Package 'maxLik'

November 9, 2015

Version 1.3-4

Date 2015-11-08

Title Maximum Likelihood Estimation and Related Tools

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

Maintainer Ott Toomet <otoomet@gmail.com>

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

Imports sandwich

Description Functions for Maximum Likelihood (ML) estimation and non-linear optimization, and related tools. It includes a unified way to call different optimizers, and classes and methods to handle the results from the ML viewpoint. It also includes a number of convenience tools for testing and developing your own models.

License GPL (>= 2)

ByteCompile yes

URL http://www.maxLik.org

NeedsCompilation no

Repository CRAN

Date/Publication 2015-11-09 14:41:52

R topics documented:

`maxLik-package` *Maximum Likelihood Estimation*

Description

This is a set of functions and tools to perform Maximum Likelihood (ML) estimation. The focus of the package is on the non-linear optimization from the ML viewpoint, and it provides several convenience wrappers and tools, like BHHH algorithm and extraction of variance-covariance matrix.

Details

"maxLik" package is a set of convenience tools and wrappers to perform Maximum Likelihood (ML) analysis. It includes a) wrappers for several existing optimizers (implemented by `optim`); b) original optimizers, including Newton-Raphson; and c) several convenience tools to use these optimizers from the ML perspective. Examples are BHHH optimization (`maxBHHH`) and utilities that extract standard errors from the estimates. Other highlights include a unified interface for all included optimizers, tools to check the programmed analytic derivatives, and constrained optimization.

From the user's perspective, the central function in the package is `maxLik`. In the simplest form it takes two arguments: the log-likelihood function, and a vector of parameters' start values. It returns an object of class 'maxLik' with convenient methods such as `summary`, `coef`, and `stdEr`. It also supports a plethora of other arguments, for instance one can supply analytic gradient and Hessian, select the desired optimizer, and control the optimization in different ways.

One of the most useful utility functions in the package is `compareDerivatives` that allows one to compare the analytic and numeric derivatives for debugging the derivative code. Another useful function is `condiNumber` for analyzing multicollinearity problems in the estimated models.

Author(s)

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

Maintainer: Ott Toomet <otoomet@gmail.com>

Examples

```
## estimate mean and variance of normal random vector
set.seed( 123 )
x <- rnorm(50, 1, 2 )

## log likelihood function.
## Note: 'param' is a vector
llf <- function( param ) {
   mu <- param[ 1 ]
   sigma <- param[ 2 ]
   llValue <- dnorm(x, mean=mu, sd=sigma, log=TRUE)
   return(sum(llValue))
}

## Estimate it.  Take standard normal as start values
ml <- maxLik( llf, start = c(mu=0, sigma=1) )
print(summary(ml))
## Estimates close to c(1,2) :-)

## Example how to use maxLik in your own function and allow users
## to override the default parameters
##
## 'estimate': user contructed estimation routine
## Note: it accepts both 'control' and '...'
estimate <- function(control=NULL, ...) {
   return(maxLik(llf, start=c(1,1),
                 control=c(list(iterlim=100), control),
                           # user-supplied 'control' overrides default
                           # 'iterlim=100'
                 ...))
}
m <- estimate(control=list(iterlim=1), fixed=2)
                           # user can override default 'iterlim' and
                           # supply additional parameters ('fixed')
show(maxControl(m))
                           # iterlim should be 1
print(coef(m))
                           # sigma should be 1.000
```

---

activePar *free parameters under maximisation*

Description

Return a logical vector, indicating which parameters were free under maximisation, as opposed to the fixed parameters that are treated as constants. See argument "fixed" for maxNR.

Usage

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

Arguments

x	object, created by a maximisation routine, or derived from a maximisation object.
...	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

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)
# keep the first parameter constant
cons <- maxNR(f, start=1:2, fixed=c(TRUE,FALSE))
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 also the documentation for the corresponding generic functions

Usage

```
## S3 method for class 'maxLik'
AIC(object, ..., k=2)
## S3 method for class 'maxim'
coef(object, ...)
## S3 method for class 'maxLik'
stdEr(x, eigentol=1e-12, ...)
```

Arguments

object	a 'maxLik' object (or a 'maxim' object for coef)
k	numeric, the penalty per parameter to be used; the default 'k = 2' is the classical AIC.
x	a 'maxLik' object
eigentol	The standard errors are only calculated if the ration of the smallest and largest eigenvalue of the Hessian matrix is less than "eigentol". Otherwise the Hessian is treated as singular.
...	other arguments for methods

Details

AIC calculates Akaike's Information Criterion (and other information criteria).

coef extracts the estimated parameters (model's coefficients).

stdEr extracts standard errors (using the Hessian matrix).

Examples

```
## estimate mean and variance of normal random vector
set.seed( 123 )
x <- rnorm(50, 1, 2 )
## log likelihood function.
## Note: 'param' is a vector
llf <- function( param ) {
   mu <- param[ 1 ]
   sigma <- param[ 2 ]
   return(sum(dnorm(x, mean=mu, sd=sigma, log=TRUE)))
}
```

```
## Estimate it.  Take standard normal as start values
ml <- maxLik(llf, start = c(mu=0, sigma=1) )

coef(ml)
stdEr(ml)
AIC(ml)
```

`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. In case of the simple Maximum Likelihood, it is equal to the variance covariance matrix of the parameters, multiplied by the number of observations.

Warnings

The **sandwich** package is required for this function.

This method works only if the observaton-specific gradient information was available for the estimation. This is the case if the observation-specific gradient was supplied (see the `grad` argument for `maxLik`), or the log-likelihood function returns a vector of observation-specific values.

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 related diagnostics information. It is intended for testing and debugging code for analytic derivatives for maximization 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, in that case the derivative will be a matrix.
`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`	numeric vector, parameter at which the derivatives are compared. The derivative is taken with respect to this vector. both `fm grad` (if function) and `hess` (if present) must accept this value as the first parameter.
`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

Analytic derivatives (and Hessian) substantially improve the estimation speed and reliability. However, these are typically hard to program. This utility compares the programmed result and the (internally calculated) numeric derivative. For every component of `f`, it prints the parameter value, analytic and numeric derivative, and their relative difference

$$\text{rel.diff} = \frac{\text{analytic} - \text{numeric}}{\frac{1}{2}(\text{analytic} + \text{numeric})}.$$

If analytic = 0 = numeric, the rel.diff = 0. If analytic derivatives are correct and the function is sufficiently smooth, expect the relative differences to be less than 10^{-7}.

Value

A list with 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))
##
## Example of normal log-likelihood.  Two-parameter
## function.
##
x <- rnorm(100, 1, 2) # generate rnorm x
l <- function(b) sum(dnorm(x, mean=b[1], sd=b[2], log=TRUE))
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)
```

`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. It is a generic function with a default method, and a method for `maxLik` objects.

Usage

```
condiNumber(x, ...)
## Default S3 method:
condiNumber(x, exact = FALSE, norm = FALSE,
   printLevel=print.level, print.level=1, digits = getOption( "digits" ), ... )
## 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 kappa)
`norm`	logical, whether the columns should be normalised to have unit norm
`printLevel`	numeric, positive value will output the numbers during the calculations. Useful for interactive work.
`print.level`	same as 'printLevel', for backward compatibility
`digits`	minimal number of significant digits to print (only relevant if argument `print.level` is larger than zero).
`...`	Further arguments to `condiNumber.default` are currently ignored; further arguments to `condiNumber.maxLik` are passed to `condiNumber.default`.

Details

Statistical model often fail because of a high correlation between the explanatory variables in the 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 help to understand the problem.

`condiNumber` inspects the matrices column-by-column and indicates which variables lead to a jump in the condition number (cause singularity). If the matrix column name does not immediately indicate the problem, one may run an OLS model by estimating this column using all the previous columns as explanatory variables. Those columns that explain almost all the variation in the current one will have very high t-values.

Value

Invisible vector of condition numbers by column. If the start values for `maxLik` are named, the condition numbers are named accordingly.

Author(s)

Ott Toomet

References

Greene, W. (2012): *Econometrics Analysis*, 7th edition, p. 130.

See Also

`kappa`

Examples

```
set.seed(0)
## generate a simple nearly 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 outlandish estimates and standard errors
                  # while R^2 is 0.88. This suggests multicollinearity
condiNumber(model.matrix(m))   # note the value 'explodes' at x3
## we may test the results further:
print(summary(lm(x3 ~ -1 + x1 + x2)))
# Note the extremely high t-values and R^2: x3 is (almost) completely
# explained by x1 and x2
```

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 log-likelihood gradients at the estimated parameter value evaluated at each observation. Observations in rows, parameters in columns.

Warnings

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

This method works only if the observaton-specific gradient information was available for the estimation. This is the case of the observation-specific gradient was supplied (see the grad argument for maxLik), or the log-likelihood function returns a vector of observation-specific values.

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 )
head(estfun( a ), 10)

## Estimate with analytic gradient.
## Note: it returns a vector
gradlik <- function(theta) 1/theta - t
b <- maxLik(loglik, gradlik, start=1)
head(estfun( b ), 10)
```

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

Description

Combine variable parameters with with fixed parameters 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`	Parameter values 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

This function first confirms that `length(x) + length(xFixed) == length(xFull)`. Next,

- If `xFull` has names, match at least `xFixed` by name.
- Else `xFull = c(x, xFixes)`, the default.

Finally, call `fnFull(xFull, ...)`.

Value

value returned by `fnFull`

Author(s)

Spencer Graves

See Also

optim dlmMLE maxLik maxNR

Examples

```
##
## Example with 'optim'
##
fn <- function(x) (x[2]-2*x[1])^2
# note: true minimum is 0 on line 2*x[1] == x[2]
fullEst <- optim(par=c(1,1), method="BFGS", fn=fn)
fullEst$par
# par = c(0.6, 1.2) at minimum (not convex)

# Fix the last component to 4
est4 <- optim(par=1, fn=fnSubset, method="BFGS", fnFull=fn, xFixed=4)
est4$par
# now there is a unique minimun x[1] = 2

# Fix the first component
fnSubset(x=1, fnFull=fn, xFixed=c(a=4), xFull=c(a=1, b=2))
# After substitution:  xFull = c(a=4, b=1),
# so fn = (1 - 2*4)^2 = (-7)^2 = 49

est4. <- optim(par=1, fn=fnSubset, method="BFGS",
               fnFull=fn, xFixed=c(a=4),
               xFull=c(a=1, b=2))
est4.$par
# At optimum: xFull=c(a=4, b=8),
# so fn = (8 - 2*4)^2 = 0

##
## Example with 'maxLik'
##
fn2max <- function(x) -(x[2]-2*x[1])^2
# -> need to have a maximum
max4 <- maxLik(fnSubset, start=1, fnFull=fn2max, xFixed=4)
summary(max4)
# Similar result using fixed parameters in maxNR, called by maxLik
max4. <- maxLik(fn2max, start=c(1, 4), fixed=2)
summary(max4.)
```

---

hessian *Hessian matrix*

Description

This function extracts the Hessian of the objective function at optimum. The Hessian information should be supplied by the underlying optimization algorithm, possibly by an approximation.

Usage

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

Arguments

x	an optimization result of class 'maxim' or 'maxLik'
...	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

See Also

maxLik, activePar, condiNumber

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(100) # 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)
# Note that now 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 scalar numeric, log likelihood of the estimated model

Author(s)

Arne Henningsen, Ott Toomet

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, adding constrained optimization and fixed parameters.

Usage

```
maxBFGS(fn, grad=NULL, hess=NULL, start, fixed=NULL,
   control=NULL,
   constraints=NULL,
   finalHessian=TRUE,
   parscale=rep(1, length=length(start)),
   ... )

maxCG(fn, grad=NULL, hess=NULL, start, fixed=NULL,
   control=NULL,
   constraints=NULL,
   finalHessian=TRUE,
   parscale=rep(1, length=length(start)), ...)

maxSANN(fn, grad=NULL, hess=NULL, start, fixed=NULL,
   control=NULL,
   constraints=NULL,
   finalHessian=TRUE,
   parscale=rep(1, length=length(start)),
   ... )

maxNM(fn, grad=NULL, hess=NULL, start, fixed=NULL,
   control=NULL,
   constraints=NULL,
   finalHessian=TRUE,
   parscale=rep(1, length=length(start)),
   ...)
```

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 a vector of observation-specific likelihood values. Those are summed internally where necessary. If the parameters are out of range, fn should return NA. See details for constant parameters.
grad	gradient of fn. Must have the parameter vector as the first argument. If NULL, numeric gradient is used (maxNM and maxSANN do not use 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 `fn`. Not used by any of these methods, included for compatibility with `maxNR`.

`start` initial values for the parameters. If start values are named, those names are also carried over to the results.

`fixed` parameters to be treated as constants at their `start` values. If present, it is treated as an index vector of `start` parameters.

`control` list of control parameters or a 'MaxControl' object. If it is a list, the default values are used for the parameters that are left unspecified by the user. These functions accept the following parameters:

reltol sqrt(.Machine$double.eps), stopping condition. Relative convergence tolerance: the algorithm stops if the relative improvement between iterations is less than 'reltol'. Note: for compatibility reason 'tol' is equivalent to 'reltol' for optim-based optimizers.

iterlim integer, maximum number of iterations. Default values are 200 for 'BFGS', 500 ('CG' and 'NM'), and 10000 ('SANN'). Note that 'iteration' may mean different things for different optimizers.

printLevel integer, larger number prints more working information. Default 0, no information.

nm_alpha 1, Nelder-Mead simplex method reflection coefficient (see Nelder & Mead, 1965)

nm_beta 0.5, Nelder-Mead controction coefficient

nm_gamma 2, Nelder-Mead expansion coefficient

sann_cand `NULL` or a function for `"SANN"` algorithm to generate a new candidate point; if `NULL`, Gaussian Markov kernel is used (see argument `gr` of `optim`).

sann_temp 10, starting temperature for the "SANN" cooling schedule. See `optim`.

sann_tmax 10, number of function evaluations at each temperature for the "SANN" optimizer. See `optim`.

sann_randomSeed 123, integer to seed random numbers to ensure replicability of "SANN" optimization and preserve `R` random numbers. Use options like `sann_randomSeed=Sys.time()` or `sann_randomSeed=sample(100,1)` if you want stochastic results.

`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 $A\theta + B > 0$. More than one row in `ineqA` and `ineqB` corresponds to more than one linear constraint, in that case all these must be zero (equality) or positive (inequality constraints). The equality-constrained problem is forwarded to `sumt`, the inequality-constrained case to `constrOptim2`.

`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.

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)
...	further arguments for fn and grad.

Details

In order to provide a consistent interface, all these functions also accept arguments that other optimizers use. For instance, maxNM accepts the 'grad' argument despite being a gradient-less method.

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 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 parameter vector 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: **type** type of constrained optimization **outer.iterations** number of iterations in the constraints step **barrier.value** value of the barrier function
control	the optimization control parameters in the form of a MaxControl object.

Author(s)

Ott Toomet, Arne Henningsen

References

Nelder, J. A. & Mead, R. A, Simplex Method for Function Minimization, The Computer Journal, 1965, 7, 308-313

See Also

optim, nlm, maxNR, maxBHHH, maxBFGSR for a maxNR-based BFGS implementation.

Examples

```
# Maximum Likelihood estimation 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))
  ## 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),
control=list(printLevel=1))
print(summary(res))
```

MaxControl-class *Class* "MaxControl"

Description

This is the structure that holds the optimization control options. The corresponding constructors take the parameters, perform consistency checks, and return the control structure. Alternatively, it overwrites the supplied parameters in an existing MaxControl structure. There is also a method to extract the control structure from the estimated 'maxim'-objects.

Slots

The default values and definition of the slots:

tol 1e-8, stopping condition for maxNR and related optimizers. Stop if the absolute difference between successive iterations is less than tol, returns code 2.

reltol sqrt(.Machine$double.eps), relative convergence tolerance (used by maxNR related optimizers, and optim-based optimizers. The algorithm stops if it iteration increases the value by less than a factor of `reltol*(abs(val) + reltol)`. Returns code 2.

gradtol 1e-6, stopping condition for maxNR and related optimizers. Stops if norm of the gradient is less than `gradtol`, returns code 1.

steptol 1e-10, stopping/error condition for maxNR and related optimizers. If `qac == "stephalving"` and the quadratic approximation leads to a worse, instead of a better value, or to NA, the step length is halved and a new attempt is made. If necessary, this procedure is repeated until `step < steptol`, thereafter code 3 is returned.

lambdatol 1e-6, (for maxNR related optimizers) controls whether Hessian is treated as negative definite. If the largest of the eigenvalues of the Hessian is larger than `-lambdatol` (Hessian is not negative definite), a suitable diagonal matrix is subtracted from the Hessian (quadratic hill-climbing) in order to enforce negative definiteness.

qac "stephalving", character, Qadratic Approximation Correction for maxNR related optimizers. When the new guess is worse than the initial one, program attempts to correct it: `"stephalving"` decreases the step but keeps the direction. `"marquardt"` uses *Marquardt (1963)* method by decreasing the step length while also moving closer to the pure gradient direction. It may be faster and more robust choice in areas where quadratic approximation behaves poorly.

qrtol 1e-10, QR-decomposition tolerance for Hessian inversion in maxNR related optimizers.

marquardt_lambda0 0.01, a positive numeric, initial correction term for *Marquardt (1963)* correction in maxNR-related optimizers

marquardt_lambdaStep 2, how much the *Marquardt (1963)* correction is decreased/increased at successful/unsuccesful step for maxNR related optimizers

marquardt_maxLambda 1e12, maximum allowed correction term for maxNR related optimizers. If exceeded, the algorithm exits with return code 3.

nm_alpha 1, Nelder-Mead simplex method reflection factor (see Nelder \& Mead, 1965)

nm_beta 0.5, Nelder-Mead contraction factor

nm_gamma 2, Nelder-Mead expansion factor

sann_cand NULL or a function for "SANN" algorithm to generate a new candidate point; if NULL, Gaussian Markov kernel is used (see argument gr of optim).

sann_temp 10, starting temperature for the "SANN" cooling schedule. See optim.

sann_tmax 10, number of function evaluations at each temperature for the "SANN" optimizer. See optim.

sann_randomSeed 123, integer to seed random numbers to ensure replicability of "SANN" optimization and preserve R random numbers. Use options like `SANN_randomSeed=Sys.time()` or `SANN_randomeSeed=sample(1000,1)` if you want stochastic results.

iterlim 150, stopping condition. Stop if more than `iterlim` iterations performed. Note that 'iteration' may mean different things for different optimzers.

printLevel 0, the level of verbosity. Larger values print more information. Result depends on the optimizer. Form `print.level` is also accepted by the methods for compatibility.

Methods

maxControl `(...)` creates a "MaxControl" object. The arguments must be in the form `option1 = value1, option2 = value2, ...`. In case there are more than one option with similar name, only the last one is taken into account. This allows the user to override default parameters in the control list. See example in maxLik-package.

maxControl `(x = "MaxControl", ...)` overwrites parameters of an existing "MaxControl" object. The '`...`' argument must be in the form `option1 = value1, option2 = value2, ...`. In case there are more than one option with similar name, only the last one is taken into account. This allows the user to override default parameters in the control list. See example in maxLik-package.

maxControl `(x = "maxim")` extracts "MaxControl" structure from an estimated model

show shows the parameter values

Details

Typically, the control options are supplied in the form of a list, in which case the corresponding default values are overwritten by the user-specified ones. However, one may also create the control structure by `maxControl(opt1=value1, opt2=value2, ...)` and supply such value directly to the optimizer. In this case the optimization routine takes all the values from the control object.

Note

Several control parameters can also be supplied directly to the optimization routines.

Author(s)

Ott Toomet

References

- Nelder, J. A. & Mead, R. A (1965) Simplex Method for Function Minimization *The Computer Journal* **7**, 308–313
- Marquardt, D. W. (1963) An Algorithm for Least-Squares Estimation of Nonlinear Parameters *Journal of the Society for Industrial and Applied Mathematics* **11**, 431–441

Examples

```
## Optimize quadratic form t(D) %*% W %*% D with p.d. weight matrix,
## s.t. constraints sum(D) = 1
quadForm <- function(D) {
   return(-t(D) %*% W %*% D)
}
eps <- 0.1
W <- diag(3) + matrix(runif(9), 3, 3)*eps
D <- rep(1/3, 3)
                        # initial values
library(maxLik)
## create control object and use it for optimization
co <- maxControl(printLevel=2, qac="marquardt", marquardt_lambda0=1)
```

```
res <- maxNR(quadForm, start=D, control=co)
print(summary(res))
## Now perform the same with no trace information
co <- maxControl(co, printLevel=0)
res <- maxNR(quadForm, start=D, control=co) # no tracing information
print(summary(res))  # should be the same as above
maxControl(res) # shows the control structure
```

`maximType` *Type of Minimization/Maximization*

Description

Returns the type of optimization as supplied 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

See Also

maxNR

Examples

```
## maximize two-dimensional exponential hat.  True maximum c(2,1):
f <- function(a) exp(-(a[1] - 2)^2 - (a[2] - 1)^2)
m <- maxNR(f, start=c(0,0))
coef(m)
maximType(m)
## Now use BFGS maximisation.
m <- maxBFGS(f, start=c(0,0))
maximType(m)
```

maxLik *Maximum likelihood estimation*

Description

This is the main interface for the **maxLik** package, and the function that performs Maximum Likelihood estimation. It is a wrapper for different optimizers returning an object of class "maxLik". Corresponding methods handle the likelihood-specific properties of the estimates, including standard errors.

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 of the corresponding individual observation.
grad	gradient of log-likelihood. Must have the parameter vector as the first argument. Must return either a single gradient vector with length equal to the number of parameters, or a matrix where each row is the gradient vector of the corresponding individual observation. 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 Hessian will be used.
start	numeric vector, initial value of parameters. If it has names, these will also be used for naming the results.
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, such as control are passed to the selected maximisation routine, i.e. maxNR, maxBFGS, maxBFGSR, maxBHHH, maxSANN, maxCG, or maxNM (depending on argument method). Arguments not used by the optimizers are forwarded to logLik, grad and hess.

Details

maxLik supports 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'. The structure is identical to that of the class "maxim" (see maxNR) but the methods differ.

Warning

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

Author(s)

Ott Toomet, Arne Henningsen

See Also

maxNR, nlm and optim for different non-linear optimisation routines, see maxBFGS for the constrained maximization examples.

Examples

```
## Estimate the parameter of exponential distribution
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, control=list(printLevel=2))
summary( a )
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, 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
##
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(100, 1, 2) # use mean=1, stdd=2
N <- length(x)
res <- maxLik(loglik, start=c(0,1)) # use 'wrong' start values
summary( res )
##
## The previous example showing parameter names and fixed values
##
resFix <- maxLik(loglik, start=c(mu=0, sigma=1), fixed="sigma")
summary(resFix)  # 'sigma' is exactly 1.000 now.
```

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,
      constraints = NULL, finalHessian = TRUE, bhhhHessian=FALSE,
      fixed = NULL, activePar = NULL, control=NULL, ... )
maxBFGSR(fn, grad = NULL, hess = NULL, start,
      constraints = NULL, finalHessian = TRUE,
      fixed = NULL, activePar = NULL, control=NULL, ... )
maxBHHH(fn, grad = NULL, hess = NULL, start,
      finalHessian = "BHHH", ... )
```

Arguments

fn	the 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 (this is is summed internally). If the BHHH method is used and argument gradient is not given, fn must return a numeric vector of observation-specific log-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 BHHH method is used, grad must return a matrix, where rows corresponds to the gradient vectors for 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 Hessian, based on gradient, is computed. Hessian is used by the Newton-Raphson method only, and eventually by the other methods if finalHessian is requested.
start	initial parameter values. If start values are named, those names are also carried over to the results.

`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 $A\theta + B > 0$. More than one row in `ineqA` and `ineqB` corresponds to more than one linear constraint, in that case all these must be zero (equality) or positive (inequality constraints). The equality-constrained problem is forwarded to `sumt`, the inequality-constrained case to `constrOptim2`.

`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 (actual, not BHHH) final Hessian does not carry any extra penalty for the NR method, but does for the other methods.

`bhhhHessian` logical. Indicating whether to use the information equality approximation (Bernd, Hall, Hall, and Hausman, 1974) for the Hessian. This effectively transforms `maxNR` into `maxBHHH` and is mainly designed for internal use.

`fixed` parameters to be treated as constants at their `start` values. If present, it is treated as an index vector of `start` parameters.

`activePar` this argument is retained for backward compatibility only; please use argument `fixed` instead.

`control` list of control parameters. The control parameters used by these optimizers are

tol 10^{-8}, stopping condition. Stop if the absolute difference between successive iterations is less than `tol`. Return `code=2`.

reltol sqrt(.Machine$double.eps), stopping condition. Relative convergence tolerance: the algorithm stops if the relative improvement between iterations is less than 'reltol'. Return code 2.

gradtol stopping condition. Stop if norm of the gradient is less than `gradtol`. Return code 1.

steptol 1e-10, stopping/error condition. If `qac == "stephalving"` and the quadratic approximation leads to a worse, instead of a better value, or to `NA`, the step length is halved and a new attempt is made. If necessary, this procedure is repeated until step < `steptol`, thereafter code 3 is returned.

lambdatol 10^{-6}, controls whether Hessian is treated as negative definite. If the largest of the eigenvalues of the Hessian is larger than `-lambdatol` (Hessian is not negative definite), a suitable diagonal matrix is subtracted from the Hessian (quadratic hill-climbing) in order to enforce negative definiteness.

qrtol 10^{-10}, QR-decomposition tolerance for the Hessian inversion.

qac "stephalving", Quadratic Approximation Correction. When the new guess is worse than the initial one, the algorithm attemts to correct it: "stephalving" decreases the step but keeps the direction, "marquardt" uses *Marquardt (1963)* method by decreasing the step length while also moving closer to the pure gradient direction. It may be faster and more robust choice in areas where quadratic approximation behaves poorly. `maxNR` and `maxBHHH` only.

marquardt_lambda0 10^{-2}, positive numeric, initial correction term for *Marquardt (1963)* correction.

marquardt_lambdaStep 2, how much the *Marquardt (1963)* correction term is decreased/increased at each successful/unsuccesful step. `maxNR` and `maxBHHH` only.

marquardt_maxLambda 10^{12}, maximum allowed *Marquardt (1963)* correction term. If exceeded, the algorithm exits with return code 3. `maxNR` and `maxBHHH` only.

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

printLevel this argument determines the level of printing which is done during the optimization process. The default value 0 means that no printing occurs, 1 prints the initial and final details, 2 prints all the main tracing information for every iteration. Higher values will result in even more output.

`...` further arguments to `fn`, `grad` and `hess`. Further arguments to `maxBHHH` are also passed to `maxNR`. To maintain compatibility with the earlier versions, ... also passes a number of control options (`tol`, `reltol`, `gradtol`, `steptol`, `lambdatol`, `qrtol`, `iterlim`) to the optimizers.

Details

The idea of the Newton method is to approximate the function at a given location by a multidimensional quadratic function, and use the estimated maximum as the start 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 (information equality) approximation is only valid for log-likelihood functions. It requires the score (gradient) values by individual observations and hence those must be returned by individual observations by `grad` or `fn`. 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,

$$\mathsf{H}^{BHHH} = -\frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial \ell(\boldsymbol{\vartheta})}{\boldsymbol{\vartheta}}\frac{\partial \ell(\boldsymbol{\vartheta})}{\boldsymbol{\vartheta}'}\right]$$

The functions `maxNR`, `maxBFGSR`, and `maxBHHH` can work with constant parameters, useful if a parameter value converges to the boundary of support, or for testing. One way is to put `fixed` to non-NULL, specifying which parameters should be treated as constants. The parameters can also be fixed in runtime (only for `maxNR` and `maxBHHH`) by signaling it with the `fn` return value. See Henningsen & Toomet (2011) for details.

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 the `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 is correct. The users are encouraged to use `compareDerivatives` function, designed for this purpose. If analytic gradient/Hessian are wrong, the algorithm may not converge, or may converge to a wrong point.

As the BHHH method 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, 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**, 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**, 76–90.

Fletcher, R. (1970): A New Approach to Variable Metric Algorithms, *Computer Journal* **13**, 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**, 23–26.

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

Henningsen, A. and Toomet, O. (2011): maxLik: A package for maximum likelihood estimation in R *Computational Statistics* **26**, 443–458

Marquardt, D.W., (1963) An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *Journal of the Society for Industrial & Applied Mathematics* **11**, 2, 431–441

Shanno, D.F. (1970): Conditioning of Quasi-Newton Methods for Function Minimization, *Mathematics of Computation* **24**, 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 on optim), also supporting inequality constraints; nlm for Newton-Raphson optimization; and optim for different gradient-based optimization methods.

Examples

```
## estimate the exponential distribution parameter by ML
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, control=list(printLevel=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)

## 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 analytic gradient
a <- maxBHHH(loglikInd, gradlikInd, start=1)
summary(a)

##
## Example with a 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(100, 1, 2) # use mean=1, stdd=2
N <- length(x)
res <- maxNR(loglik, start=c(0,1)) # use 'wrong' start values
summary(res)
##
## The previous example with named parameters and fixed values
##
resFix <- maxNR(loglik, start=c(mu=0, sigma=1), fixed="sigma")
summary(resFix)  # 'sigma' is exactly 1.000 now.
###
### Constrained optimization
###
## We maximize exp(-x^2 - y^2) where x+y = 1
hatf <- function(theta) {
  x <- theta[1]
  y <- theta[2]
  exp(-(x^2 + y^2))
  ## Note: you may prefer exp(- theta %*% theta) instead
}
## use constraints: x + y = 1
A <- matrix(c(1, 1), 1, 2)
B <- -1
res <- maxNR(hatf, start=c(0,0), constraints=list(eqA=A, eqB=B),
             control=list(printLevel=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, created by `maxLik`, `maxNR` or another optimizer.
`...`	further arguments for methods

Details

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

Value

numeric, number of iterations. Note that 'iteration' may mean different things for different optimizers.

Author(s)

Ott Toomet

See Also

`maxLik`, `maxNR`

Examples

```
## Estimate the exponential distribution parameter:
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 "maxLik" objects can return the number of observations only if log-likelihood function (or the gradient) returns values by individual 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 jointly restricted (e.g. sum of two probabilities equals unity). In this case the total number of parameters may depend on the normalization.

Value

Number of parameters in the model

Author(s)

Ott Toomet

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, including of 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 parameter values
`eps`	numeric, the step for numeric differentiation
`fixed`	logical index vector, fixed parameters. Derivative is calculated only with respect to the parameters for which `fixed == FALSE`, `NA` is returned for the fixed parameters. 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 $N_{val} \times 1$ vector and the argument is $N_{par} \times 1$ vector, the resulting gradient is a $N_{val} \times N_{par}$ matrix.

`numericHessian` checks whether a gradient function is present. If yes, it calculates the gradient of the gradient, if not, it calculates the full numeric Hessian (`numericNHessian`).

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 optimization routines. Although quite precise in simple cases, they may work very poorly in more complicated conditions.

Author(s)

Ott Toomet

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 are often quite precise
compareDerivatives(f0, gradf0, t0=1:2)
# The difference is around 1e-10
```

---

returnCode *Success or failure of the optimization*

Description

These function extract success or failure information from optimization objects. The `returnCode` gives a numeric code, and `returnMessage` a brief description about the success or failure of the optimization, and point to the problems occured (see documentation for the corresponding functions).

Usage

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

Arguments

x	object, usually an optimization result
...	further arguments for other methods

Details

returnMessage and returnCode are a generic functions, with methods for various optimisation algorithms. The message should either describe the convergence (stopping condition), or the problem.

Value

Integer for returnCode, character for returnMessage. Different optimization routines may define it in a different way.

Author(s)

Ott Toomet

See Also

maxNR, maxBFGS

Examples

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

---

summary.maxim *Summary method for maximization*

Description

Summarizes the maximization results

Usage

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

Arguments

object	optimization 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 maximization.
iterations	number of iterations.
code	exit code (see returnCode.)
message	a brief message, explaining the outcome (see returnMessage).
unsucc.step	description of last unsuccessful step, only if requested and code == 3
maximum	function value at maximum
estimate	matrix with following columns: **results** coefficient estimates at maximum **gradient** estimated gradient at maximum
constraints	information about the constrained optimization. NULL if unconstrained maximization.
hessian	estimated hessian at maximum (if requested)

Author(s)

Ott Toomet

See Also

maxNR, returnCode, returnMessage

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`	The standard errors are only calculated if the ratio of the smallest and largest eigenvalue of the Hessian matrix is less than "eigentol". Otherwise the Hessian is treated as singular.
`...`	currently not used.

Value

An object of class 'summary.maxLik' with following components:

type type of maximization.

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.

Author(s)

Ott Toomet, Arne Henningsen

See Also

maxLik

Examples

```
## ML estimation of exponential distribution:
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, control=list(printLevel=2))
summary(a)
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, start=1, control=list(printLevel=2))
summary(a)
```

---

sumt *Equality-constrained optimization*

Description

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

This implementation is primarily 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,
printLevel=print.level, print.level = 0, SUMTMaxIter = 100, ...)
```

Arguments

fn	function of a (single) vector parameter. The function may have more arguments (passed by ...), but those are not treated as the parameter.
grad	gradient function of fn. NULL if missing
hess	function, Hessian of the fn. NULL if missing
start	numeric, initial value of the parameter
maxRoutine	maximization algorithm, such as maxNR

`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 estimates at 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 that the constraints are satisfied. If the penalty function is too "weak", SUMT may repeatedly find the same optimum. In that case a warning is issued. The user may set SUMTTol to a lower value, e.g. to zero.

`SUMTPenaltyTol` stopping condition. If the 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 authomatically selected value may not lead to convergence.

`printLevel` Integer, debugging information. Larger number prints more details.

`print.level` same as 'printLevel', for backward compatibility

`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, it uses a non-negative penalty function P, such that $P(x)$ is zero iff x satisfies the constraints. One iteratively minimizes $f(x) + \varrho_k P(x)$, where the ϱ values are increased according to the rule $\varrho_{k+1} = q\varrho_k$ for some constant $q > 1$, until convergence is achieved in the sense that the barrier value $P(x)'P(x)$ is close to zero. Note that there is no guarantee that the global constrained optimum is found. Standard practice recommends to use the best solution found in "sufficiently many" replications.

Any of the maximization algorithms in the **maxLik**, such as `maxNR`, can be used for the unconstrained step.

Analytic gradient and hessian are used if provided.

Value

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

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

type type of constrained optimization

barrier.value value of the penalty function at maximum

code code for the stopping condition

message a short message, describing the stopping condition

outer.iterations number of iterations in the SUMT step

Note

In case of equality constraints, it may be more efficient to enclose the function in a wrapper function. The wrapper calculates full set of parameters based on a smaller set of parameters, and the constraints.

Author(s)

Ott Toomet, Arne Henningsen

See Also

`sumt` in package **clue**.

Examples

```
## We maximize exp(-x^2 - y^2) where x+y = 1
hatf <- function(theta) {
   x <- theta[1]
   y <- theta[2]
   exp(-(x^2 + y^2))
   ## Note: you may prefer exp(- theta %*% theta) instead
}
## use constraints: x + y = 1
A <- matrix(c(1, 1), 1, 2)
B <- -1
res <- sumt(hatf, start=c(0,0), maxRoutine=maxNR,
            constraints=list(eqA=A, eqB=B))
print(summary(res))
```

`vcov.maxLik` *Variance Covariance Matrix of maxLik objects*

Description

Extract variance-covariance matrices from `maxLik` objects.

Usage

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

Arguments

object	a 'maxLik' object.
eigentol	eigenvalue tolerance, controlling when the Hessian matrix is treated as numerically singular.
...	further arguments (currently ignored).

Details

The standard errors are only calculated if the ratio of the smallest and largest eigenvalue of the Hessian matrix is less than "eigentol". Otherwise the Hessian is treated as singular.

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

See Also

vcov, maxLik.

Examples

```
## ML estimation of exponential random variables
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, control=list(printLevel=2))
vcov(a)
## Estimate with analytic gradient and hessian
a <- maxLik(loglik, gradlik, hesslik, start=1)
vcov(a)
```

Index