Package ‘zoomgrid’

January 3, 2019

Type Package

Title Grid Search Algorithm with a Zoom

Version 1.0.0

Description Provides the grid search algorithm with a zoom.
The grid search algorithm with a zoom aims to help solving difficult optimization problem
where there are many local optima inside the domain of the target function.
It offers suitable initial or starting value for the following optimization procedure,
provided that the global optimum exists in the neighbourhood of the initial or starting value.
The grid search algorithm with a zoom saves time tremendously in cases with high-
dimensional arguments.

Depends R (>= 3.0.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

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

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

Imports parallel

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Yukai Yang [aut, cre] (<https://orcid.org/0000-0002-2623-8549>)

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

Repository CRAN

Date/Publication 2019-01-03 17:00:03 UTC
**R topics documented:**

- build_grid .............................................. 2
- grid_search ............................................. 3
- grid_search_check ..................................... 6
- zoomgrid ............................................... 8

**Index**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_grid</td>
<td>2</td>
</tr>
<tr>
<td>grid_search</td>
<td>3</td>
</tr>
<tr>
<td>grid_search_check</td>
<td>6</td>
</tr>
<tr>
<td>zoomgrid</td>
<td>8</td>
</tr>
</tbody>
</table>

---

**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 wanna 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

```r
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

```r
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 the class GRID from `build_grid`.
- **MoreArgs**: a list of other arguments to FUN, see `mapply`.
- **zoom**: number of (additional) rounds or layers of the zoom-in, 0 by default.
- **decay**: a number in between 0 and 1 representing the decay rate of the grid sizes of the zoom.
- **num**: number of points to return, i.e. the smallest num points, 1 by default the minimum.
- **parallel**: a boolean indicating if the parallel computation is carried out, by default FALSE.
- **cores**: The number of cores to use, i.e. at most how many child processes will be run simultaneously. For details, see `mcmapply` in parallel package.
- **silent**: a boolean indicating if the information regarding the computation is printed.

Details

The target function FUN to be minimized is a scalar real valued function with multiple arguments. The maximization can be achieved by multiplying -1 to the original function, and then input the new function to FUN.

The grid must be created by using the function `build_grid` function in the package.

Any other invariant arguments to the function FUN can be specified in MoreArgs by using a list with variable names.

The common grid search first build a grid within some bounded area. And then the target function will be evaluated at each point in the grid. The points that produce the smallest num target function values shall be returned.

The grid search algorithm with a zoom consists of

\[ n^0 + n^1 + n^2 + \ldots + n^z \]

grid searches, where \( n = \text{num} \) and \( z = \text{zoom} \).

As mentioned above, in each grid search, num is the number of points that will be returned. And therefore, in the end, there will be

\[ n^1 + n^2 + n^3 + \ldots + n^{z+1} \]

points returned, where \( n = \text{num} \) and \( z = \text{zoom} \).

Each time when the algorithm zooms in, it will automatically build subgrids based on the points that have been found in the super grid search. Due to the exhaustive property of the grid search algorithm, it is desirable to make fewer points in the subgrid. The decay rate decay provides the opportunity to control the number of points in the subgrids. The number of points for each argument of the target function in the subgrid will be \( \max \{ \text{Int} (\text{decay} \ast N), 3 \} \).
Parallel computation is implemented in the function, which can be activated by setting `parallel = TRUE`.
cores, which represents the number of cores, works only when `parallel = TRUE`. By default `cores=NULL` implies that the function will detect the number of cores and use it.
The boolean silent controls if there will be output in the console.

Value

a list containing the results from the grid search with a zoom.
The list contains the following components:
par the approximate global minimizer
points all the local minimizer points found by the grid search with a zoom

Author(s)

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

See Also

build_grid, grid_search_check

Examples

# A rastrigin function
ndim = 2 # number of dimension
na = 10 # parameter A
# vx in [-5.12, 5.12]

# minimizer = rep(0, ndim)
# minimum = 0
Rastrigin <- function(vx) return(na * ndim + sum(vx*vx - na * cos(2*pi*vx)))

# set seed and initialize the initial or starting value
set.seed(1)
par = runif(ndim, -5.12, 5.12)
cat("start from", par)

# results from different optimization algorithms
optim(par = par, Rastrigin, method='Nelder-Mead')
optim(par = par, Rastrigin, method='BFGS')
optim(par = par, Rastrigin, method='L-BFGS-B')
optim(par = par, Rastrigin, method='SANN')

# a toy example
# build the grid first
bin = c(from=-5.12, to=5.12, by=.5)
grid = build_grid(bin, bin)
# so this is a relatively sparse grid
grid_search_check

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

# We can build a 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
ret2 = grid_search(Rastrigin, grid, num=2, parallel=TRUE, silent=FALSE)
ret2$par

# grid search with a zoom!
ret3 = grid_search(Rastrigin, grid, zoom=2, num=2, parallel=TRUE, silent=FALSE)
ret3$par

grid_search_check

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

Description

This function checks the time consumed by running the grid search algorithm with a zoom as well as some other conditions.

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 the class GRID from build_grid.

MoreArgs 
 a list of other arguments to FUN, see mapply.

zoom 
number of (additional) rounds or layers of the zoom-in, 0 by default.

decay 
a number in between 0 and 1 representing the decay rate of the grid sizes of the zoom.

num 
number of points to return, i.e. the smallest num points, 1 by default the minimum.
grid_search_check

parallel a boolean indicating if the parallel computation is carried out, by default FALSE.
cores The number of cores to use, i.e. at most how many child processes will be run simultaneously. For details, see mcmapply in parallel package.
silent a boolean indicating if the information regarding the computation is printed.

Details

The running of this function takes only several seconds. So it is recommended to run this function before grid_search to check the approximate time consumed by grid_search by using exactly the same arguments.

This function is extremely useful when the user is going to run grid_search on some super-computing server and need to know approximately how long time it will take in order to specify the corresponding settings according to some batch system like SLURM for example.

The boolean silent controls if there will be output in the console.

For details, see grid_search.

Value

a number of the time in seconds.

Author(s)

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

See Also

build_grid, grid_search

Examples

# Rastrigin function
ndim = 2 # number of dimension
nA = 10 # parameter A
# vx in [-5.12, 5.12]

# minimizer = rep(0, ndim)
# minimum = 0
Rastrigin <- function(vx) return(nA * ndim + sum(vx^2 - nA * cos(2*pi*vx)))

# a toy example
# build the grid first
bin = c(from=-5.12, to=5.12, by=.5)
grid = build_grid(bin, bin)
# so this is a relatively sparse grid

# serial computation
ret0 = grid_search(Rastrigin, grid, silent=FALSE)
ret0$par
# If we expand the grid to allow for more points
bin = c(from=-5.12, to=5.12, by=.1)
grid = build_grid(bin, bin)

# run the check before the grid search
ret1 = grid_search_check(Rastrigin, grid, silent=FALSE)
ret1 = grid_search(Rastrigin, grid, silent=FALSE)

---

**zoomgrid**

**zoomgrid: a package implementing the grid search algorithm with a zoom.**

### Description

The package implements the grid search algorithm with a zoom.

### Details

The grid search aims to help solving difficult optimization problem where there are many local optimizers inside the domain of the target function or the parameter space of the log-likelihood function. It offers suitable initial or starting value for the following optimization procedure provided that the global optimum exists in the neighbourhood of the initial or starting value. And therefore the grid search does not do optimization. But if the grid is fine enough, the result approximates the optimizer.

The grid search algorithm consists of two steps:

– firstly, it builds a grid with many points which discretize the domain of the target function;
– secondly, it evaluates the target function at these points in the grid and find the point which produces the smallest (minimization) or largest (maximization) function value.

A fine grid is needed in order for a satisfactory result (initial or starting value of the following optimization, or an approximation of the optimizer). For cases with multiple (high-dimensional) arguments, the grid may contain too many points to evaluate the target function within a reasonably short time period. Thus, a grid search algorithm with a zoom is implemented as an option in the package.

The grid search algorithm with a zoom does not require a very find grid, but a grid with a zoom-in mechanism. Based on the algorithm introduced above, the user of the package can choose to zoom in around the neighbourhoods the first \( n \) smallest (minimization) or largest (maximization) points obtained from the first round grid search. A much finer grid will be built in the neighbourhoods these points, and the grid search will continue.

The grid search with a zoom stops after \( k \) rounds and gives a final result. The result is equivalent to that from a grid search based on (one round) a very fine grid, if the true global optimum is in the neighbourhoods of the \( n \) points from the first round grid search. By doing so, the grid search algorithm with a zoom saves time tremendously in cases with high-dimensional arguments.

Note that the user specifies \( n \) and \( k \) beforehand.

Parallel computation, which requires the `parallel` package, is implemented and is an option in the package.
Author and Maintainer

Yukai Yang
Department of Statistics, Uppsala University
<yukai.yang@statistik.uu.se>

References

Yang, Yukai (2012, 1.5.4) "Modelling Nonlinear Vector Economic Time Series", PhD thesis, Aarhus University, Department of economics and business and CREATEs.

functions in the package

build_grid build the grid for the following grid search algorithm.
grid_search_check check the time consumed by running the grid search algorithm with a zoom.
grid_search conduct the grid search algorithm with a zoom.
Index

*Topic algorithms
  build_grid, 2
  grid_search, 3
  grid_search_check, 6

build_grid, 2, 4–7, 9

grid_search, 3, 3, 7, 9
grid_search_check, 3, 5, 6, 9

mapply, 4, 6

zoomgrid, 8
zoomgrid-package (zoomgrid), 8