Package: vntrs (via r-universe)

October 10, 2026

Title Variable Neighborhood Trust Region Search

Version 0.3.0

Description Implements a variable neighborhood trust region search (VNTRS) algorithm for nonlinear global optimization, based on Bierlaire et al. (2009) ``A Heuristic for Nonlinear Global Optimization'' <doi:10.1287/ijoc.1090.0343>. The method combines neighborhood exploration with a trust-region framework to search the solution space efficiently. It can terminate a local search early when the iterates converge toward a previously visited local optimum or when further improvement within the current region is unlikely. The algorithm can also be used to identify multiple local optima.

URL https://loelschlaeger.de/vntrs/

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

License GPL-3

Encoding UTF-8

Imports checkmate, oeli (>= 0.7.5), Rcpp

LinkingTo Rcpp, RcppArmadillo

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

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

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

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

Date/Publication 2026-09-03 17:30:02 UTC

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

RemoteRef HEAD

RemoteSha bf4915b5249012c913abd2be6c6ea2c58811e26f

Contents

vntrs 2

Index **7**

`vntrs` *Variable neighborhood trust region search*

Description

Run the variable neighborhood trust region search algorithm. Features:

- expanding-neighborhood exploration,
- local trust-region optimization,
- optional parameter bounds,
- analytical derivatives are not required,
- early stopping of inferior or previously explored search paths,
- collection of multiple local optima,
- optional time and function-evaluation budgets.

Usage

```
vntrs(
  f,
  npar,
  lower = NULL,
  upper = NULL,
  minimize = TRUE,
  collect_all = FALSE,
  init_runs = 5L,
  init_min = -1,
  init_max = 1,
  init_iterlim = 20L,
  neighborhoods = 5L,
  neighbors = 5L,
  beta = 0.05,
  iterlim = 100L,
  scale = c("relative", "absolute"),
  identical_tolerance = 0.001,
  known_optimum_radius = if (scale == "relative") 0.1 else 1,
  inferior_tolerance = if (scale == "relative") 1e-06 else 3,
  interruption_gradient_tolerance = 0.001,
  gradient_tolerance = 1e-06,
  time_limit = NULL,
  evaluation_limit = NULL,
  quiet = TRUE
)
```

Arguments

`f` [function]
A function that accepts a numeric parameter vector and returns either

- a numeric objective value, or
- a list with value and optional gradient and hessian components.

Missing derivatives are approximated by finite differences.

Recommendation: Provide analytical derivatives when available.

`npar` [integer(1)]
The length of the parameter vector passed to f.

`lower, upper` [numeric(npar) | NULL]
Lower and upper bounds for the parameters. NULL represents no bound; use -Inf or Inf for individual unbounded parameters. Starting points and neighborhood points are restricted to these bounds.

`minimize` [logical(1)]
If TRUE, minimize f; if FALSE, maximize it.

`collect_all` [logical(1)]
If TRUE, do not interrupt a local search solely because it appears to approach an optimum inferior to the best one found so far. This increases the chance of returning multiple local optima, but usually requires more function evaluations. Duplicate optima are still merged.

Recommendation: Use FALSE when only the best solution matters and TRUE when local optima are themselves of interest.

`init_runs` [integer(1)]
Number of random starting points used to initialize the search. A short local search is run from each point, and the best result seeds the neighborhood search. More runs improve exploration but increase cost approximately linearly.

Recommendation: Start with 5; increase it for strongly multimodal problems.

`init_min, init_max`
[numeric(1)]
For a parameter without two finite bounds lower and upper, an initial value is uniformly sampled between init_min and init_max and then restricted by any one-sided bound.

If a parameter has finite lower and upper bounds, it is instead sampled uniformly across that interval.

Recommendation: Use -1 and 1 for reasonably scaled unbounded parameters; otherwise choose a range containing plausible solutions.

`init_iterlim` [integer(1)]
Maximum number of trust-region iterations for each local search started from one of the init_runs random points.

Recommendation: Start with 20; increase it when initialization searches stop before reaching an optimum.

`neighborhoods` [integer(1)]
Number of neighborhoods tried around the best solution found so far. Neighborhood k has scale 1.5^{k-1}. The algorithm tries at most neighborhoods sizes

without improvement. Whenever a better solution is found, it starts again with the first neighborhood.

Recommendation: Start with `5`; use more for broader global exploration at additional computational cost.

`neighbors` [`integer(1)`]
Number of new starting points generated for each neighborhood. Each of these trial points is a parameter vector obtained by moving away from the best solution found so far in a sampled direction. A separate local trust-region search starts from every trial point. Trying more points makes it more likely to reach different regions of the parameter space, but also increases the number of local searches.

Recommendation: Start with `5`; increase it when broader exploration is worth the additional computation time.

`beta` [`numeric(1)`]
Controls which directions are more likely to be selected when generating trial points. The local model describes the curvature around the best solution through eigenvectors (directions) and eigenvalues (strength of curvature). The sampling weight of a direction is proportional to `exp(beta * lambda / d)`, where `lambda` is its eigenvalue and `d` is the current neighborhood scale. With `beta = 0`, all positive and negative eigenvector directions are equally likely. Larger values concentrate the search on directions in which the objective bends more strongly.

This is the weighting rule for β in Bierlaire et al. (2009), Section 2.3, Equation (17).

Recommendation: Start with `0.05`; use `0` to disable the curvature preference, for example when the objective is noisy.

`iterlim` [`integer(1)`]
Maximum number of trust-region iterations for each local search during neighborhood exploration.

Recommendation: Start with `100`; increase it if otherwise promising local searches frequently stop before convergence.

`scale` [`character(1)`]
Determines whether tolerances are interpreted on a `"relative"` or `"absolute"` scale. Relative scaling reduces dependence on the units of the parameters and objective.

Recommendation: Use `"relative"` unless all quantities have meaningful common units.

`identical_tolerance`
[`numeric(1)`]
Parameter tolerance for deciding whether two solutions represent the same optimum. With `scale = "relative"`, each component difference is divided by $\max(1, |x_i|, |y_i|)$. The points are treated as the same optimum when the largest component difference does not exceed `identical_tolerance`. The later point is then not stored as a separate optimum, and a local search approaching such a point may be stopped early. This rule also applies when `collect_all = TRUE`.

Recommendation: Start with `1e-3`; increase it when repeated searches return slightly different parameter vectors for what is clearly the same optimum; decrease it when genuinely different optima may lie close together.

known_optimum_radius

[numeric(1)]

Euclidean distance used to decide whether the current point is near a previously identified optimum. With `scale = "relative"`, each parameter difference between points x and y is divided by $\max(1, |x_i|, |y_i|)$.

Recommendation: Use the scale-dependent default (`0.1` for relative distances and `1` for absolute distances); reduce it when nearby optima should be distinguished; increase it when a wider region around known optima should be treated as already explored.

inferior_tolerance

[numeric(1)]

Controls when the algorithm gives up early on a local search that is unlikely to improve the best known solution. With `scale = "absolute"`, it is measured in objective-function units. With `scale = "relative"`, the objective difference is divided by $\max(1, |f_{best}|)$, where f_{best} is the best known value. It is used only after at least one optimum has been found. The search is stopped in either of the following situations:

- Its value is worse than the best known value by more than `inferior_tolerance`.
- It is within `known_optimum_radius` of a known optimum but does not improve the best known value by more than `inferior_tolerance`.

Setting `collect_all = TRUE` disables both objective-value rules above, allowing searches toward inferior local optima to continue. Searches that approach an already identified optimum can still be stopped using `identical_tolerance`.

Recommendation: Use the scale-dependent default (`1e-6` for relative objective differences and `3` for absolute differences); reduce the value to stop inferior searches earlier; increase it to make interruption more conservative.

interruption_gradient_tolerance

[numeric(1)]

Non-negative threshold for the relative or absolute (depending on `scale`) (projected; to deal with parameter bounds) gradient norm used by the premature-interruption rule. Once an optimum is known, objective-value checks controlled by `inferior_tolerance` are considered only when the current gradient norm is no greater than this value. This indicates that the local search is approaching a stationary point and is therefore unlikely to leave the current region.

Setting `collect_all = TRUE` disables the interruption rules, so `interruption_gradient_tolerance` then has no effect.

Recommendation: Use `1e-3`, the (absolute) threshold used for premature interruption in Bierlaire et al. (2009). Use a smaller value to make premature interruption more conservative.

gradient_tolerance

[numeric(1)]

First-order convergence tolerance for each local trust-region search. A local search satisfies the first-order convergence condition when the relative or absolute (depending on `scale`) (projected; to deal with parameter bounds) gradient is no greater than `gradient_tolerance`. Additional curvature checks are used to avoid accepting saddle points.

Recommendation: Start with `1e-6`; increase it for noisy objectives or when faster, less precise local searches are sufficient; decrease it for smooth objectives with reliable derivatives when greater local accuracy is required.

`time_limit` [`numeric(1) | NULL`]
Optional approximate time limit in seconds. It is checked between local searches, so a running objective evaluation or local search is not interrupted and the elapsed time may exceed the limit.

`evaluation_limit` [`integer(1) | NULL`]
Optional maximum number of calls to `f`. The limit includes evaluations used to validate the objective and to approximate missing derivatives. It is enforced before every call, so the specified number is not exceeded. If the limit is reached, the algorithm returns optima completed before that point, or `NULL` if none have been found.

`quiet` [`logical(1)`]
If `TRUE`, suppress progress messages. Warnings are still emitted.

Value

A `data.frame` summarizing the identified optima, ordered from best to worst by objective value, or `NULL` if none could be determined.

References

Bierlaire et al. (2009) "A Heuristic for Nonlinear Global Optimization" doi:10.1287/ijoc.1090.0343.

Examples

```
### Example 1: Rosenbrock function; one global minimum; no local minima
set.seed(1)
rosenbrock <- function(x) 100 * (x[2] - x[1]^2)^2 + (1 - x[1])^2
vntrs(f = rosenbrock, npar = 2)

### Example 2: Six-hump camel function; two global minima; four local minima
set.seed(1)
camel <- function(x) {
  (4 - 2.1*x[1]^2 + x[1]^4/3) * x[1]^2 + x[1]*x[2] + (-4 + 4*x[2]^2) * x[2]^2
}
vntrs(
  f = camel, npar = 2,
  lower = c(-3, -2), upper = c(3, 2), # search bounds
  collect_all = TRUE,                 # also collect local optima
  neighborhoods = 10                  # number of neighborhoods
)
```

Index

vntrs, 2