arguments**

<code>x</code>	any real number
<code>lower</code>	a real number, the lower bound of the interval
<code>upper</code>	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**

<code>x</code>	any real number
<code>lower</code>	a real number, the lower bound of the interval
<code>upper</code>	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

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

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

and new candidate points are generated using a Gaussian kernel

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

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  $p^{\text{try}}$  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  $p_i^{\text{try}} \bmod 2 \in [0, 2)$ , when identifying  $\mathbb{R}/2\mathbb{Z} \cong [0, 2)$  as sets. Then either  $p_i^{\text{try}} \bmod 2 \in [0, 1]$  or  $2 - p_i^{\text{try}} \bmod 2 \in [0, 1]$ . Then, use this updated component of the candidate.



```
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

algorithm, 2

genptry, 2

get\_par\_names, 3

gridsearch, 3

phi, 5

phiinv, 6

simann, 7