Package: optimization (via r-universe)

August 19, 2024

Type Package

Title Flexible Optimization of Complex Loss Functions with State and Parameter Space Constraints

Version 1.0-9

Description Flexible optimizer with numerous input specifications for detailed parameterisation. Designed for complex loss functions with state and parameter space constraints. Visualization tools for validation and analysis of the convergence are included.

License GPL (>= 2)

Depends R (>= 3.2.0), Rcpp (>= 0.12.12)

Imports colorspace

LinkingTo Rcpp

RoxygenNote 7.1.1

Suggests R.rsp

VignetteBuilder R.rsp

URL https://github.com/kaihusmann/optimization

BugReports https://github.com/kaihusmann/optimization/issues

Encoding UTF-8

Repository https://nw-fva.r-universe.dev

RemoteUrl https://github.com/kaihusmann/optimization

RemoteRef HEAD

RemoteSha 6a859454c17e79143e9b99704922053f5b1274ce

Contents

optimization-package *Flexible Optimization of Complex Loss Functions with State and Parameter Space Constraints*

Description

Flexible optimizer with numerous input specifications for detailed parameterisation. Designed for complex loss functions with state and parameter space constraints. Visualization tools for validation and analysis of the convergence are included.

Details

Package: optimization Type: Package Version: 1.0-6 Date: 2017-09-23 License: GPL-2

Author(s)

Kai Husmann [aut, cre], Alexander Lange [aut], Nordwestdeutsche Forstliche Versuchsanstalt (NW-FVA) [cph, fnd]

Maintainer: Kai Husmann <kai.husmann@uni-goettingen.de>

References

Corana, A., Marchesi, M., Martini, C. and Ridella, S. (1987), Minimizing Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm. ACM Transactions on Mathematical Software, 13(3):262-280.

Gao, F. and Han, L. (2012). Implementing the nelder-mead simplex algorithm with adaptive parameters. Computational Optimization and Applications, 51(1):259 277.

Geiger, C. and Kanzow, C. (1999). Das nelder-mead-verfahren. Numerische Verfahren zur Loesung unregestrierter Optimierungsaufgaben.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598): 671-680.

Nelder, J. and Mead, R. (1965). A simplex method for function minimization. Computer Journal, 7(4).

Pronzato, L., Walter, E., Venot, A. and Lebruchec, J.-F. (1984). A general-purpose global optimizer: Implementation and applications. Mathematics and Computers in Simulation, 26(5):412-422.

See Also

optim_nm, optim_sa, optim, plot

Examples

```
hi <- function(x){(x[1]**2 + x[2] - 11)**2 + (x[1] + x[2]**2 -7)**2}
optim_nm(fun = hi, k = 2)
optim_sa(fun = hi, start = c(runif(2, min = -1, max = 1)),
  trace = FALSE,
  lower = c(-4, -4),
  upper = c(4, 4),
  control = list(dyn_rf = FALSE,
    rf = 1.2,
    t0 = 10,
    nlimit = 100,
    r = 0.6,
    t_min = 0.1
  )
)
```

---

optim_nm *Optimization with Nelder-Mead*

Description

This function contains a direct search algorithm, to minimize or maximize an objective function with respect to their input parameters.

Usage

```
optim_nm(fun, k = 0, start, maximum = FALSE, trace = FALSE,
         alpha = 1, beta = 2, gamma = 1/2, delta = 1/2,
         tol = 0.00001, exit = 500, edge = 1)
```

Arguments

fun	Function to minimize or maximize. It should return a single scalar value.
k	Number of parameters of the objective function.
start	Optional vector with starting values. Number of values must be equal to k. The initial simplex is constructed around this start vector.
maximum	Logical. The default is FALSE.
trace	Logical. If TRUE, interim results are stored. Necessary for the plot function. Default is FALSE.
alpha	A positive scalar which indicates the size of the reflected simplex. The value 1 leads to a reflected simplex of the same size as the former iteration.
beta	A positive scalar which indicates the size of the expended simplex. It is usually twice as high as alpha. It must be higher than alpha.
gamma	A positive scalar which indicates the size of either the outside contracted simplex or inside contracted simplex. It is usually half as high as alpha. It must be smaller than alpha.

`delta`	A positive scalar which indicates the size of the shrinked simplex. It is usually half as high as alpha. It must be smaller than `alpha`.
`tol`	A positive scalar describing the tolerance at which the distances in between the function responses of the simplex vertices are close enough to zero to terminate the algorithm.
`exit`	A positive scalar giving the maximum number of iterations the algorithm is allowed to take. It is used to prevent infinite loops. In case of optimizing functions with higher dimensions it is quite likely that the algorithm needs more than 500 iterations. The value should therefore be adjusted to the specific optimization problem.
`edge`	A positive scalar providing the edge length of the initial simplex. It is useful to adjust the edge length if the initial guess is close to the global optimum or if the parameter space of the loss function is relatively small.

Details

The Nelder-Mead method is a comparatively simple heuristic optimization algorithm. It is, However, useful for relatively simple optimization problems without many local minima and low dimensions(n < 10). Nevertheless, the speed and accuracy are rather useful for simple problems. Moreover, the Nelder-Mead is able to optimize functions without derivatives. The handling of the optimization function is quite easy, because there are only few parameters to adjust.

Value

The output is a nmsa_optim object with following entries:

`par` Function parameters after optimization.

`function_value` Function response after optimization.

`trace` Matrix with interim results. NULL if `trace` was not activated.

`fun` The loss function.

`start` The initial function parameters.

`lower` The lower boundaries of the function parameters.

`upper` The upper boundaries of the function parameters.

`control` The number of parameters and iterations of the algorithm.

Author(s)

Alexander Lange

References

Gao, F. and Han, L. (2012). Implementing the nelder-mead simplex algorithm with adaptive parameters. Computational Optimization and Applications, 51(1):259 277.

Geiger, C. and Kanzow, C. (1999). Das Nelder-Mead-Verfahren. Numerische Verfahren zur Loesung unregestrierter Optimierungsaufgaben.

Nelder, J. and Mead, R. (1965). A simplex method for function minimization. Computer Journal, 7(4).

See Also

optim_sa, optim, plot.optim_nmsa

Examples

```
#####  Rosenbrock function
# minimum at f(1,1) = 0
   B <- function(x){
    100*(x[2]-x[1]^2)^2+(1-x[1])^2
  }

##### Minimization with an initial guess at c(-2.048, 2.048)
  optim_nm(B, start = c(-2.048, 2.048))

#####  Himmelblau's function
# minimum at f(3,2) = 0
# f(-2.805, -3.1313) = 0
# f(-3.779, -3.283) = 0
#f(3.5844, -1.848) = 0
  H <- function(x){
    (x[1]^2+x[2]-11)^2+(x[1]+x[2]^2-7)^2
  }

##### Minimization with defined number of parameters
  optim_nm(fun = H, k = 2)

##### Colville function with 4 parameters
  co <- function(x){
    x1 <- x[1]
    x2 <- x[2]
    x3 <- x[3]
    x4 <- x[4]

    term1 <- 100 * (x1^2 - x2)^2
    term2 <- (x1 - 1)^2
    term3 <- (x3-1)^2
    term4 <- 90 * (x3^2 - x4)^2
    term5 <- 10.1 * ((x2 - 1)^2 + (x4 - 1)^2)
    term6 <- 19.8 * (x2 - 1)*(x4-1)

     y <- term1 + term2 + term3 + term4 + term5 + term6
  }

  optim_nm(co, k = 4)

#### Minimization with trace
  Output <- optim_nm(H, k = 2, trace = TRUE)
  plot(Output)
  plot(Output, 'contour')
```

optim_sa *Flexible Optimization with Simulated Annealing*

Description

Random search optimization method with systematic component that searches the global optimum. The loss function is allowed to be non-linear, non-differentiable and multimodal. Undefined responses are allowed as well.

Usage

```
optim_sa(fun, start, maximization = FALSE, trace = FALSE,
         lower, upper, control = list())
```

Arguments

fun	Loss function to be optimized. It must return a scalar value. The variables must be assigned as a vector. See 'details'.
start	Vector of initial values for the function variables. Must be of same length as the variables vector of the loss function. The response of the initial variables combination must be defined (NA or NaN responses are not allowed).
maximization	Logical. Default is FALSE.
trace	Logical. If TRUE, interim results are stored. Necessary for the plot function. Default is FALSE.
lower	Vector of lower boundaries for the function variables. Must be of same length as the variables vector of the function.
upper	Vector of upper boundaries for the function variables. Must be of same length as the variables vector of the function.
control	List with optional further arguments to modify the optimization specifically to the loss function:

- `vf` Function that determines the variation of the function variables for the next iteration. The variation function is allowed to depend on the vector of variables of the current iteration, the vector of random factors `rf` and the temperature of the current iteration. Default is a uniform distributed random number with relative range `rf`.
- `rf` Numeric vector. Random factor vector that determines the variation of the random number of `vf` in relation to the dimension of the function variables for the following iteration. Default is 1. If `dyn_rf` is enabled, the `rf` change dynamically over time.
- `dyn_rf` Logical. `rf` change dynamically over time to ensure increasing precision with increasing number of iterations. Default is TRUE, see 'details'.
- `t0` Numeric. Initial temperature. Default is 1000.
- `nlimit` Integer. Maximum number of iterations of the inner loop. Default is 100.

`r` Numeric. Temperature reduction in the outer loop. Default is 0.6.

`k` Numeric. Constant for the Metropolis function. Default is 1.

`t_min` Numeric. Temperature where outer loop stops. Default is 0.1.

`maxgood` Integer. Break criterion to improve the algorithm performance. Maximum number of loss function improvements in the inner loop. Breaks the inner loop. Default is 100.

`stopac` Integer. Break criterion to improve the algorithm performance. Maximum number of repetitions where the loss improvement is lower than `ac_acc`. Breaks the inner loop. Default is 30.

`ac_acc` Numeric. Accuracy of the `stopac` break criterion in relation to the response. Default is 1/10000 of the function value at initial variables combination.

Details

Simulated Annealing is an optimization algorithm for solving complex functions that may have several optima. The method is composed of a random and a systematic component. Basically, it randomly modifies the variables combination `n_limit` times to compare their response values. Depending on the temperature and the constant `k`, there is also a likelihood of choosing variables combinations with worse response. There is thus a time-decreasing likelihood of leaving local optima. The Simulated Annealing Optimization method is therefore advantageous for multimodal functions. Undefined response values (NA) are allowed as well. This can be useful for loss functions with variables restrictions. The high number of parameters allows a very flexible parameterization. `optim_sa` is able to solve mathematical formulas as well as complex rule sets.

The performance therefore highly depends on the settings. It is indispensable to parameterize the algorithm carefully. The control list is pre-parameterized for loss functions of medium complexity. To improve the performance, the settings should be changed when solving relatively simple functions (e. g. three dimensional multimodal functions). For complex functions the settings should be changed to improve the accuracy. Most important parameters are `nlimit`, `r` and `t0`.

The dynamic `rf` adjustment depends on the number of loss function calls which are out of the variables boundaries as well as the temperature of the current iteration. The obligatory decreasing `rf` ensures a relatively wide search grid at the beginning of the optimization process that shrinks over time. It thus automatically adjusts for the trade-off between range of the search grid and accuracy. See Pronzato (1984) for more details. It is sometimes useful to disable the dynamic `rf` changing when the most performant `rf` are known. As `dyn_rf` usually improves the performance as well as the accuracy, the default is TRUE.

Value

The output is a nmsa_optim list object with following entries:

`par` Function variables after optimization.

`function_value` Loss function response after optimization.

`trace` Matrix with interim results. NULL if `trace` was not activated.

`fun` The loss function.

`start` The initial function variables.

`lower` The lower boundaries of the function variables.

`upper` The upper boundaries of the function variables.

`control` Control arguments, see 'details'.

Author(s)

Kai Husmann

References

Corana, A., Marchesi, M., Martini, C. and Ridella, S. (1987), Minimizing Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm. ACM Transactions on Mathematical Software, 13(3):262-280.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598):671-680.

Pronzato, L., Walter, E., Venot, A. and Lebruchec, J.-F. (1984). A general-purpose global optimizer: Implementation and applications. Mathematics and Computers in Simulation, 26(5):412-422.

See Also

`optim_nm`, `optim`, `plot.optim_nmsa`

Examples

```
##### Rosenbrock function
# minimum at f(1,1) = 0
ro <- function(x){
  100*(x[2]-x[1]^2)^2+(1-x[1])^2
}

# Random start values. Example arguments for the relatively simple Rosenbrock function.
ro_sa <- optim_sa(fun = ro,
                  start = c(runif(2, min = -1, max = 1)),
                  lower = c(-5, -5),
                  upper = c(5, 5),
                  trace = TRUE,
                  control = list(t0 = 100,
                            nlimit = 550,
                            t_min = 0.1,
                            dyn_rf = FALSE,
                            rf = 1,
                            r = 0.7
                  )
        )


# Visual inspection.
plot(ro_sa)
plot(ro_sa, type = "contour")
```

```
##### Holder table function

# 4 minima at
  #f(8.055, 9.665) = -19.2085
  #f(-8.055, 9.665) = -19.2085
  #f(8.055, -9.665) = -19.2085
  #f(-8.055, -9.665) = -19.2085

ho <- function(x){
  x1 <- x[1]
  x2 <- x[2]

  fact1 <- sin(x1) * cos(x2)
  fact2 <- exp(abs(1 - sqrt(x1^2 + x2^2) / pi))
  y <- -abs(fact1 * fact2)
}

# Random start values. Example arguments for the relatively complex Holder table function.
optim_sa(fun = ho,
         start = c(1, 1),
         lower = c(-10, -10),
         upper = c(10, 10),
         trace = TRUE,
         control = list(dyn_rf = FALSE,
                        rf = 1.6,
                        t0 = 10,
                        nlimit = 200,
                        r = 0.6,
                        t_min = 0.1
         )
)
```

`plot.optim_nmsa` *Plot an optim_nmsa Object*

Description

Creates convergence or contour plots for visual inspection of the optimization result. Note that 'trace' must be activated for this function.
In case of a bivariate optimization, the 'contour' plot gives an overview of the parameter development over time in the entire state space. This is useful for the evaluation of the algorithm settings and therefore helps improving the performance. The development of the response can be visualized via the 'convergence' plot.

Usage

```
## S3 method for class 'optim_nmsa'
plot(x, type = 'convergence', lower = NA, upper = NA, ...)
```

Arguments

x	Object of type 'optim_nmsa' to be plotted. The 'trace' entry must not be empty.
type	Character string which determines the plot type. Either 'convergence' or 'contour' is possible.
lower	Vector containing the lower limits of the variables in the plot. Only useful for 'contour' plots.
upper	Vector containing the upper limits of the variables in the plot. Only useful for 'contour' plots.
...	Further arguments for the generic plot function.

Author(s)

Kai Husmann, Alexander Lange

See Also

optim_nm, optim_sa

Examples

```
# S3 method for class 'optim_nlme'

# Himmelblau's function
hi <- function(x){(x[1]**2 + x[2] - 11)**2 + (x[1] + x[2]**2 -7)**2}

out_nm <- optim_nm(hi, k = 2, trace = TRUE)
out_sa <- optim_sa(fun = hi, start = c(runif(2, min = -1, max = 1)),
                   trace = TRUE, lower = c(-4, -4) ,upper=c(4, 4),
                   control = list(t0 = 1000, nlimit = 1500,r = 0.8))

# Examples for optimization results via 'Nelder-Mead' method.
plot(out_nm)
plot(out_nm, type = "contour", lower = c(-4, -4), upper = c(4, 4))

# Examples for optimization results via 'Simulated Annealing' method.
plot(out_sa)
plot(out_sa, type = "contour")
```

Index