Package 'optimr'

April 1, 2018

Version 2018-3.18

Date 2018-03-18

Title Expanded 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
and the R package universe
for smooth, possibly box constrained functions of several or
many parameters. optimr is a refactoring of the optimx package
to simplify maintenance and extension. A great deal of this
package is derived from the earlier package 'optimx', but
some functionality has been separated into distinct functions
to allow for easier maintenance and use, as well as future
development.

License GPL-2

LazyLoad Yes

Imports optextras, numDeriv

NeedsCompilation no

Suggests knitr, rmarkdown, setRNG, BB, ucminf, minqa, dfoptim, Rtnmin,
Rvmmin, Rcgmin, lbfgsb3, lbfgs, subplex

VignetteBuilder knitr

R topics documented:

`checksolver` *Test if requested solver is present*

Description

Test if requested solver is present.

Usage

```
checksolver(method, allmeth, allpkg)
```

Arguments

`method`	Character string giving the name of the solver requested.
`allmeth`	Character vector giving the names of the methods optimr can use.
`allpkg`	Character vector giving the names of the packages where solvers are found.

Value

`checksolver` tests if requested function minimization solver is present.

`coef.opm` *Summarize opm object*

Description

Summarize an `"opm"` object.

Usage

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

Arguments

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

Value

coef.opm 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 <- opm(fn = function(x) sum(x*x), par = 1:2, method="ALL", control=list(trace=1))
coef(ans)

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

## End(Not run)
```

ctrldefault *set control defaults*

Description

Set control defaults.

Usage

```
ctrldefault(npar)
```

Arguments

npar	Number of parameters to optimize.

Value

ctrldefault returns the default control settings for optimization tools.

hjn	*Compact R Implementation of Hooke and Jeeves Pattern Search Optimization*

Description

The purpose of `hjn` is to minimize an unconstrained or bounds (box) and mask constrained function of several parameters by a Hooke and Jeeves pattern search. This code is entirely in R to allow users to explore and understand the method. It also allows bounds (or box) constraints and masks (equality constraints) to be imposed on parameters.

Usage

```
hjn(par, fn, lower=-Inf, upper=Inf, bdmsk=NULL, control = list(trace=0), ...)
```

Arguments

`par`	A numeric vector of starting estimates.
`fn`	A function that returns the value of the objective at the supplied set of parameters `par` using auxiliary data in .... The first argument of `fn` must be `par`.
`lower`	A vector of lower bounds on the parameters.
`upper`	A vector of upper bounds on the parameters.
`bdmsk`	An indicator vector, having 1 for each parameter that is "free" or unconstrained, and 0 for any parameter that is fixed or MASKED for the duration of the optimization.
`control`	An optional list of control settings.
`...`	Further arguments to be passed to `fn`.

Details

Functions `fn` must return a numeric value.

The `control` argument is a list.

maxfeval A limit on the number of function evaluations used in the search.

trace Set 0 (default) for no output, >0 for trace output (larger values imply more output).

eps Tolerance used to calculate numerical gradients. Default is 1.0E-7. See source code for `hjn` for details of application.

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

`tol` Tolerance used in testing the size of the pattern search step.

Note that the control `maximize` should NOT be used.

Value

A list with components:

`par`	The best set of parameters found.
`value`	The value of the objective at the best set of parameters found.
`counts`	A two-element integer vector giving the number of calls to 'fn' and 'gr' respectively. 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.
`convergence`	An integer code. '0' indicates successful convergence. '1' indicates that the function evaluation count 'maxfeval' was reached.
`message`	A character string giving any additional information returned by the optimizer, or 'NULL'.

References

Nash JC (1979). Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation. Adam Hilger, Bristol. Second Edition, 1990, Bristol: Institute of Physics Publications.

See Also

`optim`

Examples

```
#####################
## Rosenbrock Banana function
fr <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

ansrosenbrock0 <- hjn(fn=fr, par=c(1,2), control=list(maxfeval=2000, trace=0))
print(ansrosenbrock0) # use print to allow copy to separate file that
#    can be called using source()
#####################
# Simple bounds and masks test
bt.f<-function(x){
 sum(x*x)
}

n<-10
xx<-rep(0,n)
lower<-rep(0,n)
upper<-lower # to get arrays set
bdmsk<-rep(1,n)
bdmsk[(trunc(n/2)+1)]<-0
for (i in 1:n) {
   lower[i]<-1.0*(i-1)*(n-1)/n
   upper[i]<-1.0*i*(n+1)/n
}
xx<-0.5*(lower+upper)
ansbt<-hjn(xx, bt.f, lower, upper, bdmsk, control=list(trace=1, maxfeval=2000))

print(ansbt)

#####################
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)
}

xx<-rep(pi,10)
lower<-NULL
upper<-NULL
bdmsk<-NULL
genrosea<-hjn(xx,genrose.f, control=list(maxfeval=2000), gs=10)
print(genrosea)

cat("timings B vs U\n")
lo<-rep(-100,10)
up<-rep(100,10)
bdmsk<-rep(1,10)
tb<-system.time(ab<-hjn(xx,genrose.f, lower=lo, upper=up,
          bdmsk=bdmsk, control=list(trace=0, maxfeval=2000)))[1]
tu<-system.time(au<-hjn(xx,genrose.f, control=list(maxfeval=2000, trace=0)))[1]
cat("times U=",tu,"   B=",tb,"\n")
cat("solution hjnu\n")
print(au)
cat("solution hjnb\n")
print(ab)
cat("diff fu-fb=",au$value-ab$value,"\n")
cat("max abs parameter diff = ", max(abs(au$par-ab$par)),"\n")

maxfn<-function(x) {
       n<-length(x)
ss<-seq(1,n)
f<-10-(crossprod(x-ss))^2
f<-as.numeric(f)
return(f)
}

negmaxfn<-function(x) {
f<-(-1)*maxfn(x)
return(f)
}
```

```
   cat("WARNING -- this example does NOT appear to terminate\n")
   cat("test that maximize=TRUE works correctly\n")
   # 160706 -- not set up to maximize yet, except through optimr perhaps
   n<-6
   xx<-rep(1,n)
   ansmax<-hjn(xx,maxfn, control=list(maximize=TRUE,trace=1, maxfeval=2000))
   print(ansmax)

   cat("using the negmax function should give same parameters\n")
   ansnegmax<-hjn(xx,negmaxfn, control=list(trace=1))
   print(ansnegmax)


##################### From Rvmmin.Rd
cat("test bounds and masks\n")
nn<-4
startx<-rep(pi,nn)
lo<-rep(2,nn)
up<-rep(10,nn)
grbds1<-hjn(startx,genrose.f, lower=lo,upper=up, control=list(maxfeval=2000, trace=0))
print(grbds1)

cat("test lower bound only\n")
nn<-4
startx<-rep(pi,nn)
lo<-rep(2,nn)
grbds2<-hjn(startx,genrose.f, lower=lo)
print(grbds2)

cat("test lower bound single value only\n")
nn<-4
startx<-rep(pi,nn)
lo<-2
up<-rep(10,nn)
grbds3<-hjn(startx,genrose.f, lower=lo)
print(grbds3)

cat("test upper bound only\n")
nn<-4
startx<-rep(pi,nn)
lo<-rep(2,nn)
up<-rep(10,nn)
grbds4<-hjn(startx,genrose.f, upper=up, control=list(maxfeval=2000))
print(grbds4)

cat("test upper bound single value only\n")
nn<-4
startx<-rep(pi,nn)
grbds5<-hjn(startx,genrose.f, upper=10, control=list(maxfeval=2000))
print(grbds5)
```

```
cat("test masks only\n")
nn<-6
bd<-c(1,1,0,0,1,1)
startx<-rep(pi,nn)
grbds6<-hjn(startx,genrose.f, bdmsk=bd, control=list(maxfeval=2000))
print(grbds6)

cat("test upper bound on first two elements only\n")
nn<-4
startx<-rep(pi,nn)
upper<-c(10,8, Inf, Inf)
grbds7<-hjn(startx,genrose.f, upper=upper, control=list(maxfeval=2000))
print(grbds7)


cat("test lower bound on first two elements only\n")
nn<-4
startx<-rep(0,nn)
lower<-c(0, -0.1 , -Inf, -Inf)
grbds8a<-hjn(startx,genrose.f, lower=lower, control=list(maxfeval=2000))
print(grbds8a)

cat("test n=1 problem using simple squares of parameter\n")

sqtst<-function(xx) {
   res<-sum((xx-2)*(xx-2))
}

######### One dimension test
nn<-1
startx<-rep(0,nn)
onepar<-hjn(startx,sqtst,control=list(trace=1))
print(onepar)
```

multistart *General-purpose optimization - multiple starts*

Description

Multiple initial parameter wrapper function that calls other R tools for optimization, including the existing optimr() function.

Usage

```
multistart(parmat, fn, gr=NULL, lower=-Inf, upper=Inf,
            method=NULL, hessian=FALSE,
            control=list(),
             ...)
```

Arguments

`parmat`	a matrix of which each row is a set 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?
`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'. The default of NULL causes an appropriate set of methods to be supplied depending on the presence or absence of bounds on the parameters. The default unconstrained set is Rvmminu, Rcgminu, lbfgsb3, newuoa and nmkb. The default bounds constrained set is Rvmminb, Rcgminb, lbfgsb3, bobyqa and nmkb.
`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.

See `optimr()` for other details.

Value

An array with one row per set of starting parameters. Each row contains:

`par`	The best set of parameters found.
`value`	The value of 'fn' corresponding to 'par'.
`counts`	A two-element integer vector giving the number of calls to 'fn' and 'gr' respectively. 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.
`convergence`	An integer code. '0' indicates successful completion

`message`	A character string giving any additional information returned by the optimizer, or 'NULL'.
`hessian`	Always NULL for this routine.

Source

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

Examples

```
fnR <- function (x, gs=100.0)
{
    n <- length(x)
    x1 <- x[2:n]
    x2 <- x[1:(n - 1)]
    sum(gs * (x1 - x2^2)^2 + (1 - x2)^2)
}
grR <- function (x, gs=100.0)
{
    n <- length(x)
    g <- rep(NA, n)
    g[1] <- 2 * (x[1] - 1) + 4*gs * x[1] * (x[1]^2 - x[2])
    if (n > 2) {
        ii <- 2:(n - 1)
        g[ii] <- 2 * (x[ii] - 1) + 4 * gs * x[ii] * (x[ii]^2 - x[ii +
            1]) + 2 * gs * (x[ii] - x[ii - 1]^2)
    }
    g[n] <- 2 * gs * (x[n] - x[n - 1]^2)
    g
}

pm <- rbind(rep(1,4), rep(pi, 4), rep(-2,4), rep(0,4), rep(20,4))
pm <- as.matrix(pm)
cat("multistart matrix:\n")
print(pm)

ans <- multistart(pm, fnR, grR, method="Rvmmin", control=list(trace=0))
ans
```

`opm` *General-purpose optimization*

Description

General-purpose optimization wrapper function that calls multiple other R tools for optimization, including the existing optim() function tools.

Because SANN does not return a meaningful convergence code (conv), `opm()` does not call the SANN method, but it can be invoked in `optimr()`.

There is a pseudo-method "ALL" that runs all available methods. Note that this is upper-case. This function is a replacement for optimx() from the optimx package that calls the optimr() function for each solver in the `method` list.

Usage

```
opm(par, fn, gr=NULL, hess=NULL, lower=-Inf, upper=Inf,
            method=c("Nelder-Mead","BFGS"), 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 a 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`, whatever default actions are supplied by the methods specified will be used. However, some methods REQUIRE a gradient function, so will fail in this case. `opm()` will generally return with `convergence` set to 9998 for such methods. If 'gr' is a character string, this character string will be taken to be the name of an available gradient approximation function. Examples are "grfwd", "grback", "grcentral" and "grnd", with the last name referring to the default method of package `numDeriv`.
`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. These are vectors.
`method`	A vector of the methods to be used, each as a character string. Note that this is an important change from optim() that allows just one method to be specified. See 'Details'. If `method` has just one element, `"ALL"` (capitalized), all available and appropriate methods will be tried.
`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.

For details of how `opm()` calls the methods, see the documentation and code for `optimr()`. The documentation and code for individual methods may also be useful. Note that some simplification of the calls may have been necessary, for example, to provide reasonable default values for method controls.

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.)

`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`.

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

`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. This is equivalent to setting `method="ALL"`

`kkt` =FALSE if we do NOT want to test the Kuhn, Karush, Tucker optimality conditions. The default is generally 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 `hessian` is TRUE, this overrides control `kkt`.

`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

`dowarn` = FALSE if we want to suppress warnings generated by `opm()` or `optimr()`. Default is TRUE.

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

There may be `control` elements that apply only to some of the methods. Using these may or may not "work" with `opm()`, and errors may occur with methods for which the controls have no meaning.

However, it should be possible to call the underlying `optimr()` function with these method-specific controls.

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.)

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.

`9998` indicates that the method has been called with a NULL 'gr' function, and the method requires that such a function be supplied.

`9999` indicates the method has failed.

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;

`scalechk()` 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 possible 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<-opm(startx,fn=genrose.f,gr=genrose.g, hess=genrose.h,
    control=list(all.methods=TRUE, save.failures=TRUE, trace=1), 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
```

optchk — *General-purpose optimization*

Description

A wrapper function that attempts to check the objective function, and optionally the gradient and hessian functions, supplied by the user for optimization. It also tries to check the scale of the parameters and bounds to see if they are reasonable.

Usage

```
optchk(par, fn, gr=NULL, hess=NULL, lower=-Inf, upper=Inf,
             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.
`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.

`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.

While it can be envisaged that a user would have an analytic hessian but not an analytic gradient, we do NOT permit the user to test the hessian in this situation.

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.)

Value

A list of the following items:

grOK TRUE if the analytic gradient and a numerical approximation via `numDeriv` agree within the `control$grtesttol` as per the `R` code in function `grchk`. `NULL` if no analytic gradient function is provided.

hessOK TRUE if the analytic hessian and a numerical approximation via `numDeriv::jacobian` agree within the `control$hesstesttol` as per the `R` code in function `hesschk`. NULL if no analytic hessian or no analytic gradient is provided. Note that since an analytic gradient must be available for this test, we use the Jacobian of the gradient to compute the Hessian to avoid one level of differencing, though the `hesschk` function can work without the gradient.

scalebad TRUE if the larger of the `scaleratios` exceeds `control$scaletol`

scaleratios A vector of the parameter and bounds scale ratios. See the function code of `scalechk` for the computation of these values.

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/.

Examples

```
fr <- function(x) {   ## Rosenbrock Banana function
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
    x1 <- x[1]
    x2 <- x[2]
    c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
       200 *      (x2 - x1 * x1))
}

myctrl<- ctrldefault(2)
myctrl$trace <- 3
mychk <- optchk(par=c(-1.2,1), fr, grr, lower=rep(-10,2), upper=rep(10,2), control=myctrl)
cat("result of optchk\n")
print(mychk)
```

`optimr` *General-purpose optimization*

Description

General-purpose optimization wrapper function that calls other R tools for optimization, including the existing optim() function. `optimr` also tries to unify the calling sequence to allow a number of tools to use the same front-end, in fact using the calling sequence of the R function `optim()`.

Usage

```
optimr(par, fn, gr=NULL, hess=NULL, lower=-Inf, upper=Inf,
            method=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`, whatever default action is specified for the chosen method for the case of a null gradient code is used. For many methods, this is a finite-difference approximation, but some methods require user input for the gradient and will fail otherwise. In such cases, we try to return `convergence` of 9998. If 'gr' is a character string, then that string is taken as the name of a gradient approximation function, for example, "grfwd", "grback" and "grcentral" for standard forward, backward and central approximations. (See the code in package `optextras` for details.) Method "grnd" uses the `grad()` function from package `numDeriv`.
`hess`	A function to return (as a matrix) the hessian 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. A small set of methods can handle masks, that is, fixed parameters, and these can be specified by making the lower and upper bounds equal to the starting value. (It is possible that the starting value could be different from the lower/upper bounds set, but this behaviour has NOT yet been defined and users are cautioned.)
`method`	A character string giving the name of the optimization method to be applied. See the list `allmeth` in file `ctrldefault.R` which is part of this package.
`hessian`	A logical control that if TRUE forces the computation of an approximation to the Hessian at the final set of parameters. Note that this will NOT necessarily use the same approximation as may be provided by the method called. Instead, the function `hessian()` from package `numDeriv` is used if no gradient `gr` is supplied, else the function `jacobian()` from `numDeriv` is applied to the gradient function `gr`.
`control`	A list of control parameters. See 'Details'.
`...`	Further arguments to be passed to `fn` and `gr` if needed for computation of these quantities; otherwise, further arguments are not used.

Details

Note that arguments after `...` should 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 mechanisms simultaneously.

Possible method choices are specified by the list `allmeth` in the file `ctrldefault.R` which is part of this package. Fewer methods are available in packge `optimr` on CRAN than package `optimrx` which is NOT on CRAN to avoid issues if packages on which function `optimr()` is dependent become unavailable.

If no method is specified, the method specified by `defmethod` in file `ctrldefault.R` (which is part of this package) will be attempted.

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

While methods from the base R function `optim()` can be used recursively, and for a single parameter as well as many, this may not be true for other methods in `optimrx`. `optim` also accepts a zero-length `par`, and just evaluates the function with that argument.

Generally, you are on your own if you choose to apply constructs mentioned in the above two paragraphs.

For details of methods, please consult the documentation of the individual methods. (The NAMESPACE file lists the packages from which functions are imported.) However, method `"hjn"` is a conservative implementation of a Hooke and Jeeves (1961) and is part of this package. It is provided as a simple example of a very crude optimization method; it is NOT intended as a production method, but may be useful for didactic purposes.

The `control` argument is a list that can supply any of the components in the file `ctrldefault.R` which is part of this package. It may supply others that are useful or required for particular methods, but users are warned to be careful to ensure that extraneous or incorrect components and values are not passed.

Note that some `control` elements apply only to some of methods. See individual packages for details.

Any names given to `par` will be copied to the vectors passed to `fn` and `gr`. Apparently no other attributes of `par` are copied over, but this may need to be verified, especially if parameters are passed to non-R routines.

Value

A list with components:

par The best set of parameters found.

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

counts A two-element integer vector giving the number of calls to 'fn' and 'gr' respectively. 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.

convergence An integer code. '0' indicates successful completion. The documentation for function `opm()` gives some other possible values and their meaning.

message A character string giving any additional information returned by the optimizer, or 'NULL'.

hessian If requested, an approximation to the hessian of 'fn' at the final parameters.

Source

See the manual pages for `optim()` and the packages listed in NAMESPACE.

References

See the manual pages for `optim()` and the packages listed in NAMESPACE.

Hooke R. and Jeeves, TA (1961). Direct search solution of numerical and statistical problems. Journal of the Association for Computing Machinery (ACM). 8 (2): 212–229.

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/.

Nocedal J, and Wright SJ (1999). Numerical optimization. New York: Springer. 2nd Edition 2006.

`polyopt` *General-purpose optimization - sequential application of methods*

Description

Multiple minimization methods are applied in sequence to a single problem, with the output parameters of one method being used to start the next.

Usage

```
polyopt(par, fn, gr=NULL, lower=-Inf, upper=Inf,
            methcontrol=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?
`lower, upper`	Bounds on the variables for methods such as `"L-BFGS-B"` that can handle box (or bounds) constraints.
`methcontrol`	An data frame of which each row gives an optimization method, a maximum number of iterations and a maximum number of function evaluations allowed for that method. Each method will be executed in turn until either the maximum iterations or function evaluations are completed, whichever is first. The next method is then executed starting with the best parameters found so far, else the function exits.
`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.

See `optimr()` for other details.

Note that this function does not (yet?) make use of a hess function to compute the hessian.

Value

An array with one row per method. Each row contains:

`par`	The best set of parameters found for the method in question.
`value`	The value of ‘fn’ corresponding to ‘par’.
`counts`	A two-element integer vector giving the number of calls to ‘fn’ and ‘gr’ respectively. 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.
`convergence`	An integer code. ‘0’ indicates successful completion
`message`	A character string giving any additional information returned by the optimizer, or ‘NULL’.
`hessian`	Always NULL for this routine.

Source

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

Examples

```
fnR <- function (x, gs=100.0)
{
    n <- length(x)
    x1 <- x[2:n]
    x2 <- x[1:(n - 1)]
    sum(gs * (x1 - x2^2)^2 + (1 - x2)^2)
}
grR <- function (x, gs=100.0)
{
    n <- length(x)
    g <- rep(NA, n)
    g[1] <- 2 * (x[1] - 1) + 4*gs * x[1] * (x[1]^2 - x[2])
    if (n > 2) {
        ii <- 2:(n - 1)
        g[ii] <- 2 * (x[ii] - 1) + 4 * gs * x[ii] * (x[ii]^2 - x[ii +
            1]) + 2 * gs * (x[ii] - x[ii - 1]^2)
    }
    g[n] <- 2 * gs * (x[n] - x[n - 1]^2)
    g
}

x0 <- rep(pi, 4)
mc <- data.frame(method=c("Nelder-Mead","Rvmmin"), maxit=c(1000, 100), maxfeval= c(1000, 1000))

ans <- polyopt(x0, fnR, grR, methcontrol=mc, control=list(trace=0))
ans
mc <- data.frame(method=c("Nelder-Mead","Rvmmin"), maxit=c(100, 100), maxfeval= c(100, 1000))

ans <- polyopt(x0, fnR, grR, methcontrol=mc, control=list(trace=0))
ans

mc <- data.frame(method=c("Nelder-Mead","Rvmmin"), maxit=c(10, 100), maxfeval= c(10, 1000))

ans <- polyopt(x0, fnR, grR, methcontrol=mc, control=list(trace=0))
```

ans

Index