Package ‘dfoptim’

April 2, 2018

Type Package

Title Derivative-Free Optimization

Description Derivative-Free optimization algorithms. These algorithms do not require gradient information. More importantly, they can be used to solve non-smooth optimization problems.

Depends R (>= 2.10.1)

Version 2018.2-1

Date 2018-02-18

Author Ravi Varadhan, Johns Hopkins University, and Hans W. Borchers,
ABB Corporate Research.

Maintainer Ravi Varadhan <ravi.varadhan@jhu.edu>

URL http://www.jhsph.edu/agingandhealth/People/Faculty_personal_pages/Varadhan.html

License GPL (>= 2)

LazyLoad yes

Repository CRAN

NeedsCompilation no

Date/Publication 2018-04-02 09:37:48 UTC

R topics documented:

`dfoptim` *Derivative-Free Optimization*

Description

Derivative-Free optimization algorithms. These algorithms do not require gradient information. More importantly, they can be used to solve non-smooth optimization problems. They can also handle box constraints on parameters.

Details

Package:	dfoptim
Type:	Package
Version:	2016.7-1
Date:	2016-07-08
License:	GPL-2 or greater
LazyLoad:	yes

Derivative-Free optimization algorithms. These algorithms do not require gradient information. More importantly, they can be used to solve non-smooth optimization problems. These algorithms were translated from the Matlab code of Prof. C.T. Kelley, given in his book "Iterative methods for optimization". However, there are some non-trivial modifications of the algorithm.

Currently, the Nelder-Mead and Hooke-Jeeves algorithms is implemented. In future, more derivative-free algorithms may be added.

Author(s)

Ravi Varadhan, Johns Hopkins University
URL: http://www.jhsph.edu/agingandhealth/People/Faculty_personal_pages/Varadhan.html
Hans W. Borchers, ABB Corporate Research
Maintainer: Ravi Varadhan <ravi.varadhan@jhu.edu>

References

C.T. Kelley (1999), Iterative Methods for Optimization, SIAM.

`hjk` *Hooke-Jeeves derivative-free minimization algorithm*

Description

An implementation of the Hooke-Jeeves algorithm for derivative-free optimization. A bounded and an unbounded version are provided.

Usage

```
hjk(par, fn, control = list(), ...)

hjkb(par, fn, lower = -Inf, upper = Inf, control = list(), ...)
```

Arguments

`par`	Starting vector of parameter values. The initial vector may lie on the boundary. If `lower[i]=upper[i]` for some `i`, the `i`-th component of the solution vector will simply be kept fixed.
`fn`	Nonlinear objective function that is to be optimized. A scalar function that takes a real vector as argument and returns a scalar that is the value of the function at that point.
`lower, upper`	Lower and upper bounds on the parameters. A vector of the same length as the parameters. If a single value is specified, it is assumed that the same bound applies to all parameters. The starting parameter values must lie within the bounds.
`control`	A list of control parameters. See **Details** for more information.
`...`	Additional arguments passed to `fn`.

Details

Argument `control` is a list specifing changes to default values of algorithm control parameters. Note that parameter names may be abbreviated as long as they are unique.

The list items are as follows:

- `tol` Convergence tolerance. Iteration is terminated when the step length of the main loop becomes smaller than `tol`. This does *not* imply that the optimum is found with the same accuracy. Default is 1.e-06.
- `maxfeval` Maximum number of objective function evaluations allowed. Default is Inf, that is no restriction at all.
- `maximize` A logical indicating whether the objective function is to be maximized (TRUE) or minimized (FALSE). Default is FALSE.
- `target` A real number restricting the absolute function value. The procedure stops if this value is exceeded. Default is Inf, that is no restriction.
- `info` A logical variable indicating whether the step number, number of function calls, best function value, and the first component of the solution vector will be printed to the console. Default is FALSE.

If the minimization process threatens to go into an infinite loop, set either `maxfeval` or `target`.

Value

A list with the following components:

`par`	Best estimate of the parameter vector found by the algorithm.
`value`	value of the objective function at termination.

`convergence`	indicates convergence (=`0`) or not (=`1`).
`feval`	number of times the objective `fn` was evaluated.
`niter`	number of iterations in the main loop.

Note

This algorithm is based on the Matlab code of Prof. C. T. Kelley, given in his book "Iterative methods for optimization". It is implemented here with the permission of Prof. Kelley.

This version does not (yet) implement a cache for storing function values that have already been computed as searching the cache makes it slower.

Author(s)

Hans W Borchers <hwborchers@googlemail.com>

References

C.T. Kelley (1999), Iterative Methods for Optimization, SIAM.

Quarteroni, Sacco, and Saleri (2007), Numerical Mathematics, Springer.

See Also

`optim`, `nmk`

Examples

```
##  Hooke-Jeeves solves high-dim. Rosenbrock function
  rosenbrock <- function(x){
    n <- length(x)
    sum (100*(x[1:(n-1)]^2 - x[2:n])^2 + (x[1:(n-1)] - 1)^2)
  }
par0 <- rep(0, 10)
hjk(par0, rosenbrock)

hjkb(c(0, 0, 0), rosenbrock, upper = 0.5)
# $par
# [1] 0.50000000 0.25742722 0.06626892


##  Hooke-Jeeves does not work well on non-smooth functions
  nsf <- function(x) {
f1 <- x[1]^2 + x[2]^2
f2 <- x[1]^2 + x[2]^2 + 10 * (-4*x[1] - x[2] + 4)
f3 <- x[1]^2 + x[2]^2 + 10 * (-x[1] - 2*x[2] + 6)
max(f1, f2, f3)
  }
par0 <- c(1, 1)                                 # true min 7.2 at (1.2, 2.4)
hjk(par0, nsf) # fmin=8 at xmin=(2,2)
```

nmk *Nelder-Mead optimziation algorithm for derivative-free optimization*

Description

An implementation of the Nelder-Mead algorithm for derivative-free optimization. This allows bounds to be placed on parameters. Bounds are enforced by means of a parameter transformation.

Usage

```
nmk(par, fn, control = list(), ...)

nmkb(par, fn, lower=-Inf, upper=Inf, control = list(), ...)
```

Arguments

`par`	A starting vector of parameter values. Must be feasible, i.e. lie strictly between lower and upper bounds.
`fn`	Nonlinear objective function that is to be optimized. A scalar function that takes a real vector as argument and returns a scalar that is the value of the function at that point (see details).
`lower`	Lower bounds on the parameters. A vector of the same length as the parameters. If a single value is specified, it is assumed that the same lower bound applies to all parameters.
`upper`	Upper bounds on the parameters. A vector of the same length as the parameters. If a single value is specified, it is assumed that the same upper bound applies to all parameters.
`control`	A list of control parameters. See *Details* for more information.
`...`	Additional arguments passed to `fn`

Details

Argument `control` is a list specifing any changes to default values of algorithm control parameters for the outer loop. Note that the names of these must be specified completely. Partial matching will not work. The list items are as follows:

`tol` Convergence tolerance. Iteration is terminated when the absolute difference in function value between successive iteration is below `tol`. Default is 1.e-06.

`maxfeval`: Maximum number of objective function evaluations allowed. Default is min(5000, max(1500, 20*length(par)^2)).

`regsimp` A logical variable indicating whether the starting parameter configuration is a regular simplex. Default is TRUE.

`maximize` A logical variable indicating whether the objective function should be maximized. Default is FALSE.

`restarts.max` Maximum number of times the algorithm should be restarted before declaring failure. Default is 3.

`trace` A logical variable indicating whether the starting parameter configuration is a regular simplex. Default is FALSE.

Value

A list with the following components:

`par`	Best estimate of the parameter vector found by the algorithm.
`value`	The value of the objective function at termination.
`feval`	The number of times the objective `fn` was evaluated.
`restarts`	The number of times the algorithm had to be restarted when it stagnated.
`convergence`	An integer code indicating type of convergence. `0` indicates successful convergence. Positive integer codes indicate failure to converge.
`message`	Text message indicating the type of convergence or failure.

Note

This algorithm is based on the Matlab code of Prof. C.T. Kelley, given in his book "Iterative methods for optimization". It is implemented here with the permission of Prof. Kelley and SIAM. However, there are some non-trivial modifications of the algorithm.

Author(s)

Ravi Varadhan <rvaradhan@jhmi.edu>, Johns Hopkins University URL:http://www.jhsph.edu/agingandhealth/People/Facult

References

C.T. Kelley (1999), Iterative Methods for Optimization, SIAM.

See Also

optim, hjk

Examples

```
 rosbkext <- function(x){
# Extended Rosenbrock function
 n <- length(x)
 sum (100*(x[1:(n-1)]^2 - x[2:n])^2 + (x[1:(n-1)] - 1)^2)
 }

np <- 10
set.seed(123)

p0 <- rnorm(np)
xm1 <- nmk(fn=rosbkext, par=p0) # maximum `fevals' is not sufficient to find correct minimum
xm1b <- nmkb(fn=rosbkext, par=p0, lower=-2, upper=2)
```

```
### A non-smooth problem
hald <- function(x) {
#Hald J & Madsen K (1981), Combined LP and quasi-Newton methods
#for minimax optimization, Mathematical Programming, 20, p.42-62.
i <- 1:21
t <- -1 + (i - 1)/10
f <- (x[1] + x[2] * t) / ( 1 + x[3]*t + x[4]*t^2 + x[5]*t^3) - exp(t)
max(abs(f))
}

p0 <- runif(5)
xm2 <- nmk(fn=hald, par=p0)
xm2b <- nmkb(fn=hald, par=p0, lower=c(0,0,0,0,-2), upper=4)

## Another non-smooth functions
  nsf <- function(x) {
f1 <- x[1]^2 + x[2]^2
f2 <- x[1]^2 + x[2]^2 + 10 * (-4*x[1] - x[2] + 4)
f3 <- x[1]^2 + x[2]^2 + 10 * (-x[1] - 2*x[2] + 6)
max(f1, f2, f3)
  }
par0 <- c(1, 1)                                 # true min 7.2 at (1.2, 2.4)
nmk(par0, nsf) # fmin=8 at xmin=(2,2)
```

Index