

# Package ‘maxLik’

October 16, 2009

**Version** 0.6-0

**Date** 2009/10/16

**Title** Maximum Likelihood Estimation

**Author** Ott Toomet <otootmet@ut.ee>, Arne Henningsen <arne.henningsen@googlemail.com>, with contributions from Spencer Graves

**Maintainer** Arne Henningsen <arne.henningsen@googlemail.com>

**Depends** R (>= 2.4.0)

**Suggests**

**Description** Tools for Maximum Likelihood Estimation

**License** GPL (>= 2)

**URL** <http://CRAN.R-project.org>, <http://www.maxLik.org>

**Repository** CRAN

**Date/Publication** 2009-10-16 11:11:40

## R topics documented:

activePar . . . . .	2
AIC.maxLik . . . . .	3
compareDerivatives . . . . .	3
condiNumber . . . . .	5
fnSubset . . . . .	7
hessian . . . . .	8
logLik.maxLik . . . . .	10
maxBFGS . . . . .	11
maxBHHH . . . . .	13
maximType . . . . .	15
maxLik . . . . .	16
maxNR . . . . .	17

nIter . . . . .	21
nObs . . . . .	22
nParam . . . . .	23
numericGradient . . . . .	24
returnCode . . . . .	26
returnMessage . . . . .	27
stdEr . . . . .	28
summary.maxim . . . . .	29
summary.maxLik . . . . .	30
sumt . . . . .	32
vcov.maxLik . . . . .	33

<b>Index</b>	<b>35</b>
--------------	-----------

---

activePar	<i>free parameters under maximisation</i>
-----------	---

---

## 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 <code>maxNR</code> and its derivations support <code>activePar</code>
...	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

---

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	function returning the analytic gradient. Must use the same set of parameters as f. If f is a vector-valued function, grad must return 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.

<code>hess</code>	function returning the analytic hessian. If present, hessian matrices are compared too. Only appropriate for scalar-valued functions.
<code>t0</code>	parameter vector indicating the point at which the derivatives are compared. The derivative is taken with respect to this vector.
<code>eps</code>	numeric. Step size for numeric differentiation. Central derivative is used.
<code>print</code>	logical: TRUE to print a summary, FALSE to return the comparison only (invisibly).
<code>...</code>	further arguments to <code>f</code> , <code>grad</code> and <code>hess</code> .

### Details

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

$$\text{rel.diff} = (\text{analytic} - \text{numeric}) / (0.5 * (\text{analytic} + \text{numeric}))$$

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:

<code>t0</code>	the input argument <code>t0</code>
<code>f.t0</code>	<code>f(t0)</code>
<code>compareGrad</code>	a list with components <code>analytic = grad(t0)</code> , <code>numeric = numericGradient(f, t0)</code> , and their <code>rel.diff</code> .
<code>maxRelDiffGrad</code>	<code>max(abs(rel.diff))</code>
<code>compareHessian</code>	a list with components <code>analytic = hess(t0)</code> , <code>numeric = numericGradient(grad, t0)</code> , and their <code>rel.diff</code> .
<code>maxRelDiffHess</code>	<code>max(abs(rel.diff))</code> for the Hessian

### Author(s)

Ott Toomet ([otootmet@ut.ee](mailto:otootmet@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)
```

---

code>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

<code>x</code>	numeric matrix, condition numbers of which are to be printed
<code>exact</code>	logical, should condition numbers be exact or approximations (see <code>link{kappa}</code> )
<code>norm</code>	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 understand 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
```

---

`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

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

### 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, xFixed)`, 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, [〈otootmet@econ.au.dk〉](mailto:otootmet@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

<code>object</code>	object of class <code>maxLik</code> or <code>summary.maxLik</code> , usually a model estimated with Maximum Likelihood
<code>...</code>	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, SANN and Nelder-Mead Maximization***Description**

These functions are wrappers for `optim` where the arguments are compatible with `maxNR`

**Usage**

```
maxBFGS(fn, grad = NULL, hess=NULL, start, print.level = 0,
iterlim = 200,
constraints,
  tol = 1e-08, reltol=tol, ... )
maxSANN(fn, grad = NULL, hess = NULL, start, print.level = 0, iterlim =
  10000,
  constraints,
  tol = 1e-08, reltol=tol, temp = 10, tmax = 10, parscale = rep(1, length = length(
maxNM(fn, grad = NULL, hess = NULL, start, print.level = 0, iterlim =
  500,
  constraints,
  tol = 1e-08, reltol=tol, parscale = rep(1, length = length(start)), alpha = 1, beta
```

**Arguments**

<code>fn</code>	function to be maximised. Must have the parameter vector as the first argument. In order to use numeric gradient and BHHH method, <code>fn</code> must return vector of observation-specific likelihood values. Those are summed by <code>maxNR</code> if necessary. If the parameters are out of range, <code>fn</code> should return NA. See details for constant parameters.
<code>grad</code>	gradient of the function. Must have the parameter vector as the first argument. If NULL, numeric gradient is used (only <code>maxBFGS</code> uses gradient). Gradient may return a matrix, where columns correspond to the parameters and rows to the observations (useful for <code>maxBHHH</code> ). The columns are summed internally.
<code>hess</code>	Hessian of the function. Not used by any of these methods, for compatibility with <code>maxNR</code> .
<code>start</code>	initial values for the parameters.
<code>print.level</code>	a larger number prints more working information.
<code>iterlim</code>	maximum number of iterations.
<code>constraints</code>	either NULL for unconstrained optimization or a list with two components. The components may be either <code>eqA</code> and <code>eqB</code> for equality-constrained optimization $A\theta + B = 0$ ; or <code>ineqA</code> and <code>ineqB</code> for inequality constraints $A\theta + B \geq 0$ . The equality-constrained problem is forwarded to <code>sumt</code> , the inequality-constrained case to <code>constrOptim</code> .
<code>tol, reltol</code>	the relative convergence tolerance (see <code>optim</code> ). <code>tol</code> is for compatibility with <code>maxNR</code> .

<code>temp</code>	controls the "SANN" method. It is the starting temperature for the cooling schedule. Defaults to '10'.
<code>tmax</code>	is the number of function evaluations at each temperature for the "SANN" method. Defaults to '10'. (see <a href="#">optim</a> )
<code>parscale</code>	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 <a href="#">optim</a> )
<code>alpha, beta, gamma</code>	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 <a href="#">optim</a> )
<code>...</code>	further arguments for <code>fn</code> and <code>grad</code> .

### Value

Object of class "maxim":

<code>maximum</code>	value of <code>fn</code> at maximum.
<code>estimate</code>	best set of parameters found.
<code>gradient</code>	gradient at parameter value <code>estimate</code> .
<code>hessian</code>	value of Hessian at optimum.
<code>code</code>	integer. Success code, 0 is success (see <a href="#">optim</a> ).
<code>message</code>	character string giving any additional information returned by the optimizer, or NULL.
<code>activePar</code>	logical vector indicating which parameters are treated as free (currently all parameters).
<code>iterations</code>	two-element integer vector giving the number of calls to <code>fn</code> and <code>gr</code> , respectively. This excludes those calls needed to compute the Hessian, if requested, and any calls to <code>fn</code> to compute a finite-difference approximation to the gradient.
<code>type</code>	character string "BFGS maximisation".
<code>constraints</code>	A list, describing the constrained optimization (NULL if unconstrained). Includes the following components: <ul style="list-style-type: none"> <li><code>type</code> type of constrained optimization</li> <li><code>outer.iterations</code> number of iterations in the constraints step</li> <li><code>barrier.value</code> value of the barrier function</li> </ul>

### Author(s)

Ott Toomet (otoomet@ut.ee)

### See Also

[optim](#), [nlm](#), [maxNR](#), [maxBHHH](#).

**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))

```

maxBHHH

*BHHH Maximization***Description**

This function is essentially `maxNR` where the hessian is approximated by the outer product of the gradient vector. It is suitable for likelihood maximisation only.

**Usage**

```
maxBHHH(fn, grad = NULL, hess=NULL, start, print.level = 0, iterlim = 100,
...)
```

**Arguments**

<code>fn</code>	log-likelihood function to be maximised, must have the parameter vector as the first argument and must return vector of observation-specific likelihood values if numeric gradient is requested (those are summed for the single likelihood value). If the parameters are out of range, <code>fn</code> should return NA.
<code>grad</code>	gradient of the log-likelihood function. Must have the parameter vector as the first argument. If <code>NULL</code> , numeric gradient is used. It must return a matrix, where rows corresponds to the gradient vectors of individual observations and the columns to the individual parameters. This matrix is summed over observations in order to get a single gradient vector. It's matrix product is used for approximating the Hessian.

<code>hess</code>	Hessian of the function. Not used by <code>maxBHHH</code> , for compatibility with the other maximisation routines.
<code>start</code>	initial values for the parameters.
<code>print.level</code>	a larger number prints more working information.
<code>iterlim</code>	maximum number of iterations.
<code>...</code>	further arguments for <code>maxNR</code> , <code>fn</code> and <code>grad</code> .

### Details

`maxBHHH` uses information equality in order to approximate the Hessian of the log-likelihood function. Hence we have to calculate log-likelihood and gradient by individual observations. Hessian is approximated as sum of outer products of the gradients of individual observations, or, in the matrix form,  $-\tau(\text{gradient}) \%*\% \text{gradient}$ .

### Value

Object of class "maxim":

`type` Character string: "BHHH maximisation"

Plus the other components inherited from `maxNR`.

### Warning

As `maxBHHH` uses likelihood-specific information equation, it is only suitable for maximising log-likelihood!

### Author(s)

Ott Toomet (otoomet@ut.ee)

### See Also

`maxNR`, `maxBFGS`, `optim`, `nlm`.

### Examples

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

---

`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 <code>NULL</code> , 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 <code>NULL</code> , numeric gradient will be used.
start	numeric vector, initial value of parameters.
method	maximisation method, currently either "Newton-Raphson", "BFGS", "BHHH", "SANN" or "NM" (for Nelder-Mead). Lower-case letters and shortcuts (as 'nr' for Newton-Raphson) allowed. If missing, a suitable method is selected automatically.
constraints	either <code>NULL</code> for unconstrained maximization or a list, specifying the constraints. See <a href="#">maxBFGS</a> .
...	further arguments for the maximisation routine.

**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)

**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 )
```

**Description**

Unconstrained maximization based on Newton-Raphson method.

**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,
      activePar=rep(TRUE, length(start)), ...)
```

**Arguments**

fn	function to be maximized. It must return either a single number or a numeric vector, which is summed (this is useful for BHHH method). If the parameters are out of range, fn should return NA. See details for constant parameters.
grad	gradient of the function. If NULL, numeric gradient is used. It must return a gradient vector, or matrix where <i>columns</i> correspond to individual parameters. The column sums are treated as gradient components (this is useful for BHHH method).
hess	Hessian matrix of the function. If missing, numeric Hessian, based on gradient, is used.
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 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 negative 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 <a href="#">sumt</a> .

`activePar` index vector, which parameters are treated as free. Other parameters are treated as fixed constants.

`...` further arguments to `fn`, `grad` and `hess`.

### Details

The algorithm 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 `activePar` to non-NULL, specifying which parameters we want to change. However, 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:

`constPar` index vector. Which parameters are redefined to constant

`newVal` a list with following components:

`index` which parameters will have a new value  
`val` the 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` last gradient value which was calculated. Should be close to 0 if normal convergence.

`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).
- 4 iteration limit exceeded.
- 5 Infinite value.
- 6 Infinite gradient.
- 7 Infinite Hessian.
- 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  
`activePar` logical vector, which parameters are not 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

### Author(s)

Ott Toomet (otoomet@ut.ee)

### References

Greene, W., 2002, *Advanced Econometrics* Goldfeld, S. M. and Quandt, R. E., 1972, *Nonlinear Methods in Econometrics*. Amsterdam: North-Holland

### See Also

[nlm](#) for Newton-Raphson optimization, [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 numeric 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)
##
## 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

<code>x</code>	a statistical model, or a result of maximisation, such as created by <a href="#">maxLik</a> or <a href="#">maxNR</a>
<code>...</code>	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

*Return number of observations for statistical models*


---

**Description**

Returns number of observations for statistical models. The default method assumes presence of a component `param$nObs` in `x`.

**Usage**

```
nObs(x, ...)
## Default S3 method:
nObs(x, ...)
## S3 method for class 'lm':
nObs(x, ...)
```

**Arguments**

`x` a statistical model, such as created by [lm](#)  
`...` further arguments for methods

**Details**

This is a generic function. The default method returns the component `x$param$nObs`. The `lm`-method is based on qr-decomposition, in the same way as the does [summary.lm](#).

**Value**

numeric, number of observations

**Author(s)**

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

**See Also**

[nParam](#)

**Examples**

```
# Construct a simple OLS regression:
x1 <- runif(100)
x2 <- runif(100)
y <- 3 + 4*x1 + 5*x2 + rnorm(100)
m <- lm(y~x1+x2) # estimate it
nObs(m)
```

---

nParam

*Number of model parameters*

---

**Description**

This function returns the number of model parameters. The default method returns the component `x$param$nParam`.

**Usage**

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

**Arguments**

<code>x</code>	a statistical model
<code>free</code>	logical, whether to report only the free parameters or the total number of parameters (default)
<code>...</code>	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**

```
# Construct a simple OLS regression:
x1 <- runif(100)
x2 <- runif(100)
y <- 3 + 4*x1 + 5*x2 + rnorm(100)
m <- lm(y~x1+x2) # estimate it
summary(m)
nParam(m) # you get 3
```

---

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

**Arguments**

<code>f</code>	function to be differentiated. The first argument must be the parameter vector with respect to which it is differentiated.
<code>grad</code>	function, gradient of <code>f</code>
<code>t0</code>	vector, the value of parameters
<code>eps</code>	numeric, the step for numeric differentiation
<code>activePar</code>	logical vector, length of which equal the length of the parameter. Derivative is calculated only along the parameters for which it is <code>TRUE</code> , <code>NA</code> returned for the others. If missing, all parameters are treated as active.
<code>...</code>	furter arguments for <code>f</code>

## 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 number 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 cases.

## Author(s)

Ott Toomet (siim@obs.ee)

## See Also

[compareDerivatives](#), [deriv](#)

## Examples

```
# A simple example with Gaussian bell
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 differenc is around 1e-10
```

---

returnCode	<i>Return code for optimisation and other objects</i>
------------	---

---

### Description

This function gives the return code of various optimisation objects. The return code gives a brief information about the success or problems, occurred 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, [otootomet@ut.ee](mailto:otootomet@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 originate 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)
```

---

`stdEr`*Standard deviations*

---

**Description**

Extract standard deviations from estimated models.

**Usage**

```
stdEr(x, ...)
## Default S3 method:
stdEr(x, ...)
## S3 method for class 'lm':
stdEr(x, ...)
```

**Arguments**

`x` a statistical model, such as created by `lm`  
`...` further arguments for methods

**Details**

`stdEr` is a generic function with methods for objects of "lm" class. The default method returns the square root of the diagonal of the variance-covariance matrix.

**Value**

numeric, the estimated standard errors of the coefficients.

**Author(s)**

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
## Estimate with numeric gradient and hessian
a <- maxLik(loglik, start=1, print.level=2)
stdEr(a)
```

---

summary.maxim	<i>Summary method for maximisation/minimisation</i>
---------------	---

---

**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 <code>maxim</code> . See <a href="#">maxNR</a> .
hessian	logical, whether to display Hessian matrix.
unsucc.step	logical, whether to describe last unsuccessful step if <code>code == 3</code>
...	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 <a href="#">maxNR</a> .)
message	a brief message, explaining code.
unsucc.step	description of last unsuccessful step, only if requested and <code>code == 3</code>
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. Passed directly further from <code>maxim-object</code> . 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 $\min(\text{absEig}) > (\text{eigentol} * \max(\text{absEig}))$ .
...	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.  
 activePar      logical vector, which parameters are treated as free.  
 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)

### 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),
     SUMTQ = 10,
     SUMTRho0 = NULL,
     print.level = 0, SUMTMaxIter = 100, ...)
```

### Arguments

<code>fn</code>	function of a (single) vector parameter. The function may have more arguments, but those are not treated as parameter
<code>grad</code>	gradient function of <code>fn</code> . NULL if missing
<code>hess</code>	hessian matrix of the <code>fn</code> . NULL if missing
<code>start</code>	initial value of the parameter.
<code>maxRoutine</code>	maximization algorithm
<code>constraints</code>	list, information for constrained maximization. Currently two components are supported: <code>eqA</code> and <code>eqB</code> for linear equality constraints: $A\beta + B = 0$ . The user must ensure that the matrices A and B are conformable.
<code>SUMTTol</code>	stopping condition. If the components of the parameter between successive iterations differ less than <code>SUMTTol</code> , the algorithm stops
<code>SUMTQ</code>	a double greater than one controlling the growth of the <code>rho</code> as described in Details. Defaults to 10.
<code>SUMTRho0</code>	Initial value for <code>rho</code> . If not specified, a (possibly) suitable value is selected. See Details.
<code>print.level</code>	Integer, debugging information. Larger number print more details.
<code>SUMTMaxIter</code>	Maximum SUMT iterations
<code>...</code>	Other arguments to <code>maxRoutine</code> and <code>fn</code> .

**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  $\varrho$  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 Euclidean distance between successive solutions  $x_k$  and  $x_{k+1}$  is small enough. Note that the "solution"  $x$  obtained does not necessarily satisfy the constraints, i.e., has zero  $P(x)$ . Note also 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:

`type` type of constrained optimization

`outer.iterations` number of iterations in the SUMT step

`barrier.value` value of the penalty function at maximum

**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)

**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

- \*Topic **debugging**
  - condiNumber, 5
- \*Topic **math**
  - compareDerivatives, 3
  - condiNumber, 5
  - numericGradient, 23
- \*Topic **methods**
  - activePar, 1
  - AIC.maxLik, 3
  - hessian, 8
  - logLik.maxLik, 9
  - maximType, 14
  - nIter, 20
  - nObs, 21
  - nParam, 22
  - returnCode, 25
  - returnMessage, 26
  - stdEr, 27
  - summary.maxim, 28
  - vcov.maxLik, 32
- \*Topic **models**
  - summary.maxLik, 29
- \*Topic **optimize**
  - activePar, 1
  - fnSubset, 6
  - hessian, 8
  - maxBFGS, 10
  - maxBHHH, 12
  - maximType, 14
  - maxLik, 15
  - maxNR, 17
  - sumt, 31
- \*Topic **print**
  - summary.maxim, 28
- \*Topic **utilities**
  - compareDerivatives, 3
  - condiNumber, 5
  - numericGradient, 23
  - returnCode, 25
  - returnMessage, 26
- activePar, 1, 8
- AIC.maxLik, 3
- coef.maxLik (*maxLik*), 15
- coef.summary.maxLik  
(*summary.maxLik*), 29
- compareDerivatives, 3, 24
- condiNumber, 5
- constrOptim, 11
- deriv, 4, 24
- dlnMLE, 7
- fnSubset, 6
- hessian, 8
- kappa, 6
- lm, 22, 27
- logLik.maxLik, 9
- logLik.summary.maxLik  
(*logLik.maxLik*), 9
- maxBFGS, 10, 13, 15
- maxBHHH, 12, 12
- maximType, 14
- maxLik, 7–9, 15, 21, 27, 30, 32, 33
- maxNM (*maxBFGS*), 10
- maxNR, 2, 7, 10–14, 16, 17, 21, 25, 26, 28, 29,  
31, 32
- maxSANN (*maxBFGS*), 10
- nIter, 20
- nln, 12, 13, 16, 19
- nObs, 2, 21, 23
- nParam, 22, 22
- numericGradient, 4, 23
- numericHessian (*numericGradient*),  
23

numericNHessian  
    (*numericGradient*), 23

optim, 7, 10–13, 16, 19

print.maxLik (*maxLik*), 15

returnCode, 25, 26

returnMessage, 25, 26

std.maxlik (*AIC.maxLik*), 3

stdEr, 27

summary.lm, 22

summary.maxim, 28

summary.maxLik, 29

sumt, 11, 18, 31, 32

vcov, 27, 33

vcov.maxLik, 32