Package: zoomgrid (via r-universe)

May 24, 2026

Type Package

Title Grid Search Algorithm with a Zoom

Version 1.1.0

Description Implements a grid search algorithm with an adaptive zooming strategy for global optimisation problems with multiple local optima. The method recursively refines the search region around promising grid points, providing reliable initial values for subsequent optimisation procedures. The algorithm is computationally efficient in moderate- to high-dimensional settings.

Depends R (>= 4.0.0)

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/yukai-yang/zoomgrid

BugReports https://github.com/yukai-yang/zoomgrid/issues

Imports cli

Suggests future, future.apply, knitr, rmarkdown

VignetteBuilder knitr

Repository https://yukai-yang.r-universe.dev

Date/Publication 2026-02-28 21:20:18 UTC

RemoteUrl https://github.com/yukai-yang/zoomgrid

RemoteRef HEAD

RemoteSha fdb39711cd274c2d6fe5afdc15b71055b4ca0134

Contents

`build_grid` *Build the grid for the grid search algorithm with a zoom.*

Description

This function builds the grid for the grid search algorithm with a zoom.

Usage

```
build_grid(...)
```

Arguments

`...`	a sequence of vectors or lists containing the information about the grid to be built, see Usage and Details.

Details

The argument `...` is a sequence of vectors or lists containing the information about the grid to be built. Each element in the sequence is either a vector or a list taking one of the following forms

- `x`, if `x` is already a sequence of the grid points for the corresponding argument.

- `c(from=, to=, by=)`

- `c(from=, to=, length=)`

- `list(from=, to=, by=)`

- `list(from=, to=, length=)`

where

- `from`: the min of the argument of the target function

- `to`: the max of the argument of the target function

- `by`: the increment of the sequence

- `length`: desired length.

There are many different ways to organize the points on the grid for certain argument of the target function, the user can make them freely and input directly by `build_grid(x, ...)`. Notice that `x` does not need to be increasing, as the function will sort it. The design that `x` does not need to be increasing makes it convenient for the user to interpolate more points at some region without considering to sort it all the time.

When `by` is provided, the `length` will be ignored. So if the user want to specify the `length`, please do not use `by`.

The order of the sequence `...` matters as it represents the order of the corresponding arguments of the target function to be optimized.

Value

a new object of the class GRID with the grid ready for the grid search with a zoom.

The object contains the following components:

grid	the grid
size	number of points in the grid
npar	number of arguments or parameters

Author(s)

Yukai Yang, <yukai.yang@statistik.uu.se>

See Also

grid_search_check, grid_search

Examples

```
vx = 1:5
build_grid(vx, c(from=1, to=2, by=.2), list(from=3, to=4, length=5))
```

---

grid_search *Carry out the grid search algorithm with a zoom.*

Description

This function carries out the grid search algorithm with a zoom.

Usage

```
grid_search(
  FUN,
  grid,
  MoreArgs = NULL,
  zoom = 0,
  decay = 0.5,
  num = 1,
  parallel = FALSE,
  cores = NULL,
  silent = TRUE
)
```

Arguments

FUN	the target function to be minimized.
grid	an object of class GRID created by build_grid.
MoreArgs	a named list of additional arguments to FUN, see mapply.
zoom	number of (additional) zoom-in layers, 0 by default.
decay	a number in $(0, 1)$ controlling the decay of subgrid sizes.
num	number of points to return at each grid search, 1 by default.
parallel	a logical; if TRUE, parallel computation is used.
cores	an integer specifying the requested number of workers when parallel = TRUE. If NULL, the function uses 2 workers by default (subject to future::availableCores()).
silent	a logical indicating whether progress information is printed.

Details

The target function FUN to be minimized is a scalar real-valued function. Maximization can be achieved by multiplying -1 to the original function and then passing the new function to FUN.

The grid must be created by build_grid.

Any other invariant arguments to FUN can be specified in MoreArgs using a named list, see mapply.

The common grid search first builds a grid within a bounded region, evaluates FUN at each grid point, and returns the num points that yield the smallest values.

zoom = 0 implies no zoom-in (a single grid search). Any integer zoom > 0 applies additional zoom-in layers. With zoom > 0, the algorithm performs

$$n^0 + n^1 + n^2 + \cdots + n^z$$

grid searches, where n is num and z is zoom. Consequently, the total number of returned points is

$$n^1 + n^2 + n^3 + \cdots + n^{z+1}$$

.

At each zoom-in layer, the algorithm builds subgrids around the best points found in the previous layer. To limit the computational burden, the subgrid size is reduced by the decay rate decay. For each parameter, the number of points in the subgrid is max(Int(decay * N), 3), where N is the number of points in the original grid for that parameter.

Parallel computation can be enabled by setting parallel = TRUE. In that case, the function uses the **future** framework with future::multisession (cross-platform). The number of workers is determined as follows:

1. Let n be the value returned by future::availableCores().
2. Let m be the user input cores. If cores = NULL, set $m = 2$.
3. The number of workers is $\min(m, n)$.

If parallel = TRUE, the packages **future** and **future.apply** must be installed.

The boolean silent controls whether progress information is printed to the console.

Value

a list with components:

par	the approximate global minimizer
points	all candidate points found by the grid search with zoom-in layers

Author(s)

Yukai Yang, <yukai.yang@statistik.uu.se>

See Also

build_grid, grid_search_check

Examples

```
# Rastrigin function
ndim = 2
nA = 10
Rastrigin <- function(vx) nA * ndim + sum(vx * vx - nA * cos(2 * pi * vx))

# Build a grid
bin = c(from = -5.12, to = 5.12, by = .5)
grid = build_grid(bin, bin)

# Serial computation
ret0 = grid_search(Rastrigin, grid, silent = FALSE)
ret0$par


# Finer grid
bin = c(from = -5.12, to = 5.12, by = .1)
grid = build_grid(bin, bin)

# Serial computation
ret1 = grid_search(Rastrigin, grid, silent = FALSE)
ret1$par

# Parallel computation (requires future and future.apply)
ret2 = grid_search(Rastrigin, grid, num = 2, parallel = TRUE, cores = 2, silent = FALSE)
ret2$par

# Grid search with zoom-in layers
ret3 = grid_search(Rastrigin, grid, zoom = 2, num = 2, parallel = TRUE, cores = 2, silent = FALSE)
ret3$par
```

grid_search_check *Check the time consumed by running the grid search algorithm with a zoom.*

Description

This function provides a quick runtime estimate for grid_search under the same settings. It performs two short pilot runs on smaller grids (with zoom = 0) and extrapolates the expected time for the full grid and the requested number of zoom-in layers.

Usage

```
grid_search_check(
  FUN,
  grid,
  MoreArgs = NULL,
  zoom = 0,
  decay = 0.5,
  num = 1,
  parallel = FALSE,
  cores = NULL,
  silent = TRUE
)
```

Arguments

FUN	the target function to be minimized.
grid	an object of class GRID created by build_grid.
MoreArgs	a named list of additional arguments to FUN, see mapply.
zoom	number of (additional) zoom-in layers, 0 by default.
decay	a number in $(0, 1)$ controlling the decay of subgrid sizes.
num	number of points to return at each grid search, 1 by default.
parallel	a logical; if TRUE, parallel computation is used.
cores	an integer specifying the requested number of workers when parallel = TRUE. If NULL, the function uses 2 workers by default (subject to future::availableCores()). The number of workers used is min(cores, future::availableCores()).
silent	a logical indicating whether progress information is printed.

Details

This is useful before launching a large run, for example on a compute server or under a batch system such as SLURM, where an approximate runtime is needed to request resources.

The boolean silent controls whether progress information is printed to the console. For details on the algorithm and the meaning of the arguments, see grid_search.

Value

a numeric value giving the estimated runtime in seconds.

Author(s)

Yukai Yang, <yukai.yang@statistik.uu.se>

See Also

build_grid, grid_search

Examples

```
# Rastrigin function
ndim <- 2
nA <- 10
Rastrigin <- function(vx) nA * ndim + sum(vx * vx - nA * cos(2 * pi * vx))

# Build a grid
bin <- c(from = -5.12, to = 5.12, by = .5)
grid <- build_grid(bin, bin)

# Estimate runtime (serial)
t_est <- grid_search_check(Rastrigin, grid, silent = FALSE)
t_est


# Finer grid
bin <- c(from = -5.12, to = 5.12, by = .1)
grid <- build_grid(bin, bin)

# Estimate runtime, then run the search
t_est <- grid_search_check(Rastrigin, grid, parallel = TRUE, cores = 2, silent = FALSE)
ret   <- grid_search(Rastrigin, grid, parallel = TRUE, cores = 2, silent = FALSE)
```

Index