Package 'optimx'

June 13, 2021

Version 2021-6.12

Date 2021-06-12

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 call to several function minimization codes in R in a single
statement. These methods handle smooth, possibly box constrained functions
of several or many parameters. Note that function 'optimr()' was prepared to
simplify the incorporation of minimization codes going forward. Also implements some
utility codes and some extra solvers, including safeguarded Newton methods.
Many methods previously separate are now included here.
This is the version for CRAN.

License GPL-2

LazyLoad Yes

Imports numDeriv

NeedsCompilation no

Suggests knitr, rmarkdown, markdown, setRNG, BB, ucminf, minqa,
dfoptim, lbfgsb3c, lbfgs, subplex

VignetteBuilder knitr

Repository CRAN

Date/Publication 2021-06-13 18:10:02 UTC

R topics documented:

`optimx-package` *A replacement and extension of the optim() function, plus various optimization tools*

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

The three functions ufn, ugr and uhess wrap corresponding user functions fn, gr, and hess so that these functions can be executed safely (via try()) and also so parameter or function scaling can be applied. The wrapper functions also allow for maximization of functions (via minimization of the negative of the function) using the logical parameter `maximize`.

There are three test functions, fnchk, grchk, and hesschk, to allow the user function to be tested for validity and correctness. However, no set of tests is exhaustive, and extensions and improvements are welcome. The package numDeriv is used for generation of numerical approximations to derivatives.

Details

Index:

```
axsearch     Perform an axial search optimality check
bmchk        Check bounds and masks for parameter constraints
bmstep       Compute the maximum step along a search direction.
fnchk        Test validity of user function
gHgen        Compute gradient and Hessian as a given
             set of parameters
gHgenb       Compute gradient and Hessian as a given
             set of parameters appying bounds and masks
grback       Backward numerical gradient approximation
grcentral    Central numerical gradient approximation
grchk        Check that gradient function evaluation
             matches numerical gradient
grfwd        Forward numerical gradient approximation
grnd         Gradient approximation using \code{numDeriv}
hesschk      Check that Hessian function evaluation
             matches numerical approximation
kktchk         Check the Karush-Kuhn-Tucker optimality conditions
scalechk   Check scale of initial parameters and bounds
optsp        An environment to hold some globally useful items
             used by optimization programs
proptimr     compact output of optimr() result object
```

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.

`axsearch` *Perform axial search around a supposed minimum and provide diagnostics*

Description

Nonlinear optimization problems often terminate at points in the parameter space that are not satisfactory optima. This routine conducts an axial search, stepping forward and backward along each parameter and computing the objective function. This allows us to compute the `tilt` and `radius of curvature` or `roc` along that parameter axis.

`axsearch` assumes that one is MINIMIZING the function `fn`. While we believe that it will work using the wrapper `ufn` from this package with the 'maximize=TRUE' setting, we believe it is much safer to write your own function that is to be minimized. That is minimize (-1)*(function to be maximized). All discussion here is in terms of minimization.

Axial search may find parameters with a function value lower than that at the supposed minimum, i.e., lower than `fmin`.

In this case `axsearch` exits immediately with the new function value and parameters. This can be used to restart an optimizer, as in the optimx wrapper.

Usage

```
axsearch(par, fn=NULL, fmin=NULL, lower=NULL, upper=NULL, bdmsk=NULL, trace=0, ...)
```

Arguments

`par`	A numeric vector of values of the optimization function parameters that are at a supposed minimum.
`fn`	The user objective function
`fmin`	The value of the objective function at the parameters `par`. ?? what if fmin==NULL?
`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. Partly for historical reasons, we use the same array during the progress of optimization as an indicator that a parameter is at a lower bound (bdmsk element set to -3) or upper bound (-1).
`trace`	If trace>0, then local output is enabled.
`...`	Extra arguments for the user function.

Details

The axial search MAY give a lower function value, in which case, one can restart. Its primary use is in presenting some features of the function surface in the tilt and radius of curvature measures returned. However, better measures should be possible, and this function should be regarded as largely experimental.

Value

A list with components:

bestfn	The lowest (best) function value found (??maximize??) during the axial search, else the original fmin value. (This is actively set in that case.)
par	The vector of parameters at the best function value.
details	A data frame reporting the original parameters, the forward step and backward step function values, the size of the step taken for a particular parameter, the tilt and the roc (radius of curvature). Some elements will be NA if we find a lower function value during the axial search.

Examples

```
####################
# require(optimx)
# Simple bounds test for n=4
bt.f<-function(x){
  sum(x*x)
}

bt.g<-function(x){
  gg<-2.0*x
}

n<-4
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)

cat("lower bounds:")
print(lower)
cat("start:       ")
print(xx)
cat("upper bounds:")
print(upper)

abtrvm <- list() # ensure we have the structure

cat("Rvmmin \n\n")
# Note: trace set to 0 below. Change as needed to view progress.

# Following can be executed if package optimx available
# abtrvm <- optimr(xx, bt.f, bt.g, lower=lower, upper=upper, method="Rvmmin",
#                 control=list(trace=0))
# Note: use lower=lower etc. because there is a missing hess= argument
# print(abtrvm)

abtrvm$par <- c(0.00, 0.75, 1.50, 2.25)
abtrvm$value <- 7.875
```

```
cat("Axial search")
axabtrvm <- axsearch(abtrvm$par, fn=bt.f, fmin=abtrvm$value, lower, upper, bdmsk=NULL,
                     trace=0)
print(axabtrvm)

abtrvm1 <- list() # set up structure
# Following can be executed if package optimx available
# cat("Now force an early stop\n")
# abtrvm1 <- optimr(xx, bt.f, bt.g, lower=lower, upper=upper, method="Rvmmin",
#                   control=list(maxit=1, trace=0))
# print(abtrvm1)

abtrvm1$value <- 8.884958
abtrvm1$par <- c(0.625, 1.625, 2.625, 3.625)

cat("Axial search")
axabtrvm1 <- axsearch(abtrvm1$par, fn=bt.f, fmin=abtrvm1$value, lower, upper, bdmsk=NULL,
                      trace=0)
print(axabtrvm1)

cat("Do NOT try axsearch() with maximize\n")
```

---

bmchk *Check bounds and masks for parameter constraints used in nonlinear optimization*

Description

Nonlinear optimization problems often have explicit or implicit upper and lower bounds on the parameters of the function to be miminized or maximized. These are called bounds or box constraints. Some of the parameters may be fixed for a given problem or for a temporary trial. These fixed, or masked, paramters are held at one value during a specific 'run' of the optimization.

It is possible that the bounds are inadmissible, that is, that at least one lower bound exceeds an upper bound. In this case we set the flag `admissible` to FALSE.

Parameters that are outside the bounds are moved to the nearest bound and the flag `parchanged` is set TRUE. However, we DO NOT change masked parameters, and they may be outside the bounds. This is an implementation choice, since it may be useful to test objective functions at point outside the bounds.

The package bmchk is essentially a test of the R function bmchk(), which is likely to be incorporated within optimization codes.

Usage

```
bmchk(par, lower=NULL, upper=NULL, bdmsk=NULL, trace=0, tol=NULL, shift2bound=TRUE)
```

Arguments

par	A numeric vector of starting values of the optimization function parameters.
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. Partly for historical reasons, we use the same array during the progress of optimization as an indicator that a parameter is at a lower bound (bdmsk element set to -3) or upper bound (-1).
trace	An integer that controls whether diagnostic information is displayed. A positive value displays information, 0 (default) does not.
tol	If provided, is used to detect a MASK, that is, lower=upper for some parameter.
shift2bound	If TRUE, non-masked paramters outside bounds are adjusted to the nearest bound. We then set parchanged = TRUE which implies the original parameters were infeasible.

Details

The bmchk function will check that the bounds exist and are admissible, that is, that there are no lower bounds that exceed upper bounds.

There is a check if lower and upper bounds are very close together, in which case a mask is imposed and maskadded is set TRUE. NOTE: it is generally a VERY BAD IDEA to have bounds close together in optimization, but here we use a tolerance based on the double precision machine epsilon. Thus it is not a good idea to rely on bmchk() to test if bounds constraints are well-posed.

Value

A list with components:

bvec	The vector of parameters, possibly adjusted for bounds. Parameters outside bounds are adjusted to the nearest bound.
bdmsk	adjusted input masks
bchar	indicator for humans – "-","L","F","U","+","M" for out-of-bounds-low, lower bound, free, upper bound, out-of-bounds-high, masked (fixed)
lower	(adjusted) lower bounds. If upper-lower<tol, we create a mask rather than leave bounds. In this case we could eliminate the bounds. At the moment, this change is NOT made, but a commented line of code is present in the file bmchk.R.
upper	(adjusted) upper bounds
nolower	TRUE if no lower bounds, FALSE otherwise
noupper	TRUE if no upper bounds, FALSE otherwise
bounds	TRUE if there are any bounds, FALSE otherwise
admissible	TRUE if bounds are admissible, FALSE otherwise This means no lower bound exceeds an upper bound. That is the bounds themselves are sensible. This condition has nothing to do with the starting parameters.

maskadded	TRUE when a mask has been added because bounds are very close or equal, FALSE otherwise. See the code for the implementation.
parchanged	TRUE if parameters are changed by bounds, FALSE otherswise. Note that parchanged = TRUE implies the input parameter values were infeasible, that is, violated the bounds constraints.
feasible	TRUE if parameters are within or on bounds, FALSE otherswise.
onbound	TRUE if any parameter is on a bound, FALSE otherswise. Note that parchanged = TRUE implies onbound = TRUE, but this is not used inside the function. This output value may be important, for example, in using the optimization function nmkb from package dfoptim.

Examples

```
####################

cat("25-dimensional box constrained function\n")
flb <- function(x)
    { p <- length(x); sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2) }

start<-rep(2, 25)
cat("\n start:")
print(start)
lo<-rep(2,25)
cat("\n lo:")
print(lo)
hi<-rep(4,25)
cat("\n hi:")
print(hi)
bt<-bmchk(start, lower=lo, upper=hi, trace=1)
print(bt)
```

---

bmstep *Compute the maximum step along a search direction.*

Description

Nonlinear optimization problems often have explicit or implicit upper and lower bounds on the parameters of the function to be miminized or maximized. These are called bounds or box constraints. Some of the parameters may be fixed for a given problem or for a temporary trial. These fixed, or masked, paramters are held at one value during a specific 'run' of the optimization.

The bmstep() function computes the maximum step possible (which could be infinite) along a particular search direction from current parameters to bounds.

Usage

```
bmstep(par, srchdirn, lower=NULL, upper=NULL, bdmsk=NULL, trace=0)
```

Arguments

par	A numeric vector of starting values of the optimization function parameters.
srchdirn	A numeric vector giving the search direction.
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. Partly for historical reasons, we use the same array during the progress of optimization as an indicator that a parameter is at a lower bound (bdmsk element set to -3) or upper bound (-1).
trace	An integer that controls whether diagnostic information is displayed. A positive value displays information, 0 (default) does not.

Details

The bmstep function will compute and return (as a double or Inf) the maximum step to the bounds.

Value

A double precision value or Inf giving the maximum step to the bounds.

Examples

```
######################
xx <- c(1, 1)
lo <- c(0, 0)
up <- c(100, 40)
sdir <- c(4,1)
bm <- c(1,1) # both free
ans <- bmstep(xx, sdir, lo, up, bm, trace=1)
# stepsize
print(ans)
# distance
print(ans*sdir)
# New parameters
print(xx+ans*sdir)
```

---

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.

Examples

```
allmeth <- c("Rvmmin", "nlminb","ipopttest")
allpkg <- c("Rvmmin", "stats","ipoptr")

print(checksolver("nlminb", allmeth, allpkg))
# If Rvmmin NOT available, get msg that PACKAGE not available.
print(checksolver("Rvmmin", allmeth, allpkg))
# Get message that SOLVER not found
print(checksolver("notasolver", allmeth, allpkg))
```

---

coef *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=0))
print(coef(ans))

ansx <- optimx(fn = function(x) sum(x*x), par = 1:2, control=list(all.methods=TRUE, trace=0))
print(coef(ansx))


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

## End(Not run)
```

---

ctrldefault *set control defaults*

Description

Set control defaults.

Usage

```
ctrldefault(npar)

dispdefault(ctrl)
```

Arguments

npar	Number of parameters to optimize.
ctrl	A list (likely generated by ‘ctrldefault‘) of default settings to ‘optimx‘.

Value

ctrldefault returns the default control settings for optimization tools.

dispdefault provides a compact display of the contents of a control settings list.

fnchk *Run tests, where possible, on user objective function*

Description

fnchk checks a user-provided R function, ffn.

Usage

```
fnchk(xpar, ffn, trace=0, ... )
```

Arguments

xpar	the (double) vector of parameters to the objective funcion
ffn	a user-provided function to compute the objective function
trace	set >0 to provide output from fnchk to the console, 0 otherwise
...	optional arguments passed to the objective function.

Details

fnchk attempts to discover various errors in function setup in user-supplied functions primarily intended for use in optimization calculations. There are always more conditions that could be tested!

Value

The output is a list consisting of list(fval=fval, infeasible=infeasible, excode=excode, msg=msg)

fval	The calculated value of the function at parameters xpar if the function can be evaluated.
infeasible	FALSE if the function can be evaluated, TRUE if not.
excode	An exit code, which has a relationship to
msg	A text string giving information about the result of the function check: Messages and the corresponding values of excode are:

- fnchk OK; excode = 0; infeasible = FALSE
- Function returns INADMISSIBLE; excode = -1; infeasible = TRUE
- Function returns a vector not a scalar; excode = -4; infeasible = TRUE
- Function returns a list not a scalar; excode = -4; infeasible = TRUE
- Function returns a matrix list not a scalar; excode = -4; infeasible = TRUE
- Function returns an array not a scalar; excode = -4; infeasible = TRUE
- Function returned not length 1, despite not vector, matrix or array; excode = -4; infeasible = TRUE
- Function returned non-numeric value; excode = 0; excode = -1; infeasible = TRUE
- Function returned Inf or NA (non-computable); excode = -1; infeasible = TRUE

Author(s)

John C. Nash <nashjc@uottawa.ca>

Examples

```
# Want to illustrate each case.
# Ben Bolker idea for a function that is NOT scalar
# rm(list=ls())
# library(optimx)
sessionInfo()
benbad<-function(x, y){
  # y may be provided with different structures
  f<-(x-y)^2
} # very simple, but ...

y<-1:10
x<-c(1)
cat("fc01: test benbad() with y=1:10, x=c(1)\n")
fc01<-fnchk(x, benbad, trace=4, y)
print(fc01)

y<-as.vector(y)
cat("fc02: test benbad() with y=as.vector(1:10), x=c(1)\n")
fc02<-fnchk(x, benbad, trace=1, y)
print(fc02)

y<-as.matrix(y)
cat("fc03: test benbad() with y=as.matrix(1:10), x=c(1)\n")
fc03<-fnchk(x, benbad, trace=1, y)
print(fc03)

y<-as.array(y)
cat("fc04: test benbad() with y=as.array(1:10), x=c(1)\n")
fc04<-fnchk(x, benbad, trace=1, y)
print(fc04)

y<-"This is a string"
cat("test benbad() with y a string, x=c(1)\n")
fc05<-fnchk(x, benbad, trace=1, y)
print(fc05)

cat("fnchk with Rosenbrock\n")
fr <- function(x) {   ## Rosenbrock Banana function
  x1 <- x[1]
  x2 <- x[2]
  100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
xtrad<-c(-1.2,1)
ros1<-fnchk(xtrad, fr, trace=1)
print(ros1)
npar<-2
opros<-list2env(list(fn=fr, gr=NULL, hess=NULL, MAXIMIZE=FALSE, PARSCALE=rep(1,npar), FNSCALE=1,
```

```
                      KFN=0, KGR=0, KHESS=0, dots=NULL))
uros1<-fnchk(xtrad, fr, trace=1)
print(uros1)
```

---

gHgen *Generate gradient and Hessian for a function at given parameters.*

Description

gHgen is used to generate the gradient and Hessian of an objective function used for optimization. If a user-provided gradient function gr is available it is used to compute the gradient, otherwise package numDeriv is used. If a user-provided Hessian function hess is available, it is used to compute a Hessian. Otherwise, if gr is available, we use the function jacobian() from package numDeriv to compute the Hessian. In both these cases we check for symmetry of the Hessian. Computational Hessians are commonly NOT symmetric. If only the objective function fn is provided, then the Hessian is approximated with the function hessian from package numDeriv which guarantees a symmetric matrix.

Usage

```
gHgen(par, fn, gr=NULL, hess=NULL,
    control=list(ktrace=0), ...)
```

Arguments

par	Set of parameters, assumed to be at a minimum of the function fn.
fn	Name of the objective function.
gr	(Optional) function to compute the gradient of the objective function. If present, we use the Jacobian of the gradient as the Hessian and avoid one layer of numerical approximation to the Hessian.
hess	(Optional) function to compute the Hessian of the objective function. This is rarely available, but is included for completeness.
control	A list of controls to the function. Currently asymptol (default of 1.0e-7 which tests for asymmetry of Hessian approximation (see code for details of the test); ktrace, a logical flag which, if TRUE, monitors the progress of gHgen (default FALSE), and stoponerror, defaulting to FALSE to NOT stop when there is an error or asymmetry of Hessian. Set TRUE to stop.
...	Extra data needed to compute the function, gradient and Hessian.

Details

None

Value

ansout a list of four items,

- gn The approximation to the gradient vector.
- Hn The approximation to the Hessian matrix.
- gradOK TRUE if the gradient has been computed acceptably. FALSE otherwise.
- hessOK TRUE if the gradient has been computed acceptably and passes the symmetry test. FALSE otherwise.
- nbm Always 0. The number of active bounds and masks. Present to make function consistent with gHgenb.

Examples

```
# 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)
```

```
}

trad<-c(-1.2,1)
ans100fgh<-  gHgen(trad, genrose.f, gr=genrose.g, hess=genrose.h,
      control=list(ktrace=1))
print(ans100fgh)
ans100fg<-  gHgen(trad, genrose.f, gr=genrose.g,
      control=list(ktrace=1))
print(ans100fg)
ans100f<-  gHgen(trad, genrose.f, control=list(ktrace=1))
print(ans100f)
ans10fgh<-   gHgen(trad, genrose.f, gr=genrose.g, hess=genrose.h,
      control=list(ktrace=1), gs=10)
print(ans10fgh)
ans10fg<-   gHgen(trad, genrose.f, gr=genrose.g,
      control=list(ktrace=1), gs=10)
print(ans10fg)
ans10f<-   gHgen(trad, genrose.f, control=list(ktrace=1), gs=10)
print(ans10f)
```

---

gHgenb *Generate gradient and Hessian for a function at given parameters.*

Description

gHgenb is used to generate the gradient and Hessian of an objective function used for optimization. If a user-provided gradient function gr is available it is used to compute the gradient, otherwise package numDeriv is used. If a user-provided Hessian function hess is available, it is used to compute a Hessian. Otherwise, if gr is available, we use the function jacobian() from package numDeriv to compute the Hessian. In both these cases we check for symmetry of the Hessian. Computational Hessians are commonly NOT symmetric. If only the objective function fn is provided, then the Hessian is approximated with the function hessian from package numDeriv which guarantees a symmetric matrix.

Usage

```
gHgenb(par, fn, gr=NULL, hess=NULL, bdmsk=NULL, lower=NULL, upper=NULL,
    control=list(ktrace=0), ...)
```

Arguments

par	Set of parameters, assumed to be at a minimum of the function fn.
fn	Name of the objective function.
gr	(Optional) function to compute the gradient of the objective function. If present, we use the Jacobian of the gradient as the Hessian and avoid one layer of numerical approximation to the Hessian.

hess	(Optional) function to compute the Hessian of the objective function. This is rarely available, but is included for completeness.
bdmsk	An integer vector of the same length as par. When an element of this vector is 0, the corresponding parameter value is fixed (masked) during an optimization. Non-zero values indicate a parameter is free (1), at a lower bound (-3) or at an upper bound (-1), but this routine only uses 0 values.
lower	Lower bounds for parameters in par.
upper	Upper bounds for parameters in par.
control	A list of controls to the function. Currently asymptol (default of 1.0e-7 which tests for asymmetry of Hessian approximation (see code for details of the test); ktrace, a logical flag which, if TRUE, monitors the progress of gHgenb (default FALSE), and stoponerror, defaulting to FALSE to NOT stop when there is an error or asymmetry of Hessian. Set TRUE to stop.
...	Extra data needed to compute the function, gradient and Hessian.

Details

None

Value

ansout a list of four items,

- gn The approximation to the gradient vector.
- Hn The approximation to the Hessian matrix.
- gradOK TRUE if the gradient has been computed acceptably. FALSE otherwise.
- hessOK TRUE if the gradient has been computed acceptably and passes the symmetry test. FALSE otherwise.
- nbm The number of active bounds and masks.

Examples

```
require(numDeriv)
# 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)
}


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

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

parx<-rep(1,4)
lower<-rep(-10,4)
upper<-rep(10,4)
bdmsk<-c(1,1,0,1) # masked parameter 3
fval<-genrose.f(parx)
gval<-genrose.g(parx)
Ahess<-genrose.h(parx)
gennog<-gHgenb(parx,genrose.f)
cat("results of gHgenb for genrose without gradient code at ")
print(parx)
print(gennog)
cat("compare to g =")
print(gval)
cat("and Hess\n")
print(Ahess)
```

```
cat("\n\n")
geng<-gHgenb(parx,genrose.f,genrose.g)
cat("results of gHgenb for genrose at ")
print(parx)
print(gennog)
cat("compare to g =")
print(gval)
cat("and Hess\n")
print(Ahess)
cat("****************************************\n")
parx<-rep(0.9,4)
fval<-genrose.f(parx)
gval<-genrose.g(parx)
Ahess<-genrose.h(parx)
gennog<-gHgenb(parx,genrose.f,control=list(ktrace=TRUE), gs=9.4)
cat("results of gHgenb with gs=",9.4," for genrose without gradient code at ")
print(parx)
print(gennog)
cat("compare to g =")
print(gval)
cat("and Hess\n")
print(Ahess)
cat("\n\n")
geng<-gHgenb(parx,genrose.f,genrose.g, control=list(ktrace=TRUE))
cat("results of gHgenb for genrose at ")
print(parx)
print(gennog)
cat("compare to g =")
print(gval)
cat("and Hess\n")
print(Ahess)
gst<-5
cat("\n\nTest with full calling sequence and gs=",gst,"\n")
gengall<-gHgenb(parx,genrose.f,genrose.g,genrose.h, control=list(ktrace=TRUE),gs=gst)
print(gengall)


top<-25
x0<-rep(2,4)
cat("\n\nTest for maximization and top=",top,"\n")
cat("Gradient and Hessian will have sign inverted")
maxt<-gHgen(x0, maxfn, control=list(ktrace=TRUE), top=top)
print(maxt)

cat("test against negmaxfn\n")
gneg <- grad(negmaxfn, x0)
Hneg<-hessian(negmaxfn, x0)
# gdiff<-max(abs(gneg-maxt$gn))/max(abs(maxt$gn))
# Hdiff<-max(abs(Hneg-maxt$Hn))/max(abs(maxt$Hn))
# explicitly change sign
gdiff<-max(abs(gneg-(-1)*maxt$gn))/max(abs(maxt$gn))
Hdiff<-max(abs(Hneg-(-1)*maxt$Hn))/max(abs(maxt$Hn))
cat("gdiff = ",gdiff,"  Hdiff=",Hdiff,"\n")
```

grback *Backward difference numerical gradient approximation.*

Description

grback computes the backward difference approximation to the gradient of user function userfn.

Usage

```
grback(par, userfn, fbase=NULL, env=optsp, ...)
```

Arguments

par	parameters to the user objective function userfn
userfn	User-supplied objective function
fbase	The value of the function at the parameters, else NULL. This is to save recomputing the function at this point.
env	Environment for scratchpad items (like deps for approximation control in this routine). Default optsp.
...	optional arguments passed to the objective function.

Details

Package:	grback
Depends:	R (>= 2.6.1)
License:	GPL Version 2.

Value

grback returns a single vector object df which approximates the gradient of userfn at the parameters par. The approximation is controlled by a global value optderiveps that is set when the package is attached.

Author(s)

John C. Nash

Examples

```
cat("Example of use of grback\n")

myfn<-function(xx, shift=100){
    ii<-1:length(xx)
    result<-shift+sum(xx^ii)
}

xx<-c(1,2,3,4)
ii<-1:length(xx)
print(xx)
gn<-grback(xx,myfn, shift=0)
print(gn)
ga<-ii*xx^(ii-1)
cat("compare to analytic gradient:\n")
print(ga)

cat("change the step parameter to 1e-4\n")
optsp$deps <- 1e-4
gn2<-grback(xx,myfn, shift=0)
print(gn2)
```

---

grcentral *Central difference numerical gradient approximation.*

Description

`grcentral` computes the central difference approximation to the gradient of user function `userfn`.

Usage

```
grcentral(par, userfn, fbase=NULL, env=optsp, ...)
```

Arguments

`par`	parameters to the user objective function userfn
`userfn`	User-supplied objective function
`fbase`	The value of the function at the parameters, else NULL. This is to save recomputing the function at this point.
`env`	Environment for scratchpad items (like `deps` for approximation control in this routine). Default `optsp`.
`...`	optional arguments passed to the objective function.

Details

Package:	grcentral
Depends:	R (>= 2.6.1)
License:	GPL Version 2.

Value

`grcentral` returns a single vector object `df` which approximates the gradient of userfn at the parameters par. The approximation is controlled by a global value `optderiveps` that is set when the package is attached.

Author(s)

John C. Nash

Examples

```
cat("Example of use of grcentral\n")

myfn<-function(xx, shift=100){
    ii<-1:length(xx)
    result<-shift+sum(xx^ii)
}
xx<-c(1,2,3,4)
ii<-1:length(xx)
print(xx)
gn<-grcentral(xx,myfn, shift=0)
print(gn)
ga<-ii*xx^(ii-1)
cat("compare to\n")
print(ga)
```

`grchk` *Run tests, where possible, on user objective function and (optionally) gradient and hessian*

Description

`grchk` checks a user-provided R function, `ffn`.

Usage

```
grchk(xpar, ffn, ggr, trace=0, testtol=(.Machine$double.eps)^(1/3), ...)
```

Arguments

xpar	parameters to the user objective and gradient functions ffn and ggr
ffn	User-supplied objective function
ggr	User-supplied gradient function
trace	set >0 to provide output from grchk to the console, 0 otherwise
testtol	tolerance for equality tests
...	optional arguments passed to the objective function.

Details

Package:	grchk
Depends:	R (>= 2.6.1)
License:	GPL Version 2.

numDeriv is used to numerically approximate the gradient of function ffn and compare this to the result of function ggr.

Value

grchk returns a single object gradOK which is true if the differences between analytic and approximated gradient are small as measured by the tolerance testtol.

This has attributes "ga" and "gn" for the analytic and numerically approximated gradients.

At the time of preparation, there are no checks for validity of the gradient code in ggr as in the function fnchk.

Author(s)

John C. Nash

Examples

```
# Would like examples of success and failure. What about "near misses"??
cat("Show how grchk works\n")
require(numDeriv)
# require(optimx)

jones<-function(xx){
  x<-xx[1]
  y<-xx[2]
  ff<-sin(x*x/2 - y*y/4)*cos(2*x-exp(y))
  ff<- -ff
}

jonesg <- function(xx) {
  x<-xx[1]
  y<-xx[2]
```

```
  gx <-  cos(x * x/2 - y * y/4) * ((x + x)/2) * cos(2 * x - exp(y)) -
    sin(x * x/2 - y * y/4) * (sin(2 * x - exp(y)) * 2)
  gy <- sin(x * x/2 - y * y/4) * (sin(2 * x - exp(y)) * exp(y)) - cos(x *
              x/2 - y * y/4) * ((y + y)/4) * cos(2 * x - exp(y))
  gg <- - c(gx, gy)
}

jonesg2 <- function(xx) {
  gx <- 1
  gy <- 2
  gg <- - c(gx, gy)
}


xx <- c(1, 2)

gcans <- grchk(xx, jones, jonesg, trace=1, testtol=(.Machine$double.eps)^(1/3))
gcans

gcans2 <- grchk(xx, jones, jonesg2, trace=1, testtol=(.Machine$double.eps)^(1/3))
gcans2
```

---

grfwd *Forward difference numerical gradient approximation.*

Description

grfwd computes the forward difference approximation to the gradient of user function userfn.

Usage

```
grfwd(par, userfn, fbase=NULL, env=optsp, ...)
```

Arguments

par	parameters to the user objective function userfn
userfn	User-supplied objective function
fbase	The value of the function at the parameters, else NULL. This is to save recomputing the function at this point.
env	Environment for scratchpad items (like deps for approximation control in this routine). Default optsp.
...	optional arguments passed to the objective function.

Details

Package:	grfwd
Depends:	R (>= 2.6.1)
License:	GPL Version 2.

Value

grfwd returns a single vector object df which approximates the gradient of userfn at the parameters par. The approximation is controlled by a global value optderiveps that is set when the package is attached.

Author(s)

John C. Nash

Examples

```
cat("Example of use of grfwd\n")

myfn<-function(xx, shift=100){
    ii<-1:length(xx)
    result<-shift+sum(xx^ii)
}
xx<-c(1,2,3,4)
ii<-1:length(xx)
print(xx)
gn<-grfwd(xx,myfn, shift=0)
print(gn)
ga<-ii*xx^(ii-1)
cat("compare to\n")
print(ga)
```

---

grnd *A reorganization of the call to numDeriv grad() function.*

Description

Provides a wrapper for the numDeriv approximation to the gradient of a user supplied objective function userfn.

Usage

```
grnd(par, userfn, ...)
```

Arguments

par	A vector of parameters to the user-supplied function fn
userfn	A user-supplied function
...	Other data needed to evaluate the user function.

Details

The Richardson method is used in this routine.

Value

grnd returns an approximation to the gradient of the function userfn

Examples

```
cat("Example of use of grnd\n")
require(numDeriv)
myfn<-function(xx, shift=100){
    ii<-1:length(xx)
    result<-shift+sum(xx^ii)
}
xx<-c(1,2,3,4)
ii<-1:length(xx)
print(xx)
gn<-grnd(xx,myfn, shift=0)
print(gn)
ga<-ii*xx^(ii-1)
cat("compare to\n")
print(ga)
```

---

hesschk — *Run tests, where possible, on user objective function and (optionally) gradient and hessian*

Description

hesschk checks a user-provided R function, ffn.

Usage

```
hesschk(xpar, ffn, ggr, hhess, trace=0, testtol=(.Machine$double.eps)^(1/3), ...)
```

Arguments

xpar	parameters to the user objective and gradient functions ffn and ggr
ffn	User-supplied objective function
ggr	User-supplied gradient function
hhess	User-supplied Hessian function
trace	set >0 to provide output from hesschk to the console, 0 otherwise
testtol	tolerance for equality tests
...	optional arguments passed to the objective function.

Details

Package:	hesschk
Depends:	R (>= 2.6.1)
License:	GPL Version 2.

numDeriv is used to compute a numerical approximation to the Hessian matrix. If there is no analytic gradient, then the hessian() function from numDeriv is applied to the user function ffn. Otherwise, the jacobian() function of numDeriv is applied to the ggr function so that only one level of differencing is used.

Value

The function returns a single object hessOK which is TRUE if the analytic Hessian code returns a Hessian matrix that is "close" to the numerical approximation obtained via numDeriv; FALSE otherwise.

hessOK is returned with the following attributes:

- "nullhess"Set TRUE if the user does not supply a function to compute the Hessian.
- "asym"Set TRUE if the Hessian does not satisfy symmetry conditions to within a tolerance. See the hesschk for details.
- "ha"The analytic Hessian computed at paramters xpar using hhess.
- "hn"The numerical approximation to the Hessian computed at paramters xpar.
- "msg"A text comment on the outcome of the tests.

Author(s)

John C. Nash

Examples

```
# 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)
}

trad<-c(-1.2,1)
ans100<-hesschk(trad, genrose.f, genrose.g, genrose.h, trace=1)
print(ans100)
ans10<-hesschk(trad, genrose.f, genrose.g, genrose.h, trace=1, gs=10)
print(ans10)
```

---

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()
####################
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

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")

######### One dimension test
```

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

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

---

kktchk *Check Kuhn Karush Tucker conditions for a supposed function minimum*

Description

Provide a check on Kuhn-Karush-Tucker conditions based on quantities already computed. Some of these used only for reporting.

Usage

```
kktchk(par, fn, gr, hess=NULL, upper=NULL, lower=NULL,
            maximize=FALSE, control=list(), ...)
```

Arguments

par	A vector of values for the parameters which are supposedly optimal.
fn	The objective function
gr	The gradient function
hess	The Hessian function
upper	Upper bounds on the parameters
lower	Lower bounds on the parameters
maximize	Logical TRUE if function is being maximized. Default FALSE.
control	A list of controls for the function
...	The dot arguments needed for evaluating the function and gradient and hessian

Details

kktchk computes the gradient and Hessian measures for BOTH unconstrained and bounds (and masks) constrained parameters, but the kkt measures are evaluated only for the constrained case.

Value

The output is a list consisting of

gmax	The absolute value of the largest gradient component in magnitude.
evratio	The ratio of the smallest to largest Hessian eigenvalue. Note that this may be negative.
kkt1	A logical value that is TRUE if we consider the first (i.e., gradient) KKT condition to be satisfied. WARNING: The decision is dependent on tolerances and scaling that may be inappropriate for some problems.
kkt2	A logical value that is TRUE if we consider the second (i.e., positive definite Hessian) KKT condition to be satisfied. WARNING: The decision is dependent on tolerances and scaling that may be inappropriate for some problems.
hev	The calculated hessian eigenvalues, sorted largest to smallest??
ngatend	The computed (unconstrained) gradient at the solution parameters.
nnatend	The computed (unconstrained) hessian at the solution parameters.

See Also

optim

Examples

```
cat("Show how kktc works\n")

# require(optimx)

jones<-function(xx){
  x<-xx[1]
  y<-xx[2]
  ff<-sin(x*x/2 - y*y/4)*cos(2*x-exp(y))
  ff<- -ff
}

jonesg <- function(xx) {
  x<-xx[1]
  y<-xx[2]
  gx <-  cos(x * x/2 - y * y/4) * ((x + x)/2) * cos(2 * x - exp(y)) -
    sin(x * x/2 - y * y/4) * (sin(2 * x - exp(y)) * 2)
  gy <- sin(x * x/2 - y * y/4) * (sin(2 * x - exp(y)) * exp(y)) - cos(x *
            x/2 - y * y/4) * ((y + y)/4) * cos(2 * x - exp(y))
  gg <- - c(gx, gy)
}

ans <- list() # to ensure structure available
# If optimx package available, the following can be run.
# xx<-0.5*c(pi,pi)
# ans <- optimr(xx, jones, jonesg, method="Rvmmin")
# ans
```

```
ans$par <- c(3.154083, -3.689620)

kkans <- kktchk(ans$par, jones, jonesg)
kkans
```

`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.
- `2` (Rvmmin) indicates that a point has been found with small gradient norm (< (1 + abs(fmin))*eps*eps).
- `3` (Rvmmin) indicates approx. inverse Hessian cannot be updated at steepest descent iteration (i.e., something very wrong).
- `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 `par`s, 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 https://www.jstatsoft.org/v43/i09/.

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

See Also

`spg`, `nlm`, `nlminb`, `bobyqa`, `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,
   method="ALL", control=list(save.failures=TRUE, trace=0), gs=10)
# Set trace=1 for output of individual solvers
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 https://www.jstatsoft.org/v43/i09/.

Nash JC (2014). On Best Practice Optimization Methods in R., *Journal of Statistical Software*, 60(2), 1-14., URL https://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. 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.

CAUTION: because there is a seldom-used parameter hess, you should NOT make a call like

ans <- optimr(start, myf, myg, lower, upper)

or you will likely get wrong results. Instead use

ans <- optimr(start, myf, myg, lower=lower, upper=upper)

NOTE: The default update formula for the "CG" option of `optim()` is `type=2` or Polak-Ribiere.

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.

References

See the manual pages for `optim()`.

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 https://www.jstatsoft.org/v43/i09/.

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

Examples

```
 # Simple Test Function 1:
tryfun.f = function(x) {
     fun <- sum(x^2 )
## if (trace) ... to be fixed
print(c(x = x, fun = fun))
     fun
}
tryfun.g = function(x) {
     grad<-2.0*x
     grad
}
tryfun.h = function(x) {
     n<-length(x)
     t<-rep(2.0,n)
     hess<-diag(t)
}
```

```
strt <- c(1,2,3)
ansfgh <- optimr(strt, tryfun.f, tryfun.g, tryfun.h, method="nlm",
     hessian=TRUE, control=list(trace=2))
proptimr(ansfgh) # compact output of result
```

---

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.

Note that package optimr allows solvers to be called individually by the optim() syntax, with the parscale control to scale parameters applicable to all methods. However, running multiple methods, or using the follow.on capability has been moved to separate routines in the optimr package.

Cautions:

1) Using some control list options with different or multiple methods may give unexpected results.

2) Testing the KKT conditions can take much longer than solving the optimization problem, especially when the number of parameters is large and/or analytic gradients are not available. Note that the default for the control kkt is TRUE.

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: https://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.

Note: Package `optimr` is a derivative of this package. It was developed initially to overcome maintenance difficulties with the current package related to avoiding confusion if some multiple options were specified together, and to allow the `optim()` function syntax to be used consistently, including the `parscale` control. However, this package does perform well, and is called by a number of popular other packages.

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 https://www.jstatsoft.org/v43/i09/.

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

See Also

`spg`, `nlm`, `nlminb`, `bobyqa`, `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
```

`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
```

---

proptimr *Compact display of an* optimr() *result object*

Description

proptimr displays the contents of a result computed by optimr().

Usage

```
proptimr(opres)
```

Arguments

opres	the object returned by function optimr()

Value

This function is intended for output only.

Author(s)

John C. Nash

Rcgmin — *An R implementation of a nonlinear conjugate gradient algorithm with the Dai / Yuan update and restart. Based on Nash (1979) Algorithm 22 for its main structure.*

Description

The purpose of Rcgmin is to minimize an unconstrained or bounds (box) and mask constrained function of many parameters by a nonlinear conjugate gradients method. 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.

Rcgmin is a wrapper that calls Rcgminu for unconstrained problems, else Rcgminb.

Usage

```
Rcgmin(par, fn, gr, lower, upper, bdmsk, control = list(), ...)
```

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.
gr	A function that returns the gradient of the objective at the supplied set of parameters par using auxiliary data in .... The first argument of fn must be par. This function returns the gradient as a numeric vector. If gr is not provided or is NULL, then the simple forward gradient code grfwd is used. However, we recommend carefully coded and checked analytic derivatives for Rcgmin. The use of numerical gradients for Rcgmin is discouraged. First, the termination test uses a size measure on the gradient, and numerical gradient approximations

can sometimes give results that are too large. Second, if there are bounds constraints, the step(s) taken to calculate the approximation to the derivative are NOT checked to see if they are out of bounds, and the function may be undefined at the evaluation point.

There is also the option of using the routines `grfwd`, `grback`, `grcentral` or `grnd`. The last of these calls the `grad()` function from package numDeriv. These are called by putting the name of the (numerical) gradient function in quotation marks, e.g.,

gr="grfwd"

to use the standard forward difference numerical approximation.

Note that all but the `grnd` routine use a stepsize parameter that can be redefined in a special scratchpad storage variable `deps`. The default is `deps = 1e-07`. However, redefining this is discouraged unless you understand what you are doing.

`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.

maxit A limit on the number of iterations (default 500). Note that this is used to compute a quantity `maxfeval`<-round(sqrt(n+1)*maxit) where n is the number of parameters to be minimized.

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 `Rcgmin` 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 square of the gradient. Default is 0 on input, which uses a value of tolgr = npar*npar*.Machine$double.eps in testing if crossprod(g) <= tolgr * (abs(fmin) + reltest). If the user supplies a value for `tol` that is non-zero, then that value is used for tolgr.

reltest=100 is only alterable by changing the code. fmin is the current best value found for the function minimum value.

Note that the scale of the gradient means that tests for a small gradient can easily be mismatched to a given problem. The defaults in Rcgmin are a "best guess".

`checkgrad` = TRUE if we want gradient function checked against numerical approximations. Default is FALSE.

`checkbounds` = TRUE if we want bounds verified. Default is TRUE.

The source code Rcgmin for R is likely to remain a work in progress for some time, so users should watch the console output.

As of 2011-11-21 the following controls have been REMOVED

usenumDeriv There is now a choice of numerical gradient routines. See argument gr.

maximize To maximize user_function, supply a function that computes (-1)*user_function. An alternative is to call Rcgmin via the package optimx, where the MAXIMIZE field of the OPCON structure in package optfntools is 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. '2' indicates initial point is infeasible.
message	A character string giving any additional information returned by the optimizer, or 'NULL'.
bdmsk	Returned index describing the status of bounds and masks at the proposed solution. Parameters for which bdmsk are 1 are unconstrained or "free", those with bdmsk 0 are masked i.e., fixed. For historical reasons, we indicate a parameter is at a lower bound using -3 or upper bound using -1.

References

Dai, Y. H. and Y. Yuan (2001). An efficient hybrid conjugate gradient method for unconstrained optimization. Annals of Operations Research 103 (1-4), 33–47.

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

Nash, J. C. and M. Walker-Smith (1987). Nonlinear Parameter Estimation: An Integrated System in BASIC. New York: Marcel Dekker. See https://www.nashinfo.com/nlpe.htm for a downloadable version of this plus some extras.

See Also

optim

Examples

```
####################
require(numDeriv)
## Rosenbrock Banana function
fr <- function(x) {
    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))
}

grn<-function(x){
    gg<-grad(fr, x)
}


ansrosenbrock0 <- Rcgmin(fn=fr,gr=grn, par=c(1,2))
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)
}

bt.g<-function(x){
  gg<-2.0*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<-Rcgmin(xx, bt.f, bt.g, lower, upper, bdmsk, control=list(trace=1))

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)
}
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
gg
}

# analytic gradient test
xx<-rep(pi,10)
lower<-NULL
upper<-NULL
bdmsk<-NULL
genrosea<-Rcgmin(xx,genrose.f, genrose.g, gs=10)
genrosenn<-Rcgmin(xx,genrose.f, gs=10) # use local numerical gradient
cat("genrosea uses analytic gradient\n")
print(genrosea)
cat("genrosenn uses default gradient approximation\n")
print(genrosenn)



cat("timings B vs U\n")
lo<-rep(-100,10)
up<-rep(100,10)
bdmsk<-rep(1,10)
tb<-system.time(ab<-Rcgminb(xx,genrose.f, genrose.g, lower=lo, upper=up, bdmsk=bdmsk))[1]
tu<-system.time(au<-Rcgminu(xx,genrose.f, genrose.g))[1]
cat("times U=",tu,"   B=",tb,"\n")
cat("solution Rcgminu\n")
print(au)
cat("solution Rcgminb\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)
}

gmaxfn<-function(x) {
     gg<-grad(maxfn, x)
}


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



cat("test that maximize=TRUE works correctly\n")

n<-6
xx<-rep(1,n)
ansmax<-Rcgmin(xx,maxfn, control=list(maximize=TRUE,trace=1))
print(ansmax)

cat("using the negmax function should give same parameters\n")
ansnegmax<-Rcgmin(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<-Rcgmin(startx,genrose.f, gr=genrose.g,lower=lo,upper=up)
print(grbds1)

cat("test lower bound only\n")
nn<-4
startx<-rep(pi,nn)
lo<-rep(2,nn)
grbds2<-Rcgmin(startx,genrose.f, gr=genrose.g,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<-Rcgmin(startx,genrose.f, gr=genrose.g,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<-Rcgmin(startx,genrose.f, gr=genrose.g,upper=up)
print(grbds4)

cat("test upper bound single value only\n")
nn<-4
startx<-rep(pi,nn)
grbds5<-Rcgmin(startx,genrose.f, gr=genrose.g,upper=10)
print(grbds5)



cat("test masks only\n")
nn<-6
bd<-c(1,1,0,0,1,1)
startx<-rep(pi,nn)
grbds6<-Rcgmin(startx,genrose.f, gr=genrose.g,bdmsk=bd)
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<-Rcgmin(startx,genrose.f, gr=genrose.g,upper=upper)
print(grbds7)


cat("test lower bound on first two elements only\n")
nn<-4
startx<-rep(0,nn)
lower<-c(0,1.1, -Inf, -Inf)
grbds8<-Rcgmin(startx,genrose.f,genrose.g,lower=lower, control=list(maxit=2000))
print(grbds8)

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

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

gsqtst<-function(xx) {
    gg<-2*(xx-2)
}

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

```
print(onepar)

cat("Suppress warnings\n")
oneparnw<-Rcgmin(startx,sqtst,  gr=gsqtst,control=list(dowarn=FALSE,trace=1))
print(oneparnw)
```

`Rcgminb` *An R implementation of a bounded nonlinear conjugate gradient algorithm with the Dai / Yuan update and restart. Based on Nash (1979) Algorithm 22 for its main structure. CALL THIS VIA* `Rcgmin` *AND DO NOT USE DIRECTLY.*

Description

The purpose of `Rcgminb` is to minimize a bounds (box) and mask constrained function of many parameters by a nonlinear conjugate gradients method. This code is entirely in R to allow users to explore and understand the method. It allows bounds (or box) constraints and masks (equality constraints) to be imposed on parameters.

This code should be called through `Rcgmin` which selects `Rcgminb` or `Rcgminu` according to the presence of bounds and masks.

Usage

```
Rcgminb(par, fn, gr, lower, upper, bdmsk, control = list(), ...)
```

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`.
`gr`	A function that returns the gradient of the objective at the supplied set of parameters `par` using auxiliary data in .... The first argument of `fn` must be `par`. This function returns the gradient as a numeric vector. The use of numerical gradients for Rcgminb is STRONGLY discouraged.
`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.

maxit A limit on the number of iterations (default 500). Note that this is used to compute a quantity `maxfeval`<-round(sqrt(n+1)*maxit) where n is the number of parameters to be minimized.

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 `Rcgminb` for details of application.

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

The source code `Rcgminb` for R is likely to remain a work in progress for some time, so users should watch the console output.

As of 2011-11-21 the following controls have been REMOVED

usenumDeriv There is now a choice of numerical gradient routines. See argument `gr`.

maximize To maximize user_function, supply a function that computes (-1)*user_function. An alternative is to call Rcgmin via the package optimx.

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. '2' indicates initial point is infeasible.
`message`	A character string giving any additional information returned by the optimizer, or 'NULL'.
`bdmsk`	Returned index describing the status of bounds and masks at the proposed solution. Parameters for which bdmsk are 1 are unconstrained or "free", those with bdmsk 0 are masked i.e., fixed. For historical reasons, we indicate a parameter is at a lower bound using -3 or upper bound using -1.

References

See `Rcgmin` documentation. Note that bounds and masks were adapted from the work by Nash and Walker-Smith(1987).

See Also

`optim`

Rcgminu — *An R implementation of an unconstrained nonlinear conjugate gradient algorithm with the Dai / Yuan update and restart. Based on Nash (1979) Algorithm 22 for its main structure. CALL THIS VIA* Rcgmin *AND DO NOT USE DIRECTLY.*

Description

The purpose of Rcgminu is to minimize an unconstrained function of many parameters by a nonlinear conjugate gradients method. This code is entirely in R to allow users to explore and understand the method.

This code should be called through Rcgmin which selects Rcgminb or Rcgminu according to the presence of bounds and masks.

Usage

```
Rcgminu(par, fn, gr, control = list(), ...)
```

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.
gr	A function that returns the gradient of the objective at the supplied set of parameters par using auxiliary data in .... The first argument of fn must be par. This function returns the gradient as a numeric vector. The use of numerical gradients for Rcgminu is STRONGLY discouraged.
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.

maxit A limit on the number of iterations (default 500). Note that this is used to compute a quantity maxfeval<-round(sqrt(n+1)*maxit) where n is the number of parameters to be minimized.

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 Rcgminu for details of application.

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

The source code Rcgminu for R is likely to remain a work in progress for some time, so users should watch the console output.

As of 2011-11-21 the following controls have been REMOVED

usenumDeriv There is now a choice of numerical gradient routines. See argument gr.

maximize To maximize user_function, supply a function that computes (-1)*user_function. An alternative is to call Rcgmin via the package optimx.

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. '2' indicates initial point is infeasible.
message	A character string giving any additional information returned by the optimizer, or 'NULL'.
bdmsk	Returned index describing the status of bounds and masks at the proposed solution. Parameters for which bdmsk are 1 are unconstrained or "free", those with bdmsk 0 are masked i.e., fixed. For historical reasons, we indicate a parameter is at a lower bound using -3 or upper bound using -1.

References

See Rcgmin documentation.

See Also

optim

Rvmmin *Variable metric nonlinear function minimization, driver.*

Description

A driver to call the unconstrained and bounds constrained versions of an R implementation of a variable metric method for minimization of nonlinear functions, possibly subject to bounds (box) constraints and masks (fixed parameters). The algorithm is based on Nash (1979) Algorithm 21 for main structure, which is itself drawn from Fletcher's (1970) variable metric code. This is also the basis of optim() method 'BFGS' which, however, does not deal with bounds or masks. In the present method, an approximation to the inverse Hessian (B) is used to generate a search direction t = - B %*% g, a simple backtracking line search is used until an acceptable point is found, and the matrix B is updated using a BFGS formula. If no acceptable point can be found, we reset B to the identity i.e., the search direction becomes the negative gradient. If the search along the negative gradient is unsuccessful, the method terminates.

This set of codes 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

```
Rvmmin(par, fn, gr, lower, upper, bdmsk, control = list(), ...)
```

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.
gr	A function that returns the gradient of the objective at the supplied set of parameters par using auxiliary data in .... The first argument of fn must be par. This function returns the gradient as a numeric vector. Note that a gradient function must generally be provided. However, to ensure compatibility with other optimizers, if gr is NULL, the forward gradient approximation from routine grfwd will be used. The use of numerical gradients for Rvmmin is discouraged. First, the termination test uses a size measure on the gradient, and numerical gradient approximations can sometimes give results that are too large. Second, if there are bounds constraints, the step(s) taken to calculate the approximation to the derivative are NOT checked to see if they are out of bounds, and the function may be undefined at the evaluation point. There is also the option of using the routines grfwd, grback, grcentral or grnd. The last of these calls the grad() function from package numDeriv. These are called by putting the name of the (numerical) gradient function in quotation marks, e.g., gr="grfwd" to use the standard forward difference numerical approximation. Note that all but the grnd routine use a stepsize parameter that can be redefined in a special scratchpad storage variable deps. The default is deps = 1e-07. However, redefining this is discouraged unless you understand what you are doing.
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. Successful completion. The source code Rvmmin for R is still a work in progress, so users should watch the console output.

The control argument is a list.

maxit A limit on the number of iterations (default 500 + 2*n where n is the number of parameters). This is the maximum number of gradient evaluations allowed.

maxfevals A limit on the number of function evaluations allowed (default 3000 + 10*n).

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

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

checkgrad = TRUE if we wish analytic gradient code checked against the approximations computed by `numDeriv`. Default is TRUE.

checkbounds = TRUE if we wish parameters and bounds to be checked for an admissible and feasible start. Default is TRUE.

keepinputpar = TRUE if we want bounds check to stop program when parameters are out of bounds. Else when FALSE, moves parameter values to nearest bound. Default is FALSE.

maximize To maximize user_function, supply a function that computes (-1)*user_function. An alternative is to call Rvmmin via the package optimx.

eps a tolerance used for judging small gradient norm (default = 1e-07). a gradient norm smaller than (1 + abs(fmin))*eps*eps is considered small enough that a local optimum has been found, where fmin is the current estimate of the minimal function value.

acctol To adjust the acceptable point tolerance (default 0.0001) in the test (f <= fmin + gradproj * steplength * acctol). This test is used to ensure progress is made at each iteration.

stepredn Step reduction factor for backtrack line search (default 0.2)

reltest Additive shift for equality test (default 100.0)

stopbadupdate A logical flag that if set TRUE will halt the optimization if the Hessian inverse cannot be updated after a steepest descent search. This indicates an ill-conditioned Hessian. A settign of FALSE causes Rvmmin methods to be aggressive in trying to optimize the function, but may waste effort. Default TRUE.

As of 2011-11-21 the following controls have been REMOVED

usenumDeriv There is now a choice of numerical gradient routines. See argument `gr`.

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 vector of two integers giving the number of function and gradient evaluations.
`convergence`	An integer indicating the situation on termination of the function. `0` indicates that the method believes it has succeeded. Other values: `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.
`message`	A description of the situation on termination of the function.
`bdmsk`	Returned index describing the status of bounds and masks at the proposed solution. Parameters for which bdmsk are 1 are unconstrained or "free", those with bdmsk 0 are masked i.e., fixed. For historical reasons, we indicate a parameter is at a lower bound using -3 or upper bound using -1.

References

Fletcher, R (1970) A New Approach to Variable Metric Algorithms, Computer Journal, 13(3), pp. 317-322.

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

See Also

optim

Examples

```
####################
## All examples for the Rvmmin package are in this .Rd file
##

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

ansrosenbrock <- Rvmmin(fn=fr,gr="grfwd", par=c(1,2))
print(ansrosenbrock)
cat("\n")
cat("No gr specified as a test\n")
ansrosenbrock0 <- Rvmmin(fn=fr, par=c(1,2))
print(ansrosenbrock0)
# use print to allow copy to separate file that can be called using source()

####################
# Simple bounds and masks test
#
# The function is a sum of squares, but we impose the
# constraints so that there are lower and upper bounds
# away from zero, and parameter 6 is fixed at the initial
# value

bt.f<-function(x){
  sum(x*x)
}

bt.g<-function(x){
  gg<-2.0*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)
cat("Initial parameters:")
print(xx)
cat("Lower bounds:")
print(lower)
cat("upper bounds:")
print(upper)
cat("Masked (fixed) parameters:")
print(which(bdmsk == 0))

ansbt<-Rvmmin(xx, bt.f, bt.g, lower, upper, bdmsk, control=list(trace=1))

print(ansbt)

######################
# A version of a generalized Rosenbrock problem
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
  gg
}

# analytic gradient test
xx<-rep(pi,10)
lower<-NULL
upper<-NULL
bdmsk<-NULL
genrosea<-Rvmmin(xx,genrose.f, genrose.g, gs=10)
genrosenf<-Rvmmin(xx,genrose.f, gr="grfwd", gs=10) # use local numerical gradient
genrosenullgr<-Rvmmin(xx,genrose.f, gs=10) # no gradient specified
cat("genrosea uses analytic gradient\n")
```

```
print(genrosea)
cat("genrosenf uses grfwd standard numerical gradient\n")
print(genrosenf)
cat("genrosenullgr has no gradient specified\n")
print(genrosenullgr)
cat("Other numerical gradients can be used.\n")

cat("timings B vs U\n")
lo<-rep(-100,10)
up<-rep(100,10)
bdmsk<-rep(1,10)
tb<-system.time(ab<-Rvmminb(xx,genrose.f, genrose.g, lower=lo, upper=up, bdmsk=bdmsk))[1]
tu<-system.time(au<-Rvmminu(xx,genrose.f, genrose.g))[1]
cat("times U=",tu,"   B=",tb,"\n")
cat("solution Rvmminu\n")
print(au)
cat("solution Rvmminb\n")
print(ab)
cat("diff fu-fb=",au$value-ab$value,"\n")
cat("max abs parameter diff = ", max(abs(au$par-ab$par)),"\n")

# Test that Rvmmin will maximize as well as minimize

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("test that maximize=TRUE works correctly\n")

n<-6
xx<-rep(1,n)
ansmax<-Rvmmin(xx,maxfn, gr="grfwd", control=list(maximize=TRUE,trace=1))
print(ansmax)

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


####################
cat("test bounds and masks\n")
nn<-4
startx<-rep(pi,nn)
```

```
lo<-rep(2,nn)
up<-rep(10,nn)
grbds1<-Rvmmin(startx,genrose.f, genrose.g, lower=lo,upper=up)
print(grbds1)

cat("test lower bound only\n")
nn<-4
startx<-rep(pi,nn)
lo<-rep(2,nn)
grbds2<-Rvmmin(startx,genrose.f, genrose.g, 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<-Rvmmin(startx,genrose.f, genrose.g, 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<-Rvmmin(startx,genrose.f, genrose.g, upper=up)
print(grbds4)

cat("test upper bound single value only\n")
nn<-4
startx<-rep(pi,nn)
grbds5<-Rvmmin(startx,genrose.f, genrose.g, upper=10)
print(grbds5)



cat("test masks only\n")
nn<-6
bd<-c(1,1,0,0,1,1)
startx<-rep(pi,nn)
grbds6<-Rvmmin(startx,genrose.f, genrose.g, bdmsk=bd)
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<-Rvmmin(startx,genrose.f, genrose.g, upper=upper)
print(grbds7)


cat("test lower bound on first two elements only\n")
nn<-4
```

```
startx<-rep(0,nn)
lower<-c(0,1.1, -Inf, -Inf)
grbds8<-Rvmmin(startx,genrose.f,genrose.g,lower=lower, control=list(maxit=2000))
print(grbds8)

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

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

nn<-1
startx<-rep(0,nn)
onepar<-Rvmmin(startx,sqtst, gr="grfwd", control=list(trace=1))
print(onepar)

cat("Suppress warnings\n")
oneparnw<-Rvmmin(startx,sqtst, gr="grfwd", control=list(dowarn=FALSE,trace=1))
print(oneparnw)
```

`Rvmminb` *Variable metric nonlinear function minimization with bounds constraints*

Description

A bounds-constarined R implementation of a variable metric method for minimization of nonlinear functions subject to bounds (box) constraints and masks (fixed parameters).

See manual Rvmmin.Rd for more details and examples.

Usage

```
Rvmminb(par, fn, gr, lower, upper, bdmsk, control = list(), ...)
```

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`.
`gr`	A function that returns the gradient of the objective at the supplied set of parameters `par` using auxiliary data in .... The first argument of `fn` must be `par`. This function returns the gradient as a numeric vector. Note that a gradient function MUST be provided. See the manual for `Rvmmin`, which is the usual way `Rvmminb` is called. The user must take responsibility for errors if `Rvmminb` is called directly.
`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. See the manual Rvmmin.Rd for details.
...	Further arguments to be passed to fn.

Details

This routine is intended to be called from Rvmmin, which will, if necessary, supply a gradient approximation. However, some users will want to avoid the extra overhead, in which case it is important to provide an appropriate and high-accuracy gradient routine.

Note that bounds checking, if it is carried out, is done by Rvmmin.

Functions fn must return a numeric value.

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 vector of two integers giving the number of function and gradient evaluations.
convergence	An integer indicating the situation on termination of the function. 0 indicates that the method believes it has succeeded. Other values: 1 indicates that the iteration limit maxit had been reached. 2 indicates that a point has been found with small gradient norm (< (1 + abs(fmin))*eps*eps) 3 indicates approx. inverse Hessian cannot be updated at steepest descent iteration (i.e., something very wrong) 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.
message	A description of the situation on termination of the function.
bdmsk	Returned index describing the status of bounds and masks at the proposed solution. Parameters for which bdmsk are 1 are unconstrained or "free", those with bdmsk 0 are masked i.e., fixed. For historical reasons, we indicate a parameter is at a lower bound using -3 or upper bound using -1.

See Also

optim

Examples

```
## See Rvmmin.Rd
```

`Rvmminu` *Variable metric nonlinear function minimization, unconstrained*

Description

An R implementation of a variable metric method for minimization of unconstrained nonlinear functions.

See the manual Rvmmin.Rd for details.

Usage

```
Rvmminu(par, fn, gr, control = list(), ...)
```

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`.
`gr`	A function that returns the gradient of the objective at the supplied set of parameters `par` using auxiliary data in .... The first argument of `fn` must be `par`. This function returns the gradient as a numeric vector. Note that a gradient function MUST be provided. See the manual for `Rvmmin`, which is the usual way `Rvmminu` is called. The user must take responsibility for errors if `Rvmminu` is called directly.
`control`	An optional list of control settings. See the manual Rvmmin.Rd for details. Some control elements apply only when parameters are bounds constrained and are not used in this function.
`...`	Further arguments to be passed to `fn`.

Details

This routine is intended to be called from `Rvmmin`, which will, if necessary, supply a gradient approximation. However, some users will want to avoid the extra overhead, in which case it is important to provide an appropriate and high-accuracy gradient routine.

Functions `fn` must return a numeric value.

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 vector of two integers giving the number of function and gradient evaluations.
`convergence`	An integer indicating the situation on termination of the function. `0` indicates that the method believes it has succeeded. Other values:

`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.

`message` A description of the situation on termination of the function.

See Also

`optim`

Examples

```
####in Rvmmin.Rd ####
```

`scalechk` *Check the scale of the initial parameters and bounds input to an optimization code used in nonlinear optimization*

Description

Nonlinear optimization problems often have different scale for different parameters. This function is intended to explore the differences in scale. It is, however, an imperfect and heuristic tool, and could be improved.

At this time scalechk does NOT take account of masks. (?? should 110702)

Usage

```
scalechk(par, lower = lower, upper = upper, bdmsk=NULL, dowarn = TRUE)
```

Arguments

`par`	A numeric vector of starting values of the optimization function parameters.
`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.
`dowarn`	Set TRUE to issue warnings. Othwerwise this is a silent routine. Default TRUE.

Details

The scalechk function will check that the bounds exist and are admissible, that is, that there are no lower bounds that exceed upper bounds.

NOTE: Free paramters outside bounds are adjusted to the nearest bound. We then set parchanged = TRUE which implies the original parameters were infeasible.

There is a check if lower and upper bounds are very close together, in which case a mask is imposed and maskadded is set TRUE. NOTE: it is generally a VERY BAD IDEA to have bounds close together in optimization, but here we use a tolerance based on the double precision machine epsilon. Thus it is not a good idea to rely on scalechk() to test if bounds constraints are well-posed.

Value

A list with components:

Returns: # list(lpratio, lbratio) – the log of the ratio of largest to smallest parameters # and bounds intervals (upper-lower) in absolute value (ignoring Inf, NULL, NA)

`lpratio`	The log of the ratio of largest to smallest parameters in absolute value (ignoring Inf, NULL, NA)
`lbration`	The log of the ratio of largest to smallest bounds intervals (upper-lower) in absolute value (ignoring Inf, NULL, NA)

Examples

```
#####################
  par <- c(-1.2, 1)
  lower <- c(-2, 0)
  upper <- c(100000, 10)
  srat<-scalechk(par, lower, upper,dowarn=TRUE)
  print(srat)
  sratv<-c(srat$lpratio, srat$lbratio)
  if (max(sratv,na.rm=TRUE) > 3) { # scaletol from ctrldefault in optimx
     warnstr<-"Parameters or bounds appear to have different scalings.\n
     This can cause poor performance in optimization. \n
     It is important for derivative free methods like BOBYQA, UOBYQA, NEWUOA."
     cat(warnstr,"\n")
  }
```

`snewton` *Safeguarded Newton methods for function minimization using R functions.*

Description

These version of the safeguarded Newton solves the equations with the R function solve(). In `snewton` a backtracking line search is used, while in `snewtonm` we rely on a Marquardt stabilization.

Usage

```
snewton(par, fn, gr, hess, control = list(trace=0, maxit=500), ...)

snewtonm(par, fn, gr, hess, control = list(trace=0, maxit=500), ...)
```

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.
gr	A function that returns the gradient of the objective at the supplied set of parameters par using auxiliary data in .... The first argument of fn must be par. This function returns the gradient as a numeric vector.
hess	A function to compute the Hessian matrix. This should be provided as a square, symmetric matrix.
control	An optional list of control settings.
...	Further arguments to be passed to fn.

Details

Functions fn must return a numeric value. gr must return a vector. hess must return a matrix. The control argument is a list. See the code for snewton.R for completeness. Some of the values that may be important for users are:

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

watch Set TRUE if the routine is to stop for user input (e.g., Enter) after each iteration. Default is FALSE.

maxit A limit on the number of iterations (default 500 + 2*n where n is the number of parameters). This is the maximum number of gradient evaluations allowed.

maxfeval A limit on the number of function evaluations allowed (default 3000 + 10*n).

eps a tolerance used for judging small gradient norm (default = 1e-07). a gradient norm smaller than (1 + abs(fmin))*eps*eps is considered small enough that a local optimum has been found, where fmin is the current estimate of the minimal function value.

acctol To adjust the acceptable point tolerance (default 0.0001) in the test (f <= fmin + gradproj * steplength * acctol). This test is used to ensure progress is made at each iteration.

stepdec Step reduction factor for backtrack line search (default 0.2)

defstep Initial stepsize default (default 1)

reltest Additive shift for equality test (default 100.0)

Value

A list with components:

xs The best set of parameters found.

fv The value of the objective at the best set of parameters found.

grd The value of the gradient at the best set of parameters found. A vector.

H The value of the Hessian at the best set of parameters found. A matrix.

niter The number of Newton iterations used in finding the solution.

message A message giving some information on the status of the solution.

References

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

See Also

optim

Examples

```
#Rosenbrock banana valley function
f <- function(x){
return(100*(x[2] - x[1]*x[1])^2 + (1-x[1])^2)
}
#gradient
gr <- function(x){
return(c(-400*x[1]*(x[2] - x[1]*x[1]) - 2*(1-x[1]), 200*(x[2] - x[1]*x[1])))
}
#Hessian
h <- function(x) {
a11 <- 2 - 400*x[2] + 1200*x[1]*x[1]; a21 <- -400*x[1]
return(matrix(c(a11, a21, a21, 200), 2, 2))
}

fg <- function(x){ #function and gradient
  val <- f(x)
  attr(val,"gradient") <- gr(x)
  val
}
fgh <- function(x){ #function and gradient
  val <- f(x)
  attr(val,"gradient") <- gr(x)
  attr(val,"hessian") <- h(x)
  val
}

x0 <- c(-1.2, 1)

sr <- snewton(x0, fn=f, gr=gr, hess=h, control=list(trace=1))
print(sr)

srm <- snewtonm(x0, fn=f, gr=gr, hess=h, control=list(trace=1))
print(srm)
```

```
#Example 2: Wood function
#
wood.f <- function(x){
  res <- 100*(x[1]^2-x[2])^2+(1-x[1])^2+90*(x[3]^2-x[4])^2+(1-x[3])^2+
    10.1*((1-x[2])^2+(1-x[4])^2)+19.8*(1-x[2])*(1-x[4])
  return(res)
}
#gradient:
wood.g <- function(x){
  g1 <- 400*x[1]^3-400*x[1]*x[2]+2*x[1]-2
  g2 <- -200*x[1]^2+220.2*x[2]+19.8*x[4]-40
  g3 <- 360*x[3]^3-360*x[3]*x[4]+2*x[3]-2
  g4 <- -180*x[3]^2+200.2*x[4]+19.8*x[2]-40
  return(c(g1,g2,g3,g4))
}
#hessian:
wood.h <- function(x){
  h11 <- 1200*x[1]^2-400*x[2]+2;    h12 <- -400*x[1]; h13 <- h14 <- 0
  h22 <- 220.2; h23 <- 0;    h24 <- 19.8
  h33 <- 1080*x[3]^2-360*x[4]+2;    h34 <- -360*x[3]
  h44 <- 200.2
  H <- matrix(c(h11,h12,h13,h14,h12,h22,h23,h24,
                h13,h23,h33,h34,h14,h24,h34,h44),ncol=4)
  return(H)
}
################################################
w0 <- c(-3, -1, -3, -1)

wd <- snewton(w0, fn=wood.f, gr=wood.g, hess=wood.h, control=list(trace=1))
print(wd)

wdm <- snewtonm(w0, fn=wood.f, gr=wood.g, hess=wood.h, control=list(trace=1))
print(wdm)
```

---

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)
```

---

tn *Truncated Newton minimization of an unconstrained function.*

Description

An R implementation of the Truncated Newton method of Stephen Nash for driver to call the unconstrained function minimization. The algorithm is based on Nash (1979)

This set of codes is entirely in R to allow users to explore and understand the method.

Usage

```
tn(x, fgfun, trace, ...)
```

Arguments

x	A numeric vector of starting estimates.
fgfun	A function that returns the value of the objective at the supplied set of parameters par using auxiliary data in .... The gradient is returned as attribute "gradient". The first argument of fgfun must be par.
trace	> 0 if progress output is to be presented.
...	Further arguments to be passed to fn.

Details

Function fgfun must return a numeric value in list item f and a numeric vector in list item g.

Value

A list with components:

xstar	The best set of parameters found.
f	The value of the objective at the best set of parameters found.
g	The gradient of the objective at the best set of parameters found.
ierror	An integer indicating the situation on termination. 0 indicates that the method believes it has succeeded; 2 that more than maxfun (default 150*n, where there are n parameters); 3 if the line search appears to have failed (which may not be serious); and -1 if there appears to be an error in the input parameters.
nfngr	A number giving a measure of how many conjugate gradient solutions were used during the minimization process.

References

Stephen G. Nash (1984) "Newton-type minimization via the Lanczos method", SIAM J Numerical Analysis, vol. 21, no. 4, pages 770-788.

For Matlab code, see https://www.netlib.org/opt/tn

See Also

optim

Examples

```
#####################
## All examples are in this .Rd file
##
## Rosenbrock Banana function
fr <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
gr <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    g1 <- -400 * (x2 - x1*x1) * x1 - 2*(1-x1)
    g2 <- 200*(x2 - x1*x1)
    gg<-c(g1, g2)
}

rosefg<-function(x){
   f<-fr(x)
   g<-gr(x)
   attr(f, "gradient") <- g
   f
}

x<-c(-1.2, 1)

ansrosenbrock <- tn(x, rosefg)
print(ansrosenbrock) # use print to allow copy to separate file that
cat("Compare to optim\n")
ansoptrose <- optim(x, fr, gr)
print(ansoptrose)


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
gg
}

grosefg<-function(x, gs=100.0) {
    f<-genrose.f(x, gs)
    g<-genrose.g(x, gs)
    attr(f, "gradient") <- g
    f
}

n <- 100
x <- (1:100)/20
groseu<-tn(x, grosefg, gs=10)
print(groseu)

groseuo <- optim(x, fn=genrose.f, gr=genrose.g, method="BFGS",
      control=list(maxit=1000), gs=10)
cat("compare optim BFGS\n")
print(groseuo)


lower<-1+(1:n)/100
upper<-5-(1:n)/100
xmid<-0.5*(lower+upper)

grosec<-tnbc(xmid, grosefg, lower, upper)
print(grosec)

cat("compare L-BFGS-B\n")
grosecl <- optim(par=xmid, fn=genrose.f, gr=genrose.g,
     lower=lower, upper=upper, method="L-BFGS-B")
print(grosecl)
```

---

tnbc *Truncated Newton function minimization with bounds constraints*

Description

A bounds-constarined R implementation of a truncated Newton method for minimization of non-linear functions subject to bounds (box) constraints.

Usage

```
tnbc(x, fgfun, lower, upper, trace=0, ...)
```

Arguments

x	A numeric vector of starting estimates.
fgfun	A function that returns the value of the objective at the supplied set of parameters par using auxiliary data in .... The gradient is returned as attribute "gradient". The first argument of fgfun must be par.
lower	A vector of lower bounds on the parameters.
upper	A vector of upper bounds on the parameters.
trace	Set >0 to cause intermediate output to allow progress to be followed.
...	Further arguments to be passed to fn.

Details

Function fgfun must return a numeric value in list item f and a numeric vector in list item g.

Value

A list with components:

xstar	The best set of parameters found.
f	The value of the objective at the best set of parameters found.
g	The gradient of the objective at the best set of parameters found.
ierror	An integer indicating the situation on termination. 0 indicates that the method believes it has succeeded; 2 that more than maxfun (default 150*n, where there are n parameters); 3 if the line search appears to have failed (which may not be serious); and -1 if there appears to be an error in the input parameters.
nfngr	A number giving a measure of how many conjugate gradient solutions were used during the minimization process.

References

Stephen G. Nash (1984) "Newton-type minimization via the Lanczos method", SIAM J Numerical Analysis, vol. 21, no. 4, pages 770-788.

For Matlab code, see https://www.netlib.org/opt/tn

See Also

optim

Examples

```
## See tn.Rd
```

Index