Package 'maxLik'

July 2, 2014

Version 1.2-0

Date 2013/10/22

Title Maximum Likelihood Estimation

Author Ott Toomet <otoomet@gmail.com>,Arne Henningsen <arne.henningsen@gmail.com>,with contributions from Spencer Graves and Yves Croissant

Maintainer Arne Henningsen <arne.henningsen@gmail.com>

Depends R (>= 2.4.0), miscTools (>= 0.6-8)

Imports sandwich

Description Tools for Maximum Likelihood Estimation

License GPL (>= 2)

ByteCompile yes

URL http://www.maxLik.org

NeedsCompilation no

Repository CRAN

Date/Publication 2013-10-22 08:19:17

R topics documented:

`activePar` *free parameters under maximisation*

Description

Return a logical vector, indicating which parameters were free under maximisation, as opposed to the fixed parameters, treated as constants.

Usage

```
activePar(x, ...)
## Default S3 method:
activePar(x, ...)
```

Arguments

`x`	object, created by a maximisation routine, or derived from a maximisation object. Currently only `maxNR` and it's derivations support `activePar`
`...`	further arguments for methods

Details

Several optimisation routines allow the user to fix some parameter values (or do it automatically in some cases). For gradient or Hessian based inference one has to know which parameters carry optimisation-related information.

Value

A logical vector, indicating whether the parameters were free to change during optimisation algorithm.

Author(s)

Ott Toomet, <otoomet@econ.au.dk>

See Also

maxNR, nObs

Examples

```
# a simple two-dimensional exponential hat
f <- function(a) exp(-a[1]^2 - a[2]^2)
#
# maximize wrt. both parameters
free <- maxNR(f, start=1:2)
summary(free)  # results should be close to (0,0)
activePar(free)
# allow only the second parameter to vary
cons <- maxNR(f, start=1:2, activePar=c(FALSE,TRUE))
summary(cons) # result should be around (1,0)
activePar(cons)
```

AIC.maxLik *Methods for the various standard functions*

Description

These are methods for the maxLik related objects. See the documentation for the corresponding generic functions

---

bread.maxLik *Bread for Sandwich Estimator*

Description

Extracting an estimator for the 'bread' of the sandwich estimator, see bread.

Usage

```
## S3 method for class 'maxLik'
bread( x, ... )
```

Arguments

x	an object of class maxLik.
...	further arguments (currently ignored).

Value

Matrix, the inverse of the expectation of the second derivative (Hessian matrix) of the log-likelihood function with respect to the parameters, usually equal to the variance covariance matrix of the parameters times the number of observations.

Warnings

The **sandwich** package must be loaded before this method can be used.

This method works only if `maxLik` was called with argument `grad` equal to a gradient function or (if no gradient function is specified) argument `logLik` equal to a log-likelihood function that return the gradients or log-likelihood values, respectively, for each observation.

Author(s)

Arne Henningsen

See Also

`bread`, `maxLik`.

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t

## Estimate with numeric gradient and hessian
a <- maxLik(loglik, start=1 )

# Extract the "bread"
library( sandwich )
bread( a )

all.equal( bread( a ), vcov( a ) * nObs( a ) )
```

---

compareDerivatives *function to compare analytic and numeric derivatives*

Description

This function compares analytic and numerical derivative and prints a few diagnostics. It is intended for testing pre-programmed derivative routines for maximisation algorithms.

Usage

```
compareDerivatives(f, grad, hess=NULL, t0, eps=1e-6, print=TRUE, ...)
```

Arguments

f	function to be differentiated. The parameter (vector) of interest must be the first argument. The function may return a vector.
grad	analytic gradient. This may be either a function, returning the analytic gradient, or a numeric vector, the pre-computed gradient. The function must use the same set of parameters as f. If f is a vector-valued function, grad must return/be a matrix where the number of rows equals the number of components of f, and the number of columns must equal to the number of components in t0.
hess	function returning the analytic hessian. If present, hessian matrices are compared too. Only appropriate for scalar-valued functions.
t0	parameter vector indicating the point at which the derivatives are compared. The derivative is taken with respect to this vector.
eps	numeric. Step size for numeric differentiation. Central derivative is used.
print	logical: TRUE to print a summary, FALSE to return the comparison only (invisibly).
...	further arguments to f, grad and hess.

Details

For every component of f, the parameter value, analytic and numeric derivative and their relative difference

rel.diff = (analytic - numeric)/(0.5*(analytic+numeric))

are printed; if analytic = 0 = numeric, we define rel.diff = 0. If analytic derivatives are correct and the function is sufficiently smooth, expect the relative differences to be less than 1e-7.

Value

A list with the following components:

t0	the input argument t0
f.t0	f(t0)
compareGrad	a list with components analytic = grad(t0), nmeric = numericGradient(f, t0), and their rel.diff.
maxRelDiffGrad	max(abs(rel.diff))

If hess is also provided, the following optional components are also present:

compareHessian	a list with components analytic = hess(t0), numeric = numericGradient(grad, t0), and their rel.diff.
maxRelDiffHess	max(abs(rel.diff)) for the Hessian

Author(s)

Ott Toomet <otoomet@ut.ee> and Spencer Graves

See Also

numericGradient deriv

Examples

```
## A simple example with sin(x)' = cos(x)
f <- function(x)c(sin=sin(x))
Dsin <- compareDerivatives(f, cos, t0=c(angle=1))
D2sin <- compareDerivatives(f, cos, function(x)-sin(x), t0=1)

##
## Example of log-likelihood of normal density.  Two-parameter
## function.
##
x <- rnorm(100, 1, 2) # generate rnorm x
l <- function(b) sum(log(dnorm((x-b[1])/b[2])/b[2]))
              # b[1] = mu, b[2] = sigma
gradl <- function(b) {
    c(mu=sum(x - b[1])/b[2]^2,
    sigma=sum((x - b[1])^2/b[2]^3 - 1/b[2]))
}
gradl. <- compareDerivatives(l, gradl, t0=c(mu=1,sigma=2))

##
## An example with f returning a vector, t0 = a scalar
##
trig <- function(x)c(sin=sin(x), cos=cos(x))
Dtrig <- function(x)c(sin=cos(x), cos=-sin(x))
Dtrig. <- compareDerivatives(trig, Dtrig, t0=1)

D2trig <- function(x)-trig(x)
D2trig. <- compareDerivatives(trig, Dtrig, D2trig, t0=1)
```

---

condiNumber *Print matrix condition numbers column-by-column*

Description

This function prints the condition number of a matrix while adding columns one-by-one. This is useful for testing multicollinearity and other numerical problems. This is a generic function with default method and method for maxLik objects.

Usage

```
condiNumber(x, ...)
## Default S3 method:
condiNumber(x, exact = FALSE, norm = FALSE,
```

```
print.level=1, ...)
## S3 method for class 'maxLik'
condiNumber(x, ...)
```

Arguments

x	numeric matrix, condition numbers of which are to be printed
exact	logical, should condition numbers be exact or approximations (see link{kappa})
norm	logical, whether the columns should be normalised to have unit norm
print.level	numeric, positive value will output the numbers during the calculations. Useful for interactive work.
...	other arguments to different methods

Details

Statistical model often fail because of strong correlation between explanatory variables in linear index (multicollinearity) or because the evaluated maximum of a non-linear model is virtually flat. In both cases, the (near) singularity of the related matrices may give us a hint, how to improve the results.

condiNumber allows to inspect the matrices column-by-column and unerstand which variable leads to a huge jump in the condition number. If the single column does not immediately tell what is the problem, one may try to estimate this column by OLS using the previous columns as explanatory variables. The columns, which explain virtually all the variation, should have extremely high t-values.

Value

Invisible vector of condition numbers by column.

Author(s)

Ott Toomet <otoomet@ut.ee>

References

W. Greene, Advanced Econometrics, p ...

See Also

kappa

Examples

```
set.seed(0)
## generate a simple multicollinear dataset
x1 <- runif(100)
x2 <- runif(100)
x3 <- x1 + x2 + 0.000001*runif(100) # this is virtually equal to x1 + x2
x4 <- runif(100)
```

```
y <- x1 + x2 + x3 + x4 + rnorm(100)
m <- lm(y ~ -1 + x1 + x2 + x3 + x4)
print(summary(m)) # note the low t-values while R^2 is 0.88.
                  # This hints multicollinearity
condiNumber(model.matrix(m)) # this _prints_ condition numbers.
                             # note the values 'explode' with x3
## we may test the results further:
print(summary(lm(x3 ~ -1 + x1 + x2))) # Note the high t-values and R^2
```

`estfun.maxLik` *Extract Gradients Evaluated at each Observation*

Description

Extract the gradients of the log-likelihood function evaluated at each observation ('Empirical Estimating Function', see `estfun`).

Usage

```
## S3 method for class 'maxLik'
estfun( x, ... )
```

Arguments

`x`	an object of class `maxLik`.
`...`	further arguments (currently ignored).

Value

Matrix of gradients of the log-likelihood function at the estimated parameter value evaluated at each observation

Warnings

The **sandwich** package must be loaded before this method can be used.

This method works only if `maxLik` was called with argument `grad` equal to a gradient function or (if no gradient function is specified) argument `logLik` equal to a log-likelihood function that return the gradients or log-likelihood values, respectively, for each observation.

Author(s)

Arne Henningsen

See Also

`estfun`, `maxLik`.

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t

## Estimate with numeric gradient and hessian
a <- maxLik(loglik, start=1 )

# Extract the gradients evaluated at each observation
library( sandwich )
estfun( a )

## Estimate with analytic gradient
gradlik <- function(theta) 1/theta - t
b <- maxLik(loglik, gradlik, start=1)
estfun( b )
all.equal( c( estfun( b ) ), gradlik( coef( b ) ) )
```

`fnSubset` *Call fnFull with variable and fixed parameters*

Description

Combine variable parameters in `x` with `xFixed` and pass to `fnFull`. Useful for optimizing over a subset of parameters without writing a separate function. Values are combined by name if available. Otherwise, `xFull` is constructed by position (the default).

Usage

```
fnSubset(x, fnFull, xFixed, xFull=c(x, xFixed), ...)
```

Arguments

`x`	Variable parameters to be passed to `fnFull`.
`fnFull`	Function whose first argument has length = length(xFull).
`xFixed`	Parameters to be combined with `x` to construct the first argument for a call to `fnFull`.
`xFull`	Prototype initial argument for `fnFull`.
`...`	Optional arguments passed to `fnFull`.

Details

1. Confirm that length(x) + length(xFixed) = length(xFull)
2. If `xFull` has names, match at least `xFixed` by name. Else xFull = c(x, xFixes), the default.
3. fnFull(xFull, ...)

Value

value returned by `fnFull`

Author(s)

Spencer Graves

See Also

`optim` `dlmMLE` `maxLik` `maxNR`

Examples

```
##
## Test with 'optim'
##
fn <- function(x, x0)(x[2]-2*x[1]-x0)^2
fullEst <- optim(1:2, fn, x0=3)

# Fix the last component
est4 <- optim(1, fnSubset, x0=3, fnFull=fn, xFixed=4)

# Fix the first component
fnSubset(1, fn, c(a=4), c(a=1, b=2), x0=3)
# After substitution:  xFull = c(a=4, b=1),
# so fn = (1-2*4-3)^2 = (-10)^2 = 100

est4. <- optim(1, fnSubset, x0=3, fnFull=fn, xFixed=c(a=4),
               xFull=c(a=1, b=2))
# At optim: xFull=c(a=4, b=10.9),
# so fn = (10.9-2*4-3)^2 = (-0.1)^2 = 0.01

##
## Test with maxNR
##
#  fn2max = -fn
fn2max <- function(x, x0, ...)(-(x[2]-2*x[1]-x0)^2)
#  Need "..." here when called directly from maxNR,
#  because maxNR will also pass 'constantPar'
# Fix the last component
NR4 <- maxNR(fnSubset, start=1, x0=3, fnFull=fn2max, xFixed=4)
# Same thing using maxNR(..., activePar)
NR4. <- maxNR(fn2max, start=c(1, 4), x0=3, constantPar=2)

##
## Test with maxLik
##
# Same as maxNR
max4 <- maxLik(fnSubset, start=1, x0=3, fnFull=fn2max, xFixed=4)
# Same thing using constantPar in maxNR, called by maxLik
max4 <- maxLik(fn2max, start=c(1, 4), x0=3, constantPar=2)
```

hessian *Hessian matrix*

Description

This function extracts the Hessian of the M-estimator of statistical model. It should be supplied by the underlying optimisation algorithm, possibly using approximations.

Usage

```
hessian(x, ...)
## Default S3 method:
hessian(x, ...)
```

Arguments

x	a M-estimator based statistical model
...	other arguments for methods

Value

A numeric matrix, the Hessian of the model at the estimated parameter values. If the maximum is flat, the Hessian is singular. In that case you may want to invert only the non-singular part of the matrix. You may also want to fix certain parameters (see activePar).

Author(s)

Ott Toomet, <otoomet@econ.au.dk>

See Also

maxLik, activePar

Examples

```
# log-likelihood for normal density
# a[1] - mean
# a[2] - standard deviation
ll <- function(a) sum(-log(a[2]) - (x - a[1])^2/(2*a[2]^2))
x <- rnorm(1000) # sample from standard normal
ml <- maxLik(ll, start=c(1,1))
# ignore eventual warnings "NaNs produced in: log(x)"
summary(ml) # result should be close to c(0,1)
hessian(ml) # How the Hessian looks like
sqrt(-solve(hessian(ml))) # Note: standard deviations are on the diagonal
#
# Now run the same example while fixing a[2] = 1
mlf <- maxLik(ll, start=c(1,1), activePar=c(TRUE, FALSE))
summary(mlf) # first parameter close to 0, the second exactly 1.0
```

```
hessian(mlf)
# Now look at the Hessian.  Note that NA-s are in place of passive
# parameters.
# now invert only the free parameter part of the Hessian
sqrt(-solve(hessian(mlf)[activePar(mlf), activePar(mlf)]))
# gives the standard deviation for the mean
```

`logLik.maxLik` *Return the log likelihood value*

Description

Return the log likelihood value of objects of class `maxLik` and `summary.maxLik`.

Usage

```
## S3 method for class 'maxLik'
logLik( object, ... )
## S3 method for class 'summary.maxLik'
logLik( object, ... )
```

Arguments

`object`	object of class `maxLik` or `summary.maxLik`, usually a model estimated with Maximum Likelihood
`...`	additional arguments to methods

Value

A single numeric, log likelihood of the estimated model

Author(s)

Arne Henningsen, Ott Toomet <otoomet@ut.ee>

See Also

`maxLik`

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t
gradlik <- function(theta) 1/theta - t
hesslik <- function(theta) -100/theta^2
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, start=1)
## print log likelihood value
```

```
logLik( a )
## print log likelihood value of summary object
b <- summary( a )
logLik( b )
```

---

maxBFGS *BFGS, conjugate gradient, SANN and Nelder-Mead Maximization*

Description

These functions are wrappers for optim where the arguments are compatible with maxNR. Note that there is a maxNR-based BFGS implementation maxBFGSR.

Usage

```
maxBFGS(fn, grad = NULL, hess=NULL, start, fixed = NULL,
   print.level = 0, iterlim = 200, constraints = NULL,
   tol = 1e-08, reltol=tol,
   finalHessian=TRUE,
   parscale=rep(1, length=length(start)), ... )

maxCG(fn, grad = NULL, hess = NULL, start, fixed = NULL,
   print.level = 0, iterlim = 500, constraints = NULL,
   tol = 1e-08, reltol=tol,
   finalHessian=TRUE,
   parscale = rep(1, length = length(start)),
   alpha = 1, beta = 0.5, gamma = 2, ...)

maxSANN(fn, grad = NULL, hess = NULL, start, fixed = NULL,
   print.level = 0, iterlim = 10000, constraints = NULL,
   tol = 1e-08, reltol=tol,
   finalHessian=TRUE,
   cand = NULL, temp = 10, tmax = 10,
   parscale = rep(1, length = length(start)),
   random.seed = 123, ... )

maxNM(fn, grad = NULL, hess = NULL, start, fixed = NULL,
   print.level = 0, iterlim = 500, constraints = NULL,
   tol = 1e-08, reltol=tol,
   finalHessian=TRUE,
   parscale = rep(1, length = length(start)),
   alpha = 1, beta = 0.5, gamma = 2, ...)
```

Arguments

fn	function to be maximised. Must have the parameter vector as the first argument. In order to use numeric gradient and BHHH method, fn must return vector of

observation-specific likelihood values. Those are summed by maxNR if necessary. If the parameters are out of range, `fn` should return `NA`. See details for constant parameters.

`grad` gradient of the function. Must have the parameter vector as the first argument. If `NULL`, numeric gradient is used (only maxBFGS uses gradient). Gradient may return a matrix, where columns correspond to the parameters and rows to the observations (useful for maxBHHH). The columns are summed internally.

`hess` Hessian of the function. Not used by any of these methods, for compatibility with `maxNR`.

`start` initial values for the parameters.

`fixed` parameters that should be fixed at their starting values: either a logical vector of the same length as argument `start`, a numeric (index) vector indicating the positions of the fixed parameters, or a vector of character strings indicating the names of the fixed parameters (parameter names are taken from argument `start`).

`print.level` a larger number prints more working information.

`iterlim` maximum number of iterations.

`constraints` either `NULL` for unconstrained optimization or a list with two components. The components may be either `eqA` and `eqB` for equality-constrained optimization $A\theta + B = 0$; or `ineqA` and `ineqB` for inequality constraints $A\theta + B > 0$. In the inequality-constraints-case, more than one row in `ineqA` and `ineqB` corresponds to more than one linear constraint, in that case all these must be positive. The equality-constrained problem is forwarded to `sumt`, the inequality-constrained case to `constrOptim2`.

`tol, reltol` the relative convergence tolerance (see `optim`). `tol` is for compatibility with `maxNR`.

`finalHessian` how (and if) to calculate the final Hessian. Either `FALSE` (not calculate), `TRUE` (use analytic/numeric Hessian) or `"bhhh"/"BHHH"` for information equality approach. The latter approach is only suitable for maximizing log-likelihood function. It requires the gradient/log-likelihood to be supplied by individual observations, see `maxBHHH` for details.

`cand` a function used in the `"SANN"` algorithm to generate a new candidate point; if it is `NULL`, a default Gaussian Markov kernel is used (see argument `gr` of `optim`).

`temp` controls the '"SANN"' method. It is the starting temperature for the cooling schedule. Defaults to '10'.

`tmax` is the number of function evaluations at each temperature for the '"SANN"' method. Defaults to '10'. (see `optim`)

`random.seed` an integer used to seed R's random number generator. This is to ensure replicability when the 'Simulated Annealing' method is used. Defaults to 123.

`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. (see `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). (see `optim`)

...	further arguments for fn and grad.

Details

The 'state' (or 'seed') of R's random number generator is saved at the beginning of the maxSANN function and restored at the end of this function so that this function does *not* affect the generation of random numbers although the random seed is set to argument random.seed and the 'SANN' algorithm uses random numbers.

Value

Object of class "maxim":

maximum	value of fn at maximum.
estimate	best set of parameters found.
gradient	vector, gradient at parameter value estimate.
gradientObs	matrix of gradients at parameter value estimate evaluated at each observation (only if grad returns a matrix or grad is not specified and fn returns a vector).
hessian	value of Hessian at optimum.
code	integer. Success code, 0 is success (see optim).
message	character string giving any additional information returned by the optimizer, or NULL.
fixed	logical vector indicating which parameters are treated as constants.
iterations	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.
type	character string "BFGS maximisation".
constraints	A list, describing the constrained optimization (NULL if unconstrained). Includes the following components: • typetype of constrained optimization • outer.iterationsnumber of iterations in the constraints step • barrier.valuevalue of the barrier function

Author(s)

Ott Toomet <otoomet@ut.ee>, Arne Henningsen

See Also

optim, nlm, maxNR, maxBHHH, maxBFGSR.

Examples

```
# Maximum Likelihood estimation of the parameter of Poissonian distribution
n <- rpois(100, 3)
loglik <- function(l) n*log(l) - l - lfactorial(n)
# we use numeric gradient
summary(maxBFGS(loglik, start=1))
# you would probably prefer mean(n) instead of that ;-)
# Note also that maxLik is better suited for Maximum Likelihood
###
### Now an example of constrained optimization
###
f <- function(theta) {
  x <- theta[1]
  y <- theta[2]
  exp(-(x^2 + y^2))
  ## Note: you may want to use exp(- theta %*% theta) instead ;-)
}
## use constraints: x + y >= 1
A <- matrix(c(1, 1), 1, 2)
B <- -1
res <- maxNM(f, start=c(1,1), constraints=list(ineqA=A, ineqB=B),
print.level=1)
print(summary(res))
```

---

maximType *Type of Minimization/Maximization*

Description

Returns the type of optimisation. It should be returned by the optimisation routine.

Usage

```
maximType(x)
```

Arguments

x	object of class 'maxim' or another object which involves numerical optimisation.

Value

A text message, describing the involved optimisation algorithm

Author(s)

Ott Toomet, <otoomet@ut.ee>

See Also

maxNR

Examples

```
## maximise two-dimensional exponential hat.  Maximum is at c(2,1):
f <- function(a) exp(-(a[1] - 2)^2 - (a[2] - 1)^2)
m <- maxNR(f, start=c(0,0))
summary(m)
maximType(m)
## Now use BFGS maximisation.
m <- maxBFGS(f, start=c(0,0))
summary(m)
maximType(m)
```

---

maxLik *Maximum likelihood estimation*

Description

This is just a wrapper for maximisation routines which return object of class "maxLik". Corresponding methods can correctly handle the likelihood-specific properties of the estimate including the fact that inverse of negative hessian is the variance-covariance matrix.

Usage

```
maxLik(logLik, grad = NULL, hess = NULL, start, method,
constraints=NULL, ...)
```

Arguments

logLik	log-likelihood function. Must have the parameter vector as the first argument. Must return either a single log-likelihood value or a numeric vector where each component is log-likelihood corresponding to individual observations.
grad	gradient of log-likelihood. Must have the parameter vector as the first argument. Must return either single gradient vector with length equal to the number of parameters, or a matrix where each row corresponds to gradient vector of individual observations. If NULL, numeric gradient will be used.
hess	hessian of log-likelihood. Must have the parameter vector as the first argument. Must return a square matrix. If NULL, numeric gradient will be used.
start	numeric vector, initial value of parameters.
method	maximisation method, currently either "NR" (for Newton-Raphson), "BFGS" (for Broyden-Fletcher-Goldfarb-Shanno), "BFGSR" (for the BFGS algorithm implemented in R), "BHHH" (for Berndt-Hall-Hall-Hausman), "SANN" (for Simulated ANNealing), "CG" (for Conjugate Gradients), or "NM" (for Nelder-Mead). Lower-case letters (such as "nr" for Newton-Raphson) are allowed. If missing, a suitable method is selected automatically.

constraints	either NULL for unconstrained maximization or a list, specifying the constraints. See maxBFGS.
...	further arguments are passed to the selected maximisation routine, i.e. maxNR, maxBFGS, maxBFGSR, maxBHHH, maxSANN, maxCG, or maxNM (depending on argument method).

Details

maxLik "support" constrained optimization in the sense that constraints are passed further to the underlying optimization routines, and suitable default method is selected. However, no attempt is made to correct the resulting variance-covariance matrix. Hence, the inference may be wrong. A corresponding warning is issued by the summary method.

Value

object of class 'maxLik' which inherits from class 'maxim'. Components are identical to those of class 'maxim', see maxNR.

Warning

The constrained maximum likelihood estimation should be considered as experimental. In particular, the variance-covariance matrix is not corrected for constrained parameter space.

Author(s)

Ott Toomet <otoomet@ut.ee>, Arne Henningsen

See Also

maxNR, nlm and optim for different non-linear optimisation routines.

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t
gradlik <- function(theta) 1/theta - t
hesslik <- function(theta) -100/theta^2
## Estimate with numeric gradient and hessian
a <- maxLik(loglik, start=1, print.level=2)
print( a )
coef( a )
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, start=1)
print( a )
coef( a )
##
##
## Next, we give an example with vector argument:  Estimate the mean and
## variance of a random normal sample by maximum likelihood
##
```

```
loglik <- function(param) {
  mu <- param[1]
  sigma <- param[2]
  ll <- -0.5*N*log(2*pi) - N*log(sigma) - sum(0.5*(x - mu)^2/sigma^2)
  ll
}
x <- rnorm(1000, 1, 2) # use mean=1, stdd=2
N <- length(x)
res <- maxLik(loglik, start=c(0,1)) # use 'wrong' start values
print( res )
coef( res )
```

---

maxNR *Newton- and Quasi-Newton Maximization*

Description

Unconstrained and equality-constrained maximization based on the quadratic approximation (Newton) method. The Newton-Raphson, BFGS (Broyden 1970, Fletcher 1970, Goldfarb 1970, Shanno 1970), and BHHH (Berndt, Hall, Hall, Hausman 1974) methods are available.

Usage

```
maxNR(fn, grad = NULL, hess = NULL, start, print.level = 0,
      tol = 1e-08, reltol=sqrt(.Machine$double.eps), gradtol = 1e-06,
      steptol = 1e-10, lambdatol = 1e-06, qrtol = 1e-10, iterlim = 150,
      constraints = NULL, finalHessian = TRUE, bhhhHessian=FALSE,
      fixed = NULL, activePar = NULL, ... )
maxBFGSR(fn, grad = NULL, hess = NULL, start, print.level = 0,
      tol = 1e-8, reltol=sqrt(.Machine$double.eps), gradtol = 1e-6,
      steptol = 1e-10, lambdatol=1e-6, qrtol=1e-10,
      iterlim = 150,
      constraints = NULL, finalHessian = TRUE,
      fixed = NULL, activePar = NULL, ... )
maxBHHH(fn, grad = NULL, hess = NULL, start, print.level = 0,
      iterlim = 100, finalHessian = "BHHH", ... )
```

Arguments

fn	function to be maximized. It must have the parameter vector as the first argument and it must return either a single number or a numeric vector, which is summed. If the BHHH method is used and argument gradient is not given, fn must return a numeric vector of observation-specific likelihood values. If the parameters are out of range, fn should return NA. See details for constant parameters. fn may also return attributes "gradient" and/or "hessian". If these attributes are set, the algorithm uses the corresponding values as gradient and Hessian.

`grad` gradient of the objective function. It must have the parameter vector as the first argument and it must return either a gradient vector of the objective function, or a matrix, where *columns* correspond to individual parameters. The column sums are treated as gradient components. If `NULL`, finite-difference gradients are computed. If the BHHH method is used, `grad` must return a matrix, where rows corresponds to the gradient vectors of individual observations and the columns to the individual parameters. If `fn` returns an object with attribute `gradient`, this argument is ignored.

`hess` Hessian matrix of the function. It must have the parameter vector as the first argument and it must return the Hessian matrix of the objective function. If missing, finite-difference Hessians, based on `gradient`, are computed. Hessians are used for maximizations with the Newton-Raphson method but not for maximizations with the BFGS or BHHH method.

`start` initial value for the parameter vector.

`print.level` this argument determines the level of printing which is done during the minimization process. The default value of 0 means that no printing occurs, a value of 1 means that initial and final details are printed and a value of 2 means that full tracing information for every iteration is printed. Higher values will result in even more details.

`tol` stopping condition. Stop if the absolute difference between successive iterations is less than `tol`, return `code=2`.

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

`gradtol` stopping condition. Stop if the norm of the gradient less than `gradtol`, return `code=1`.

`steptol` stopping/error condition. If the quadratic approximation leads to lower function value instead of higher, or `NA`, the step length is halved and a new attempt is made. This procedure is repeated until step < `steptol`, thereafter `code=3` is returned.

`lambdatol` control whether the Hessian is treated as negative definite. If the largest of the eigenvalues of the Hessian is larger than `-lambdatol`, a suitable diagonal matrix is subtracted from the Hessian (quadratic hill-climbing) in order to enforce nagetive definiteness.

`qrtol` QR-decomposition tolerance

`iterlim` stopping condition. Stop if more than `iterlim` iterations, return `code=4`.

`constraints` either `NULL` for unconstrained optimization or a list with two components `eqA` and `eqB` for equality-constrained optimization $A\theta + B = 0$. The constrained problem is forwarded to `sumt`.

`finalHessian` how (and if) to calculate the final Hessian. Either `FALSE` (do not calculate), `TRUE` (use analytic/finite-difference Hessian) or `"bhhh"/"BHHH"` for the information equality approach. The latter approach is only suitable for maximizing log-likelihood functions. It requires the gradient/log-likelihood to be supplied by individual observations. Note that computing the (real, not BHHH) final Hessian does not carry any extra penalty for the NR method, but for the other methods.

bhhhHessian	logical. Indicating whether the approximation for the Hessian suggested by Bernd, Hall, Hall, and Hausman (1974) should be used.
fixed	parameters that should be fixed at their starting values: either a logical vector of the same length as argument start, a numeric (index) vector indicating the positions of the fixed parameters, or a vector of character strings indicating the names of the fixed parameters (parameter names are taken from argument start).
activePar	this argument is retained for backward compatibility only; please use argument fixed instead.
...	further arguments to fn, grad and hess. Further arguments to maxBHHH are also passed to maxNR.

Details

The idea of the Newton method is to approximate the function in a given location with a multidimensional parabola, and use the estimated maximum as the initial value for the next iteration. Such an approximation requires knowledge of both gradient and Hessian, the latter of which can be quite costly to compute. Several methods for approximating Hessian exist, including BFGS and BHHH.

The BHHH method (maxNR with argument bhhhHessian = TRUE) or maxBHHH) is suitable only for maximizing log-likelihood functions. It uses information equality in order to approximate the Hessian of the log-likelihood function. Hence, the log-likelihood values and its gradients must e calculated by individual observations. The Hessian is approximated as the negative of the sum of the outer products of the gradients of individual observations, or, in the matrix form, -t(gradient) %*% gradient = - crossprod(gradient).

The functions maxNR, maxBFGSR, and maxBHHH can work with constant parameters and related changes of parameter values. Constant parameters are useful if a parameter value is converging toward the boundary of support, or for testing. One way is to put fixed to non-NULL, specifying which parameters should be treated as constants.

However, when using maxNR or maxBHHH, parameters can also be fixed in runtime by signaling with fn. This may be useful if an estimation converges toward the edge of the parameter space possibly causing problems. The value of fn may have following attributes (only used by maxNR):

- constPar index vector. Which parameters are redefined to constant
- newVal a list with following components:
  - indexwhich parameters will have a new value
  - valthe new value of parameters

Hence, constVal specifies which parameters are treated as constants. newVal allows one to overwrite the existing parameter values, possibly the non-constant values as well. If the attribute newVal is present, the new function value need not to exceed the previous one (maximization is not performed in that step).

Value

list of class "maxim" with following components:

maximum	fn value at maximum (the last calculated value if not converged).

`estimate` estimated parameter value.

`gradient` vector, last gradient value which was calculated. Should be close to 0 if normal convergence.

`gradientObs` matrix of gradients at parameter value `estimate` evaluated at each observation (only if `grad` returns a matrix or `grad` is not specified and `fn` returns a vector).

`hessian` Hessian at the maximum (the last calculated value if not converged).

`code` return code:

- 1 gradient close to zero (normal convergence).
- 2 successive function values within tolerance limit (normal convergence).
- 3 last step could not find higher value (probably not converged). This is related to line search step getting too small, usually because hitting the boundary of the parameter space. It may also be related to attempts to move to a wrong direction because of numerical errors. In some cases it can be helped by changing `steptol`.
- 4 iteration limit exceeded.
- 5 Infinite value.
- 6 Infinite gradient.
- 7 Infinite Hessian.
- 8 Successive function values withing relative tolerance limit (normal convergence).
- 9 (BFGS) Hessian approximation cannot be improved because of gradient did not change. May be related to numerical approximation problems or wrong analytic gradient.
- 100 Initial value out of range.

`message` a short message, describing `code`.

`last.step` list describing the last unsuccessful step if `code=3` with following components:

- theta0 previous parameter value
- f0 `fn` value at `theta0`
- climb the movement vector to the maximum of the quadratic approximation

`fixed` logical vector, which parameters are constants.

`iterations` number of iterations.

`type` character string, type of maximization.

`constraints` A list, describing the constrained optimization (`NULL` if unconstrained). Includes the following components:

- type type of constrained optimization
- outer.iterations number of iterations in the constraints step
- barrier.value value of the barrier function

Warning

No attempt is made to ensure that user-provided analytic gradient/Hessian correct. However, the users are recommended to use `compareDerivatives` function, designed for this purpose. If analytic gradient/Hessian are wrong, the algorithm may not converge, or converge to a wrong point.

As the BHHH method (maxNR with argument bhhhHessian = TRUE or maxBHHH) uses the likelihood-specific information equality, it is only suitable for maximizing log-likelihood functions!

Quasi-Newton methods, including those mentioned above, do not work well in non-concave regions. This is especially the case with the implementation in maxBFGSR. The user is advised to experiment with various tolerance options to achieve convergence.

Author(s)

Ott Toomet <otoomet@ut.ee>, Arne Henningsen, function maxBFGSR was originally developed by Yves Croissant (and placed in 'mlogit' package)

References

Berndt, E., Hall, B., Hall, R. and Hausman, J. (1974): Estimation and Inference in Nonlinear Structural Models, *Annals of Social Measurement* 3, p. 653-665.

Broyden, C.G. (1970): The Convergence of a Class of Double-rank Minimization Algorithms, *Journal of the Institute of Mathematics and Its Applications* 6, p. 76-90.

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

Goldfeld, S.M. and Quandt, R.E. (1972): *Nonlinear Methods in Econometrics*. Amsterdam: North-Holland.

Goldfarb, D. (1970): A Family of Variable Metric Updates Derived by Variational Means, *Mathematics of Computation* 24, p. 23-26.

Greene, W.H., 2008, *Econometric Analysis*, 6th edition, Prentice Hall.

Shanno, D.F. (1970): Conditioning of Quasi-Newton Methods for Function Minimization, *Mathematics of Computation* 24, p. 647-656.

See Also

maxLik for a general framework for maximum likelihood estimation (MLE); maxBHHH for maximizations using the Berndt, Hall, Hall, Hausman (1974) algorithm (which is a wrapper function to maxNR); maxBFGS for maximization using the BFGS, Nelder-Mead (NM), and Simulated Annealing (SANN) method (based optim), also supporting inequality constraints; nlm for Newton-Raphson optimization; and optim for different gradient-based optimization methods.

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) sum(log(theta) - theta*t)
## Note the log-likelihood and gradient are summed over observations
gradlik <- function(theta) sum(1/theta - t)
hesslik <- function(theta) -100/theta^2
## Estimate with finite-difference gradient and Hessian
a <- maxNR(loglik, start=1, print.level=2)
summary(a)
## You would probably prefer 1/mean(t) instead ;-)
## Estimate with analytic gradient and Hessian
```

```
a <- maxNR(loglik, gradlik, hesslik, start=1)
summary(a)

## BFGS estimation with finite-difference gradient
a <- maxBFGSR( loglik, start=1 )
summary(a)
## BFGS estimation with analytic gradient
a <- maxBFGSR( loglik, gradlik, start=1 )
summary(a)

## For the BHHH method we need likelihood values and gradients
## of individual observations
loglikInd <- function(theta) log(theta) - theta*t
gradlikInd <- function(theta) 1/theta - t
## Estimate with finite-difference gradient
a <- maxBHHH(loglikInd, start=1, print.level=2)
summary(a)
## Estimate with analytic gradient
a <- maxBHHH(loglikInd, gradlikInd, start=1)
summary(a)

##
## Next, we give an example with vector argument:  Estimate the mean and
## variance of a random normal sample by maximum likelihood
## Note: you might want to use maxLik instead
##
loglik <- function(param) {
  mu <- param[1]
  sigma <- param[2]
  ll <- -0.5*N*log(2*pi) - N*log(sigma) - sum(0.5*(x - mu)^2/sigma^2)
  ll
}
x <- rnorm(1000, 1, 2) # use mean=1, stdd=2
N <- length(x)
res <- maxNR(loglik, start=c(0,1)) # use 'wrong' start values
summary(res)
###
### Now an example of constrained optimization
###
## We maximize exp(-x^2 - y^2) where x+y = 1
f <- function(theta) {
  x <- theta[1]
  y <- theta[2]
  exp(-(x^2 + y^2))
  ## Note: you may want to use exp(- theta %*% theta) instead ;-)
}
## use constraints: x + y = 1
A <- matrix(c(1, 1), 1, 2)
B <- -1
res <- maxNR(f, start=c(0,0), constraints=list(eqA=A, eqB=B), print.level=1)
print(summary(res))
```

nIter *Return number of iterations for iterative models*

Description

Returns the number of iterations for iterative models. The default method assumes presence of a component `iterations` in `x`.

Usage

```
nIter(x, ...)
## Default S3 method:
nIter(x, ...)
```

Arguments

`x`	a statistical model, or a result of maximisation, such as created by `maxLik` or `maxNR`
`...`	further arguments for methods

Details

This is a generic function. The default method returns the component `x$iterations`.

Value

numeric, number of iterations

Author(s)

Ott Toomet, <otoomet@econ.au.dk>

See Also

`maxLik`, `maxNR`

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) sum(log(theta) - theta*t)
## Estimate with numeric gradient and numeric Hessian
a <- maxNR(loglik, start=1)
nIter(a)
```

nObs.maxLik *Number of Observations*

Description

Returns the number of observations for statistical models estimated by Maximum Likelihood using maxLik.

Usage

```
## S3 method for class 'maxLik'
nObs(x, ...)
```

Arguments

x	a statistical model estimated by Maximum Likelihood using maxLik.
...	further arguments (currently ignored).

Details

The nObs method for objects of class "maxLik" can return the number of observations only if maxLik was called with argument grad equal to a gradient function or (if no gradient function is specified) argument logLik equal to a log-likelihood function that return the gradients or log-likelihood values, respectively, for each observation.

Value

numeric, number of observations

Author(s)

Arne Henningsen, Ott Toomet

See Also

nObs, maxLik, nParam.

Examples

```
## fit a normal distribution by ML
# generate a variable from normally distributed random numbers
x <- rnorm( 100, 1, 2 )
# log likelihood function (for individual observations)
llf <- function( param ) {
   return( dnorm( x, mean = param[ 1 ], sd = param[ 2 ], log = TRUE ) )
}
## ML method
ml <- maxLik( llf, start = c( mu = 0, sigma = 1 ) )
```

```
# return number of onservations
nObs( ml )
```

`nParam.maxim` *Number of model parameters*

Description

This function returns the number of model parameters.

Usage

```
## S3 method for class 'maxim'
nParam(x, free=FALSE, ...)
```

Arguments

`x`	a model returned by a maximisation method from the **maxLik** package.
`free`	logical, whether to report only the free parameters or the total number of parameters (default)
`...`	other arguments for methods

Details

Free parameters are the parameters with no equality restrictions. Some parameters may be restricted (e.g. sum of two probabilities may be restricted to equal unity). In this case the total number of parameters may depend on the normalisation.

Value

Number of parameters in the model

Author(s)

Ott Toomet, <otoomet@econ.au.dk>

See Also

`nObs` for number of observations

Examples

```
## fit a normal distribution by ML
# generate a variable from normally distributed random numbers
x <- rnorm( 100, 1, 2 )
# log likelihood function (for individual observations)
llf <- function( param ) {
   return( dnorm( x, mean = param[ 1 ], sd = param[ 2 ], log = TRUE ) )
}
## ML method
ml <- maxLik( llf, start = c( mu = 0, sigma = 1 ) )
# return number of parameters
nParam( ml )
```

---

numericGradient *Functions to Calculate Numeric Derivatives*

Description

Calculate (central) numeric gradient and Hessian. numericGradient accepts vector-valued functions.

Usage

```
numericGradient(f, t0, eps=1e-06, fixed, ...)
numericHessian(f, grad=NULL, t0, eps=1e-06, fixed, ...)
numericNHessian(f, t0, eps=1e-6, fixed, ...)
```

Arguments

f	function to be differentiated. The first argument must be the parameter vector with respect to which it is differentiated. For numeric gradient, f may return a (numeric) vector, for Hessian it should return a numeric scalar
grad	function, gradient of f
t0	vector, the value of parameters
eps	numeric, the step for numeric differentiation
fixed	logical vector, length of which equal the length of the parameter. Derivative is calculated only along the parameters for which it is FALSE, NA returned for the others. If missing, all parameters are treated as active.
...	furter arguments for f

Details

numericGradient numerically differentiates a (vector valued) function with respect to it's (vector valued) argument. If the functions value is a NVal * 1 vector and the argument is Npar * 1 vector, the resulting gradient is a NVal * NPar matrix.

numericHessian checks whether a gradient function is present and calculates a gradient of the gradient (if present), or full numeric Hessian (numericNHessian) if grad is NULL.

Value

Matrix. For `numericGradient`, the number of rows is equal to the length of the function value vector, and the number of columns is equal to the length of the parameter vector.

For the `numericHessian`, both numer of rows and columns is equal to the length of the parameter vector.

Warning

Be careful when using numerical differentiation in optimisation routines. Although quite precise in simple cases, they may work very poorly in more complicated conditions.

Author(s)

Ott Toomet <otoomet@gmail.com>

See Also

`compareDerivatives`, `deriv`

Examples

```
# A simple example with Gaussian bell surface
f0 <- function(t0) exp(-t0[1]^2 - t0[2]^2)
numericGradient(f0, c(1,2))
numericHessian(f0, t0=c(1,2))

# An example with the analytic gradient
gradf0 <- function(t0) -2*t0*f0(t0)
numericHessian(f0, gradf0, t0=c(1,2))
# The results should be similar as in the previous case

# The central numeric derivatives have usually quite a high precision
compareDerivatives(f0, gradf0, t0=1:2)
# The difference is around 1e-10
```

`returnCode` *Return code for optimisation and other objects*

Description

This function gives the return code of various optimisation objects. The return code gives a brief information about the success or problems, occured during the optimisation (see documentation for the corresponding function).

Usage

```
returnCode(x, ...)
## Default S3 method:
returnCode(x, ...)
```

Arguments

x	object, usually an estimator, achieved by optimisation
...	further arguments for other methods

Details

The default methods returns component returnCode.

Value

Integer, the success code of optimisation procedure. However, different optimisation routines may define it in a different way.

Author(s)

Ott Toomet, <otoomet@ut.ee>

See Also

returnMessage, maxNR

Examples

```
## maximise the exponential bell
f1 <- function(x) exp(-x^2)
a <- maxNR(f1, start=2)
returnCode(a) # should be success (1 or 2)
## Now try to maximise log() function
f2 <- function(x) log(x)
a <- maxNR(f2, start=2)
returnCode(a) # should give a failure (4)
```

---

returnMessage *Information about the optimisation process*

Description

This function returns a short message, summarising the outcome of the statistical process, typically optimisation. The message should describe either the type of the convergence, or the problem. returnMessage is a generic function, with methods for various optimisation algorithms.

Usage

```
returnMessage(x, ...)
## S3 method for class 'maxim'
returnMessage(x, ...)
## S3 method for class 'maxLik'
returnMessage(x, ...)
```

Arguments

x	object, should orginate from an optimisation problem
...	further arguments to other methods.

Details

The default methods returns component returnMessage.

Value

Character string, the message describing the success or failure of the statistical procedure.

Author(s)

Ott Toomet, <otoomet@ut.ee>

See Also

returnCode, maxNR

Examples

```
## maximise the exponential bell
f1 <- function(x) exp(-x^2)
a <- maxNR(f1, start=2)
returnMessage(a) # should be success (1 or 2)
## Now try to maximise log() function
f2 <- function(x) log(x)
a <- maxNR(f2, start=2)
returnMessage(a) # should give a failure (4)
```

---

summary.maxim *Summary method for maximisation/minimisation*

Description

Summarises the maximisation results

Usage

```
## S3 method for class 'maxim'
summary( object, hessian=FALSE, unsucc.step=FALSE, ... )
```

Arguments

object	optimisation result, object of class maxim. See maxNR.
hessian	logical, whether to display Hessian matrix.
unsucc.step	logical, whether to describe last unsuccesful step if code == 3
...	currently not used.

Value

Object of class `summary.maxim`, intended to print with corresponding print method. There are following components:

`type`	type of maximisation.
`iterations`	number of iterations.
`code`	exit code (see `maxNR`.)
`message`	a brief message, explaining code.
`unsucc.step`	description of last unsuccessful step, only if requested and `code` == 3
`maximum`	function value at maximum
`estimate`	matrix with following columns: • resultscoefficient estimates at maximum • gradientestimated gradient at maximum
`constraints`	information about the constrained optimization. Passed directly further from `maxim`-object. `NULL` if unconstrained maximization.
`hessian`	estimated hessian at maximum, only if requested

Author(s)

Ott Toomet <siim@obs.ee>

See Also

`maxNR`

Examples

```
## minimize a 2D quadratic function:
f <- function(b) {
  x <- b[1]; y <- b[2];
    val <- (x - 2)^2 + (y - 3)^2
    attr(val, "gradient") <- c(2*x - 4, 2*y - 6)
    attr(val, "hessian") <- matrix(c(2, 0, 0, 2), 2, 2)
    val
}
## Note that NR finds the minimum of a quadratic function with a single
## iteration.  Use c(0,0) as initial value.
result1 <- maxNR( f, start = c(0,0) )
summary( result1 )
## Now use c(1000000, -777777) as initial value and ask for hessian
result2 <- maxNR( f, start = c( 1000000, -777777))
summary( result2 )
```

summary.maxLik *summary the Maximum-Likelihood estimation*

Description

Summary the Maximum-Likelihood estimation including standard errors and t-values.

Usage

```
## S3 method for class 'maxLik'
summary(object, eigentol=1e-12, ... )
## S3 method for class 'summary.maxLik'
coef(object, ...)
```

Arguments

object	object of class 'maxLik', or 'summary.maxLik', usually a result from Maximum-Likelihood estimation.
eigentol	nonzero print limit on the range of the absolute values of the hessian. Specifically, define: absEig <- eigen(hessian(object), symmetric=TRUE)[['values']] Then compute and print t values, p values, etc. only if min(absEig) > (eigentol * max(absEig)).
...	currently not used.

Value

summary.maxLik returns an object of class 'summary.maxLik' with following components:

type	type of maximisation.
iterations	number of iterations.
code	code of success.
message	a short message describing the code.
loglik	the loglik value in the maximum.
estimate	numeric matrix, the first column contains the parameter estimates, the second the standard errors, third t-values and fourth corresponding probabilities.
fixed	logical vector, which parameters are treated as constants.
NActivePar	number of free parameters.
constraints	information about the constrained optimization. Passed directly further from maxim-object. NULL if unconstrained maximization.

coef.summary.maxLik returns the matrix of estimated values, standard errors, and

$$t$$

- and

$$p$$

-values.

Author(s)

Ott Toomet <otoomet@ut.ee>, Arne Henningsen

See Also

maxLik

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t
gradlik <- function(theta) 1/theta - t
hesslik <- function(theta) -100/theta^2
## Estimate with numeric gradient and hessian
a <- maxLik(loglik, start=1, print.level=2)
summary(a)
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, start=1)
summary(a)
```

---

sumt *Equality-constrained optimization*

Description

Sequentially Unconstrained Maximization Technique (SUMT) based optimization for linear equality constraints.

This implementation is mostly intended to be called from other maximization routines, such as maxNR.

Usage

```
sumt(fn, grad=NULL, hess=NULL,
start,
maxRoutine, constraints,
SUMTTol = sqrt(.Machine$double.eps),
SUMTPenaltyTol = sqrt(.Machine$double.eps),
SUMTQ = 10,
SUMTRho0 = NULL,
print.level = 0, SUMTMaxIter = 100, ...)
```

Arguments

fn	function of a (single) vector parameter. The function may have more arguments, but those are not treated as parameter
grad	gradient function of fn. NULL if missing

hess	hessian matrix of the fn. NULL if missing
start	initial value of the parameter.
maxRoutine	maximization algorithm
constraints	list, information for constrained maximization. Currently two components are supported: eqA and eqB for linear equality constraints: $A\beta + B = 0$. The user must ensure that the matrices A and B are conformable.
SUMTTol	stopping condition. If the coefficient of successive outer iterations are close enough, i.e. maximum of the absolute value over the component difference is smaller than SUMTTol, the algorithm stops. Note this does not necessarily mean satisfying the constraints. In case of the penalty function is too 'weak', SUMT may repeatedly find the same optimum. In that case a warning is issued. The user may try to set SUMTTol to a lower value, e.g. to zero.
SUMTPenaltyTol	stopping condition. If barrier value (also called penalty) $(A\beta + B)'(A\beta + B)$ is less than SUMTTol, the algorithm stops
SUMTQ	a double greater than one controlling the growth of the rho as described in Details. Defaults to 10.
SUMTRho0	Initial value for rho. If not specified, a (possibly) suitable value is selected. See Details. One should consider supplying SUMTRho0 in case where the unconstrained problem does not have a maximum, or the maximum is too far from the constrained value. Otherwise the algorithm may pick values too to achive convergence.
print.level	Integer, debugging information. Larger number print more details.
SUMTMaxIter	Maximum SUMT iterations
...	Other arguments to maxRoutine and fn.

Details

The Sequential Unconstrained Minimization Technique is a heuristic for constrained optimization. To minimize a function f subject to constraints, one employs a non-negative function P penalizing violations of the constraints, such that $P(x)$ is zero iff x satisfies the constraints. One iteratively minimizes $L(x) + \varrho_k P(x)$, where the ϱ values are increased according to the rule $\varrho_{k+1} = q\varrho_k$ for some constant $q > 1$, until convergence is obtained in the sense that the barrier value $P(x)'P(x)$ is close to zero. Note that there is no guarantee that global (approximately) constrained optima are found. Standard practice would recommend to use the best solution found in "sufficiently many" replications of the algorithm.

The unconstrained minimizations are carried out by either any of the maximization algorithms in the **maxLik**, such as maxNR. Analytic gradient and hessian are used if provided, numeric ones otherwise.

Value

Object of class 'maxim'. In addition, a component

constraints	A list, describing the constrained optimization. Includes the following components:

- typetype of constrained optimization
- barrier.valuevalue of the penalty function at maximum
- codecode for the stopping condition
- messagea short message, describing the stopping condition
- outer.iterationsnumber of iterations in the SUMT step

Note

It may be a lot more efficient to embrace the actual function to be optimized to an outer function, which calculates the actual parameters based on a smaller set of parameters and the constraints.

Author(s)

Ott Toomet <otoomet@ut.ee>, Arne Henningsen

See Also

sumt

---

vcov.maxLik *Variance Covariance Matrix of maxLik objects*

Description

Extract variance-covariance matrices of objects of class maxLik.

Usage

```
   ## S3 method for class 'maxLik'
vcov( object, eigentol=1e-12, ... )
```

Arguments

object	an object of class probit or maxLik.
eigentol	nonzero print limit on the range of the absolute values of the hessian. Specifically, define: absEig <- eigen(hessian(object), symmetric=TRUE)[['values']] Then compute and print t values, p values, etc. only if min(absEig) > (eigentol * max(absEig)).
...	further arguments (currently ignored).

Value

the estimated variance covariance matrix of the coefficients. In case of the estimated Hessian is singular, it's values are Inf. The values corresponding to fixed parameters are zero.

Author(s)

Arne Henningsen, Ott Toomet <otoomet@ut.ee>

See Also

vcov, maxLik.

Examples

```
## ML estimation of exponential duration model:
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t
gradlik <- function(theta) 1/theta - t
hesslik <- function(theta) -100/theta^2
## Estimate with numeric gradient and hessian
a <- maxLik(loglik, start=1, print.level=2)
vcov(a)
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, start=1)
vcov(a)
```

Index