Package 'optimx'

February 20, 2015

Version 2013.8.7

Date 2013-08-07

Title A Replacement and Extension of the optim() Function

Author John C Nash [aut, cre], Ravi Varadhan [aut], Gabor Grothendieck [ctb]

Maintainer John C Nash <nashjc@uottawa.ca>

Description Provides a replacement and extension of the optim() function to unify and streamline optimization capabilities in R for smooth, possibly box constrained functions of several or many parameters. This is the CRAN version of the package.

License GPL-2

LazyLoad Yes

Imports numDeriv, ucminf, BB, Rcgmin, Rvmmin, minqa, setRNG, dfoptim, svUnit

NeedsCompilation no

Repository CRAN

Date/Publication 2014-11-13 08:42:46

R topics documented:

optimx-package *A replacement and extension of the optim() function*

Description

optimx provides a replacement and extension of the link{optim()} function to unify and streamline optimization capabilities in R for smooth, possibly box constrained functions of several or many parameters

Examples can be found on the optimx help page and in the demo directory, demo(package = "optimx").

Author(s)

John C Nash <nashjc@uottawa.ca> and Ravi Varadhan <RVaradhan@jhmi.edu>

Maintainer: John C Nash <nashjc@uottawa.ca>

References

Nash, John C. and Varadhan, Ravi (2011) Unifying Optimization Algorithms to Aid Software System Users: optimx for R, Journal of Statistical Software, publication pending.

See Also

optimx

coef.optimx *Summarize optimx object*

Description

Summarize an "optimx" object.

Usage

```
  ## S3 method for class 'optimx'
coef(object, ...)
  ## S3 replacement method for class 'optimx'
coef(x) <- value
```

Arguments

object	Object returned by optimx.
...	Further arguments to be passed to the function. Currently not used.
x	An optimx object.
value	Set parameters equal to this value.

Value

coef.optimx returns the best parameters found by each method that returned such parameters. The returned coefficients are in the form of a matrix with the rows named by the relevant methods and the columns named according to parameter names provided by the user in the vector of starting values, or else by "p1", "p2", ..., if names are not provided.

Examples

```
ans <- optimx(fn = function(x) sum(x*x), par = 1:2)
coef(ans)

## Not run:
proj <- function(x) x/sum(x)
f <- function(x) -prod(proj(x))
ans <- optimx(1:2, f)
ans
coef(ans) <- apply(coef(ans), 1, proj)
ans

## End(Not run)
```

---

optimx *General-purpose optimization*

Description

General-purpose optimization wrapper function that calls other R tools for optimization, including the existing optim() function. optimx also tries to unify the calling sequence to allow a number of tools to use the same front-end. These include spg from the BB package, ucminf, nlm, and nlminb. Note that optim() itself allows Nelder–Mead, quasi-Newton and conjugate-gradient algorithms as well as box-constrained optimization via L-BFGS-B. Because SANN does not return a meaningful convergence code (conv), optimx() does not call the SANN method.

Usage

```
optimx(par, fn, gr=NULL, hess=NULL, lower=-Inf, upper=Inf,
            method=c("Nelder-Mead","BFGS"), itnmax=NULL, hessian=FALSE,
            control=list(),
             ...)
```

Arguments

par	a vector of initial values for the parameters for which optimal values are to be found. Names on the elements of this vector are preserved and used in the results data frame.

fn	A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
gr	A function to return (as a vector) the gradient for those methods that can use this information. If 'gr' is NULL, a finite-difference approximation will be used. An open question concerns whether the SAME approximation code used for all methods, or whether there are differences that could/should be examined?
hess	A function to return (as a symmetric matrix) the Hessian of the objective function for those methods that can use this information.
lower, upper	Bounds on the variables for methods such as "L-BFGS-B" that can handle box (or bounds) constraints.
method	A list of the methods to be used. Note that this is an important change from optim() that allows just one method to be specified. See 'Details'.
itnmax	If provided as a vector of the same length as the list of methods method, gives the maximum number of iterations or function values for the corresponding method. If a single number is provided, this will be used for all methods. Note that there may be control list elements with similar functions, but this should be the preferred approach when using optimx.
hessian	A logical control that if TRUE forces the computation of an approximation to the Hessian at the final set of parameters. If FALSE (default), the hessian is calculated if needed to provide the KKT optimality tests (see kkt in 'Details' for the control list). This setting is provided primarily for compatibility with optim().
control	A list of control parameters. See 'Details'.
...	For optimx further arguments to be passed to fn and gr; otherwise, further arguments are not used.

Details

Note that arguments after ... must be matched exactly.

By default this function performs minimization, but it will maximize if control$maximize is TRUE. The original optim() function allows control$fnscale to be set negative to accomplish this. DO NOT use both methods.

Possible method codes at the time of writing are 'Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B', 'nlm', 'nlminb', 'spg', 'ucminf', 'newuoa', 'bobyqa', 'nmkb', 'hjkb', 'Rcgmin', or 'Rvmmin'.

The default methods for unconstrained problems (no lower or upper specified) are an implementation of the Nelder and Mead (1965) and a Variable Metric method based on the ideas of Fletcher (1970) as modified by him in conversation with Nash (1979). Nelder-Mead uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions. The Variable Metric method, "BFGS" updates an approximation to the inverse Hessian using the BFGS update formulas, along with an acceptable point line search strategy. This method appears to work best with analytic gradients. ("Rvmmmin" provides a box-constrained version of this algorithm.

If no method is given, and there are bounds constraints provided, the method is set to "L-BFGS-B".

Method "CG" is a conjugate gradients method based on that by Fletcher and Reeves (1964) (but with the option of Polak–Ribiere or Beale–Sorenson updates). The particular implementation is now dated, and improved yet simpler codes are being implemented (as at June 2009), and furthermore a version with box constraints is being tested. Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.

Method "L-BFGS-B" is that of Byrd *et. al.* (1995) which allows *box constraints*, that is each variable can be given a lower and/or upper bound. The initial value must satisfy the constraints. This uses a limited-memory modification of the BFGS quasi-Newton method. If non-trivial bounds are supplied, this method will be selected, with a warning.

Nocedal and Wright (1999) is a comprehensive reference for the previous three methods.

Function fn can return NA or Inf if the function cannot be evaluated at the supplied value, but the initial value must have a computable finite value of fn. However, some methods, of which "L-BFGS-B" is known to be a case, require that the values returned should always be finite.

While optim can be used recursively, and for a single parameter as well as many, this may not be true for optimx. optim also accepts a zero-length par, and just evaluates the function with that argument.

Method "nlm" is from the package of the same name that implements ideas of Dennis and Schnabel (1983) and Schnabel et al. (1985). See nlm() for more details.

Method "nlminb" is the package of the same name that uses the minimization tools of the PORT library. The PORT documentation is at <URL: http://netlib.bell-labs.com/cm/cs/cstr/153.pdf>. See nlminb() for details. (Though there is very little information about the methods.)

Method "spg" is from package BB implementing a spectral projected gradient method for large-scale optimization with simple constraints due R adaptation, with significant modifications, by Ravi Varadhan, Johns Hopkins University (Varadhan and Gilbert, 2009), from the original FORTRAN code of Birgin, Martinez, and Raydan (2001).

Method "Rcgmin" is from the package of that name. It implements a conjugate gradient algorithm with the Yuan/Dai update (ref??) and also allows bounds constraints on the parameters. (Rcgmin also allows mask constraints – fixing individual parameters – but there is no interface from "optimx".)

Methods "bobyqa", "uobyqa" and "newuoa" are from the package "minqa" which implement optimization by quadratic approximation routines of the similar names due to M J D Powell (2009). See package minqa for details. Note that "uobyqa" and "newuoa" are for unconstrained minimization, while "bobyqa" is for box constrained problems. While "uobyqa" may be specified, it is NOT part of the all.methods = TRUE set.

The control argument is a list that can supply any of the following components:

trace Non-negative integer. If positive, tracing information on the progress of the optimization is produced. Higher values may produce more tracing information: for method "L-BFGS-B" there are six levels of tracing. trace = 0 gives no output (To understand exactly what these do see the source code: higher levels give more detail.)

follow.on = TRUE or FALSE. If TRUE, and there are multiple methods, then the last set of parameters from one method is used as the starting set for the next.

save.failures = TRUE if we wish to keep "answers" from runs where the method does not return convcode==0. FALSE otherwise (default).

maximize = TRUE if we want to maximize rather than minimize a function. (Default FALSE). Methods nlm, nlminb, ucminf cannot maximize a function, so the user must explicitly minimize and carry out the adjustment externally. However, there is a check to avoid usage of these codes when maximize is TRUE. See fnscale below for the method used in optim that we deprecate.

all.methods = TRUE if we want to use all available (and suitable) methods.

kkt =FALSE if we do NOT want to test the Kuhn, Karush, Tucker optimality conditions. The default is TRUE. However, because the Hessian computation may be very slow, we set kkt to be FALSE if there are more than than 50 parameters when the gradient function gr is not provided, and more than 500 parameters when such a function is specified. We return logical values KKT1 and KKT2 TRUE if first and second order conditions are satisfied approximately. Note, however, that the tests are sensitive to scaling, and users may need to perform additional verification. If kkt is FALSE but hessian is TRUE, then KKT1 is generated, but KKT2 is not.

all.methods = TRUE if we want to use all available (and suitable) methods.

kkttol = value to use to check for small gradient and negative Hessian eigenvalues. Default = .Machine$double.eps^(1/3)

kkt2tol = Tolerance for eigenvalue ratio in KKT test of positive definite Hessian. Default same as for kkttol

starttests = TRUE if we want to run tests of the function and parameters: feasibility relative to bounds, analytic vs numerical gradient, scaling tests, before we try optimization methods. Default is TRUE.

dowarn = TRUE if we want warnings generated by optimx. Default is TRUE.

badval = The value to set for the function value when try(fn()) fails. Default is (0.5)*.Machine$double.xmax

usenumDeriv = TRUE if the numDeriv function grad() is to be used to compute gradients when the argument gr is NULL or not supplied.

The following control elements apply only to some of the methods. The list may be incomplete. See individual packages for details.

fnscale An overall scaling to be applied to the value of fn and gr during optimization. If negative, turns the problem into a maximization problem. Optimization is performed on fn(par)/fnscale. For methods from the set in optim(). Note potential conflicts with the control maximize.

parscale A vector of scaling values for the parameters. Optimization is performed on par/parscale and these should be comparable in the sense that a unit change in any element produces about a unit change in the scaled value.For optim.

ndeps A vector of step sizes for the finite-difference approximation to the gradient, on par/parscale scale. Defaults to 1e-3. For optim.

maxit The maximum number of iterations. Defaults to 100 for the derivative-based methods, and 500 for "Nelder-Mead".

abstol The absolute convergence tolerance. Only useful for non-negative functions, as a tolerance for reaching zero.

reltol Relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of reltol * (abs(val) + reltol) at a step. Defaults to sqrt(.Machine$double.eps), typically about 1e-8. For optim.

`alpha`, `beta`, `gamma` Scaling parameters for the `"Nelder-Mead"` method. `alpha` is the reflection factor (default 1.0), `beta` the contraction factor (0.5) and `gamma` the expansion factor (2.0).

`REPORT` The frequency of reports for the `"BFGS"` and `"L-BFGS-B"` methods if `control$trace` is positive. Defaults to every 10 iterations for `"BFGS"` and `"L-BFGS-B"`.

`type` for the conjugate-gradients method. Takes value `1` for the Fletcher–Reeves update, `2` for Polak–Ribiere and `3` for Beale–Sorenson.

`lmm` is an integer giving the number of BFGS updates retained in the `"L-BFGS-B"` method, It defaults to `5`.

`factr` controls the convergence of the `"L-BFGS-B"` method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. Default is `1e7`, that is a tolerance of about `1e-8`.

`pgtol` helps control the convergence of the `"L-BFGS-B"` method. It is a tolerance on the projected gradient in the current search direction. This defaults to zero, when the check is suppressed.

Any names given to `par` will be copied to the vectors passed to `fn` and `gr`. Note that no other attributes of `par` are copied over. (We have not verified this as at 2009-07-29.)

There are `[.optimx`, `as.data.frame.optimx`, `coef.optimx` and `summary.optimx` methods available.

Value

If there are `npar` parameters, then the result is a dataframe having one row for each method for which results are reported, using the method as the row name, with columns

`par_1, .., par_npar, value, fevals, gevals, niter, convcode, kkt1, kkt2, xtimes`

where

par_1 ..

par_npar The best set of parameters found.

value The value of `fn` corresponding to `par`.

fevals The number of calls to `fn`.

gevals The number of calls to `gr`. This excludes those calls needed to compute the Hessian, if requested, and any calls to `fn` to compute a finite-difference approximation to the gradient.

niter For those methods where it is reported, the number of "iterations". See the documentation or code for particular methods for the meaning of such counts.

convcode An integer code. `0` indicates successful convergence. Various methods may or may not return sufficient information to allow all the codes to be specified. An incomplete list of codes includes

- `1` indicates that the iteration limit `maxit` had been reached.
- `20` indicates that the initial set of parameters is inadmissible, that is, that the function cannot be computed or returns an infinite, NULL, or NA value.
- `21` indicates that an intermediate set of parameters is inadmissible.
- `10` indicates degeneracy of the Nelder–Mead simplex.
- `51` indicates a warning from the `"L-BFGS-B"` method; see component `message` for further details.

52 indicates an error from the "L-BFGS-B" method; see component message for further details.

kkt1 A logical value returned TRUE if the solution reported has a "small" gradient.

kkt2 A logical value returned TRUE if the solution reported appears to have a positive-definite Hessian.

xtimes The reported execution time of the calculations for the particular method.

The attribute "details" to the returned answer object contains information, if computed, on the gradient (ngatend) and Hessian matrix (nhatend) at the supposed optimum, along with the eigenvalues of the Hessian (hev), as well as the message, if any, returned by the computation for each method, which is included for each row of the details. If the returned object from optimx() is ans, this is accessed via the construct attr(ans, "details")

This object is a matrix based on a list so that if ans is the output of optimx then attr(ans, "details")[1,] gives the first row and attr(ans,"details")["Nelder-Mead",] gives the Nelder-Mead row. There is one row for each method that has been successful or that has been forcibly saved by save.failures=TRUE.

There are also attributes

maximize to indicate we have been maximizing the objective

npar to provide the number of parameters, thereby facilitating easy extraction of the parameters from the results data frame

follow.on to indicate that the results have been computed sequentially, using the order provided by the user, with the best parameters from one method used to start the next. There is an example (ans9) in the script ox.R in the demo directory of the package.

Note

Most methods in optimx will work with one-dimensional pars, but such use is NOT recommended. Use optimize or other one-dimensional methods instead.

There are a series of demos available. Once the package is loaded (via require(optimx) or library(optimx), you may see available demos via

demo(package="optimx")

The demo 'brown_test' may be run with the command demo(brown_test, package="optimx")

The package source contains several functions that are not exported in the NAMESPACE. These are

optimx.setup() which establishes the controls for a given run;

optimx.check() which performs bounds and gradient checks on the supplied parameters and functions;

optimx.run() which actually performs the optimization and post-solution computations;

scalecheck() which actually carries out a check on the relative scaling of the input parameters.

Knowledgeable users may take advantage of these functions if they are carrying out production calculations where the setup and checks could be run once.

Source

See the manual pages for `optim()` and the packages the DESCRIPTION `suggests`.

References

See the manual pages for `optim()` and the packages the DESCRIPTION `suggests`.

Nash JC, and Varadhan R (2011). Unifying Optimization Algorithms to Aid Software System Users: **optimx** for R., *Journal of Statistical Software*, 43(9), 1-14., URL http://www.jstatsoft.org/v43/i09/.

Nash JC (2014). On Best Practice Optimization Methods in R., *Journal of Statistical Software*, 60(2), 1-14., URL http://www.jstatsoft.org/v60/i02/.

See Also

`spg`, `nlm`, `nlminb`, `bobyqa`, `Rcgmin`, `Rvmmin`, `ucminf`, `nmkb`, `hjkb`. `optimize` for one-dimensional minimization; `constrOptim` or `spg` for linearly constrained optimization.

Examples

```
require(graphics)
cat("Note demo(ox) for extended examples\n")


## Show multiple outputs of optimx using all.methods
# genrose function code
genrose.f<- function(x, gs=NULL){ # objective function
## One generalization of the Rosenbrock banana valley function (n parameters)
n <- length(x)
        if(is.null(gs)) { gs=100.0 }
fval<-1.0 + sum (gs*(x[1:(n-1)]^2 - x[2:n])^2 + (x[2:n] - 1)^2)
        return(fval)
}

genrose.g <- function(x, gs=NULL){
# vectorized gradient for genrose.f
# Ravi Varadhan 2009-04-03
n <- length(x)
        if(is.null(gs)) { gs=100.0 }
gg <- as.vector(rep(0, n))
tn <- 2:n
tn1 <- tn - 1
z1 <- x[tn] - x[tn1]^2
z2 <- 1 - x[tn]
gg[tn] <- 2 * (gs * z1 - z2)
gg[tn1] <- gg[tn1] - 4 * gs * x[tn1] * z1
return(gg)
}

genrose.h <- function(x, gs=NULL) { ## compute Hessian
   if(is.null(gs)) { gs=100.0 }
n <- length(x)
hh<-matrix(rep(0, n*n),n,n)
```

```
for (i in 2:n) {
z1<-x[i]-x[i-1]*x[i-1]
z2<-1.0-x[i]
                hh[i,i]<-hh[i,i]+2.0*(gs+1.0)
                hh[i-1,i-1]<-hh[i-1,i-1]-4.0*gs*z1-4.0*gs*x[i-1]*(-2.0*x[i-1])
                hh[i,i-1]<-hh[i,i-1]-4.0*gs*x[i-1]
                hh[i-1,i]<-hh[i-1,i]-4.0*gs*x[i-1]
}
        return(hh)
}

startx<-4*seq(1:10)/3.
ans8<-optimx(startx,fn=genrose.f,gr=genrose.g, hess=genrose.h,
   control=list(all.methods=TRUE, save.failures=TRUE, trace=0), gs=10)
ans8
ans8[, "gevals"]
ans8["spg", ]
summary(ans8, par.select = 1:3)
summary(ans8, order = value)[1, ] # show best value
head(summary(ans8, order = value)) # best few
## head(summary(ans8, order = "value")) # best few -- alternative syntax

## order by value.  Within those values the same to 3 decimals order by fevals.
## summary(ans8, order = list(round(value, 3), fevals), par.select = FALSE)
summary(ans8, order = "list(round(value, 3), fevals)", par.select = FALSE)

## summary(ans8, order = rownames, par.select = FALSE) # order by method name
summary(ans8, order = "rownames", par.select = FALSE) # same

summary(ans8, order = NULL, par.select = FALSE) # use input order
## summary(ans8, par.select = FALSE) # same
```

`summary.optimx` *Summarize optimx object*

Description

Summarize an "optimx" object.

Usage

```
  ## S3 method for class 'optimx'
summary(object, order = NULL, par.select = TRUE, ...)
```

Arguments

`object`	Object returned by optimx.

order	A column name, character vector of columns names, R expression in terms of column names or a list of R expressions in terms of column names. NULL, the default, means no re-ordering. rownames can be used to alphabetic ordering by method name. NULL, the default, causes it not to be reordered. Note that if follow.on is TRUE re-ordering likely makes no sense. The result is ordered by the order specification, each specified column in ascending order (except for value which is in descending order if the optimization problem is a maximization problem).
par.select	a numeric, character or logical vector selecting those par values to display. For example, par=1:5 means display only the first 5 parameters. Recycled so par.select=FALSE selects no parameters.
...	Further arguments to be passed to the function. Currently not used.

Details

If order is specified then the result is reordered by the specified columns, each in ascending order (except possibly for the value column which is re-ordered in descending order for maximization problems).

Value

summary.optimx returns object with the rows ordered according to order and with those parameters selected by par.select.

Examples

```
ans <- optimx(fn = function(x) sum(x*x), par = 1:2)

# order by method name.
summary(ans, order = rownames)

# order by objective value. Do not show parameter values.
summary(ans, order = value, par.select = FALSE)

# order by objective value and then number of function evaluations
# such that objectives that are the same to 3 decimals are
# considered the same.  Show only first parameter.
summary(ans, order = list(round(value, 3), fevals), par.select = 1)
```

Index