# Package 'maxLik'

October 1, 2026

**Version** 1.6-10

**Date** 2026-09-29

**Title** Maximum Likelihood Estimation and Related Tools

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

**Imports** sandwich, generics

**URL** https://github.com/arne-henningsen/maxLik

**BugReports** https://github.com/arne-henningsen/maxLik/issues

**Suggests** MASS, clue, dlm, plot3D, tibble, tinytest, knitr

**VignetteBuilder** knitr

**Description** Functions for Maximum Likelihood (ML) estimation, 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 Maximum Likelihood viewpoint. It also includes a number of convenience tools for testing and developing your own models.

**License** GPL (>= 2)

**ByteCompile** yes

**NeedsCompilation** no

**Author** Ott Toomet [aut, cre],
Arne Henningsen [aut],
Spencer Graves [ctb],
Yves Croissant [ctb],
David Hugh-Jones [ctb],
Luca Scrucca [ctb]

**Maintainer** Ott Toomet <otoomet@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-10-01 07:20:02 UTC

# Contents


maxLik-package . . . 2
activePar . . . 4
AIC.maxLik . . . 5
bread.maxLik . . . 6
compareDerivatives . . . 7
condiNumber . . . 10
confint.maxLik . . . 11
fnSubset . . . 12
gradient . . . 14
hessian . . . 15
logLik.maxLik . . . 16
maxBFGS . . . 17
MaxControl-class . . . 21
maximType . . . 24
maxLik . . . 25
maxNR . . . 27
maxSGA . . . 33
maxValue . . . 38
nIter . . . 39
nObs.maxLik . . . 40
nParam.maxim . . . 41
numericGradient . . . 42
objectiveFn . . . 44
returnCode . . . 45
storedValues . . . 46
summary.maxim . . . 47
summary.maxLik . . . 49
sumt . . . 50
tidy.maxLik . . . 52
vcov.maxLik . . . 54

**Index** **56**


---

`maxLik-package` *Maximum Likelihood Estimation*

---

## Description

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

## Details

**maxLik** package is a set of convenience tools and wrappers focusing on Maximum Likelihood (ML) analysis, but it also contains tools for other optimization tasks. The package includes a) wrappers for several existing optimizers (implemented by `optim`); b) original optimizers, including Newton-Raphson and Stochastic Gradient Ascent; 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 test user-provided analytic derivatives, and constrained optimization.

A good starting point to learn about the usage of **maxLik** are the included vignettes "Introduction: what is maximum likelihood", "Maximum likelihood estimation with maxLik" and "Stochastic Gradient Ascent in maxLik". Another good source is Henningsen & Toomet (2011), an introductory paper to the package. Use `vignette(package="maxLik")` to see the available vignettes, and `vignette("using-maxlik")` to read the usage vignette.

From the user's perspective, the central function in the package is `maxLik`. In its simplest form it takes two arguments: the log-likelihood function, and a vector of initial parameter values (see the example below). 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.

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

In the interest of providing a unified user interface, all the optimizers are implemented as maximizers in this package. This includes the `optim`-based methods, such as `maxBFGS` and `maxSGA`, the maximizer version of popular Stochastic Gradient Descent.

## Author(s)

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

Maintainer: Ott Toomet <otoomet@gmail.com>

## References

Henningsen A, Toomet O (2011). "maxLik: A package for maximum likelihood estimation in R." Computational Statistics, 26(3), 443-458. doi: doi:10.1007/s0018001002171.

## Examples

```
### estimate mean and variance of normal random vector

## create random numbers where mu=1, sd=2
set.seed(123)
x <- rnorm(50, 1, 2 )

## log likelihood function.
## Note: 'param' is a 2-vector c(mu, sd)
llf <- function(param) {
   mu <- param[1]
```

```
   sd <- param[2]
   llValue <- dnorm(x, mean=mu, sd=sd, log=TRUE)
   sum(llValue)
}

## Estimate it with mu=0, sd=1 as start values
ml <- maxLik(llf, start = c(mu=0, sigma=1) )
print(summary(ml))
## Estimates close to c(1,2) :-)
```

---

activePar *free parameters under maximization*

---

## Description

Return a logical vector, indicating which parameters were free under maximization, 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 maximization routine, such as maxNR or maxLik, or derived from a maximization object. |
| ... | further arguments for methods |

## Details

Several optimization 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 optimization-related information.

## Value

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

## Author(s)

Ott Toomet

## See Also

maxNR, nObs

## Examples

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

## Arguments

| | |
|---|---|
| `object` | a 'maxLik' object (`coef` can also handle 'maxim' objects) |
| `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 ratio 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).

## Value

- for `AIC`, a number with attribute 'df' for degrees of freedom.
- for `coef` a numeric vector of estimated coefficients. It may be named if the names are supplied to the model.
- for `stdEr` a numeric vector of standard errors of the estimates. In case standard errors cannot be computed it is a vector of `NA`-s of correct length.

## 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,
                   printLevel=1, print=printLevel > 0,
                   max.rows=getOption("max.rows", 20),
                   max.cols=getOption("max.cols", 7),
                   ...)
```

## 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. |
| printLevel | numeric: a positive number prints summary of the comparison. 0 does not do any printing, only returns the comparison results (invisibly). |
| print | deprecated (for backward compatibility only). |
| max.rows | maximum number of matrix rows to be printed. |
| max.cols | maximum number of columns to be printed. |
| ... | 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 $= 0$ and numeric $= 0$, then rel.diff is also set to 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
```

---

## confint.maxLik *confint method for maxLik objects*

---

### Description

Wald confidence intervals for Maximum Likelihood Estimates

### Usage

```
## S3 method for class 'maxLik'
confint(object, parm, level=0.95,  ...)
```

### Arguments

| | |
|---|---|
| object | object of class "maxLik" returned by maxLik function |
| parm | the name of parameters to compute the confidence intervals. If omitted, confidence intervals for all parameters are computed. |
| level | the level of confidence interval |
| ... | additional arguments to be passed to the other methods |

## Value

A matrix of lower and upper confidence interval limits (in the first and second column respectively). The matrix rows are labeled by the parameter names (if any) and columns by the corresponding distribution quantiles.

## Author(s)

Luca Scrucca

## See Also

`confint` for the generic `confint` function, `stdEr` for computing standard errors and `summary` for summary output that includes statistical significance information.

## Examples

```
## compute MLE parameters of normal random sample
x <- rnorm(100)
loglik <- function(theta) {
   dnorm(x, mean=theta[1], sd=theta[2], log=TRUE)
}
m <- maxLik(loglik, start=c(mu=0, sd=1))
summary(m)
confint(m)
confint(m, "mu", level=0.1)
```

---

# 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.)
```

---

gradient *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, ...)
## S3 method for class 'maxim'
gradient(x, ...)
```

## Arguments

| | |
|---|---|
| x | an object inheriting from class maxim (for gradient) or maxLik. (for estfun.) |
| ... | further arguments (currently ignored). |

## Value

| | |
|---|---|
| gradient | vector, objective function gradient at estimated maximum (or the last calculated value if the estimation did not converge.) |
| estfun | matrix, observation-wise 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 in order to use estfun.

estfun only works 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, Ott Toomet

## See Also

hessian, estfun, maxLik.

## Examples

```
## ML estimation of exponential duration model:
t <- rexp(10, 2)
loglik <- function(theta) log(theta) - theta*t

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

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

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

---

`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`).

If for some reason Hessian is not available, then it returns `NULL`.

## 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. It has attribute "df", number of free parameters.

## 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 stay compatible with BHHH method (this is only applicable for maximizing log-likelihood), fn may return a vector of observation-specific function 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, a larger number prints more working information. Default 0, no information.
Because these are wrappers to optim, the working information is printed by optim. In particular, it prints **negative values of the objective function**!

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

**nm_beta** 0.5, Nelder-Mead contraction 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 (do 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". Data can be extracted through the following methods:

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

`coef` estimated parameter value.

| | |
|---|---|
| gradient | vector, last calculated gradient value. Should be close to 0 in case of normal convergence. |
| estfun | 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). |
| returnCode | integer. Success code, 0 is success (see optim). |
| returnMessage | a short message, describing the return code. |
| activePar | logical vector, which parameters are optimized over. Contains only TRUE-s if no parameters are fixed. |
| nIter | number of 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. |
| maximType | character string, type of maximization. |
| maxControl | the optimization control parameters in the form of a MaxControl object. |

The following components can only be extracted directly (with $):

| | |
|---|---|
| constraints | A list, describing the constrained optimization (NULL if unconstrained). Includes the following components:<br>**type** type of constrained optimization<br>**outer.iterations** number of iterations in the constraints step<br>**barrier.value** value of the barrier function |

## 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)
## sum of individual likelihoods
loglik <- function(l) sum(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.

General options for stochastic gradient methods:

**SG_learningRate** 0.1, learning rate, numeric

**SG_batchSize** `NULL`, batch size for Stochastic Gradient Ascent. A positive integer, or `NULL` for full-batch gradent ascent.

**SG_clip** `NULL`, gradient clipping threshold. This is the max allowed squared Euclidean norm of the gradient. If the actual norm of the gradient exceeds (square root of) this threshold, the gradient will be scaled back accordingly while preserving its direction. `NULL` means no clipping.

**SG_patience** `NULL`, or integer. Stopping condition: if the objective function is worse than its largest value so far this many times, the algorithm stops, and returns not the last parameter value but the one that gave the best results so far. This is mostly useful if gradient is computed on training data and the objective function on validation data.

**SG_patienceStep** 1L, integer. After how many epochs to check the patience value. 1 means to check (and hence to compute the objective function) at each epoch.

Options for SGA:

**SGA_momentum** 0, numeric momentum parameter for SGA. Must lie in interval $[0, 1]$.

Options for Adam:

**Adam_momentum1** 0.9, numeric in $[0, 1]$, the first moment momentum

**Adam_momentum2** 0.999, numeric in $[0, 1]$, the second moment momentum General options:

**iterlim** 150, stopping condition (the default differs for different methods). Stop if more than `iterlim` iterations performed. Note that 'iteration' may mean different things for different optimizers.

**max.rows** 20, maximum number of matrix rows to be printed when requesting verbosity in the optimizers.

**max.cols** 7, maximum number of columns to be printed. This also applies to vectors that are printed horizontally.

**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.

**storeParameters** `FALSE`, whether to store and return the parameter values at each epoch. If `TRUE`, the stored values can be retrieved with `storedParameters`-method. The parameters are stored as a matrix with rows corresponding to the epochs and columns to the parameter components.

**storeValues** `FALSE`, whether to store and return the objective function values at each epoch. If `TRUE`, the stored values can be retrieved with `storedValues`-method.

## Methods

**maxControl** `(...)` creates a "MaxControl" object. The arguments must be in the form `option1 = value1, option2 = value2, ....` The options should be slot names, but the method also supports selected other parameter forms for compatibility reasons e.g. "print.level" instead of "printLevel". In case there are more than one option with similar name, the last one overwrites the previous values. 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

```
library(maxLik)
## Create a 'maxControl' object:
maxControl(tol=1e-4, sann_tmax=7, printLevel=2)

## 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
## 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. start must be within the valid domain of logLik, i.e. logLik(start) must nor return NA. |

`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. The default method is "NR" for unconstrained problems, and "NM" or "BFGS" for constrained problems, depending on if the `grad` argument was provided. "BHHH" is a good alternative given the likelihood is returned observation-wise (see `maxBHHH`).

Note that stochastic gradient ascent (SGA) is currently not supported as this method seems to be rarely used for maximum likelihood estimation.

`constraints` either `NULL` for unconstrained maximization or a list, specifying the constraints. See `maxBFGS`.

`...` further arguments, such as `control`, `iterlim`, or `tol`, 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'. Useful methods include

- `AIC`: estimated parameter value
- `coef`: estimated parameter value
- `logLik`: log-likelihood value
- `nIter`: number of iterations
- `stdEr`: standard errors
- `summary`: print summary table with estimates, standard errors, p, and z-values.
- `vcov`: variance-covariance matrix

## 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.
## require much smaller tolerance
## setting 'tol=0' or negative essentially disables this stopping criterion
a <- maxLik(loglik, gradlik, hesslik, start=1,
            control=list(tol=-1, reltol=1e-12, gradtol=1e-12))
summary( a )
##
## Next, we give an example with vector argument:
## fit normal distribution by estimating mean and standard deviation
## by maximum likelihood
##
loglik <- function(param) {
                          # param: vector of 2, c(mean, standard deviation)
   mu <- param[1]
   sigma <- param[2]
   ll <- -0.5*N*log(2*pi) - N*log(sigma) - sum(0.5*(x - mu)^2/sigma^2)
                          # can use dnorm(x, mu, sigma, log=TRUE) instead
   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)
##
## Same example, but now with named parameters and a fixed value
##
resFix <- maxLik(loglik, start=c(mu=0, sigma=1), fixed="sigma")
summary(resFix)  # 'sigma' is exactly 1.000 now.
```

## 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.<br>`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. `start` must be a valid value for `fn`, i.e. `fn(start)` must return a numeric vector, not `NA`. |
| `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`.
If set to a negative value, the criterion is never fulfilled, and hence disabled.

**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 8. Negative value disables condition.

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

**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

object of class "maxim". Data can be extracted through the following methods:

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

`coef` estimated parameter value.

**gradient** vector, last calculated gradient value. Should be close to 0 in case of normal convergence.

**estfun** 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).

**returnCode** return code: a single digit code that describes the success of the optimization. See `returnCode`.

**returnMessage** a short message, describing the return code.

**activePar** logical vector, which parameters are optimized over. Contains only TRUE-s if no parameters are fixed.

**nIter** number of iterations.

**maximType** character string, type of maximization.

**maxControl** the optimization control parameters in the form of a MaxControl object.

The following components can only be extracted directly (with \$):

**last.step** a 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

**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.

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

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

```
## Fit exponential distribution by ML
t <- rexp(100, 2)  # create data with parameter 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 ;-)

## The same example 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, not the sum of those
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) {
                          # param is a 2-vector of c(mean, sd)
  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, sd=2
N <- length(x)
res <- maxNR(loglik, start=c(0,1)) # use 'wrong' start values
summary(res)

## The previous example with named parameters and a fixed value
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))
```

---

maxSGA *Stochastic Gradient Ascent*

---

## Description

Stochastic Gradient Ascent–based optimizers

## Usage

```
maxSGA(fn = NULL, grad = NULL, hess = NULL, start,
       nObs,
       constraints = NULL, finalHessian = FALSE,
       fixed = NULL, control=NULL, ... )
maxAdam(fn = NULL, grad = NULL, hess = NULL, start,
        nObs,
        constraints = NULL, finalHessian = FALSE,
        fixed = NULL, control=NULL, ... )
```

## Arguments

| | |
|---|---|
| fn | the function to be maximized. As the objective function values are not directly used for optimization, this argument is optional, given grad is provided. It must |

have the parameter vector as the first argument, and it must have an argument `index` to specify the integer index of the selected observations. It must return either a single number, or a numeric vector (this is is summed internally). 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 have an argument `index` to specify the integer index of selected observations. 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 `fn` returns an object with attribute `gradient`, this argument is ignored.

If `grad` is not supplied, it is computed by finite-difference method using `fn`. However, this is only adviseable for small-scale tests, not for any production run. Obviously, `fn` must be correctly defined in that case.

`hess` Hessian matrix of the function. Mainly for compatibility reasons, only used for computing the final Hessian if asked to do so by setting `finalHessian` to `TRUE`. It must have the parameter vector as the first argument and it must return the Hessian matrix of the objective function. If missing, either finite-difference Hessian, based on `gradient` or BHHH approach is computed if asked to do so.

`start` initial parameter values. If these have names, the names are also used for results.

`nObs` number of observations. This is used to partition the data into individual batches. The resulting batch indices are forwarded to the `grad` function through the argument `index`.

`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 when working with a log-likelihood function, and it requires the gradient/log-likelihood to be supplied by individual observations.

Hessian matrix is not often used for optimization problems where one applies SGA, but even if one is not interested in standard errors, it may provide useful information about the model performance. If computed by finite-difference method, the Hessian computation may be very slow.

`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. The ones used by these optimizers are

**SGA_momentum** 0, numeric momentum parameter for SGA. Must lie in interval $[0, 1]$. See details.
Adam-specific parameters

**Adam_momentum1** 0.9, numeric in interval $(0, 1)$, the first moment momentum

**Adam_momentum2** 0.999, numeric in interval $(0, 1)$, the second moment momentum General stochastic gradient parameters:

**SG_learningRate** step size the SGA algorithm takes in the gradient direction. If 1, the step equals to the gradient value. A good value is often 0.01–0.3

**SG_batchSize** SGA batch size, an integer between 1 and `nObs`. If `NULL` (default), the full batch gradient is computed.

**SG_clip** `NULL`, gradient clipping threshold. The algorithm ensures that $||g(\theta)||_2^2 \leq \kappa$ where $\kappa$ is the `SG_clip` value. If the actual norm of the gradient exceeds (square root of) $\kappa$, the gradient will be scaled back accordingly while preserving its direction. `NULL` means no clipping.
Stopping conditions:

**gradtol** stopping condition. Stop if norm of the gradient is less than `gradtol`. Default 0, i.e. do not use this condition. This condition is useful if the objective is to drive full batch gradient to zero on training data. It is not a good objective in case of the stochastic gradient, and if the objective is to optimize the objective on validation data.

**SG_patience** `NULL`, or integer. Stopping condition: the algorithm counts how many times the objective function has been worse than its best value so far, and if this exceeds `SG_patience`, the algorithm stops.

**SG_patienceStep** 1L, integer. After how many epochs to check the patience value. 1 means to check at each epoch, and hence to compute the objective function. This may be undesirable if the objective function is costly to compute.

**iterlim** stopping condition. Stop if more than `iterlim` epochs, return `code=4`. Epoch is a set of iterations that cycles through all observations. In case of full batch, iterations and epochs are equivalent. If `iterlim = 0`, does not do any learning and returns the initial values unchanged.

**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 epoch. Higher values will result in even more output.

**storeParameters** logical, whether to store and return the parameter values at each epoch. If `TRUE`, the stored values can be retrieved with `storedParameters`-method. The parameters are stored as a matrix with rows corresponding to the epochs and columns to the parameter components. There are `iterlim` + 1 rows, where the first one corresponds to the initial parameters.
Default `FALSE`.

**storeValues** logical, whether to store and return the objective function values at each epoch. If `TRUE`, the stored values can be retrieved with `storedValues`-method. There are `iterlim` + 1 values, where the first one corresponds to the value at the initial parameters.
Default `FALSE`.

See `maxControl` for more information.

`...` further arguments to `fn`, `grad` and `hess`. To maintain compatibility with the earlier versions, ... also passes certain control options to the optimizers.

## Details

Gradient Ascent (GA) is a optimization method where the algorithm repeatedly takes small steps in the gradient's direction, the parameter vector $\theta$ is updated as $\theta \leftarrow theta + \text{learningrate} \cdot \nabla f(\theta)$. In case of Stochastic GA (SGA), the gradient is not computed on the full set of observations but on a small subset, *batch*, potentially a single observation only. In certain circumstances this converges much faster than when using all observation (see *Bottou et al, 2018*).

If `SGA_momentum` is positive, the SGA algorithm updates the parameters $\theta$ in two steps. First, the momentum is used to update the "velocity" $v$ as $v \leftarrow \text{momentum} \cdot v + \text{learningrate} \cdot \nabla f(\theta)$, and thereafter the parameter $\theta$ is updates as $\theta \leftarrow \theta + v$. Initial velocity is set to 0.

The Adam algorithm is more complex and uses first and second moments of stochastic gradients to automatically adjust the learning rate. See *Goodfellow et al, 2016, page 301*.

The function `fn` is not directly used for optimization, only for printing or as a stopping condition. In this sense it is up to the user to decide what the function returns, if anything. For instance, it may be useful for `fn` to compute the objective function on either full training data, or on validation data, and just ignore the `index` argument. The latter is useful if using *patience*-based stopping. However, one may also choose to select the observations determined by the index to compute the objective function on the current data batch.

## Value

object of class "maxim". Data can be extracted through the following methods:

`\link{maxValue}`
: `fn` value at maximum (the last calculated value if not converged.)

`\link{coef}` estimated parameter value.

`\link{gradient}`
: vector, last calculated gradient value. Should be close to 0 in case of normal convergence.

`estfun` 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).

`\link{hessian}` Hessian at the maximum (the last calculated value if not converged).

`\link{storedValues}`
: return values stored at each epoch

`\link{storedParameters}`
: return parameters stored at each epoch

`\link{returnCode}`
: a numeric code that describes the convergence or error.

`\link{returnMessage}`
: a short message, describing the return code.

`\link{activePar}`
: logical vector, which parameters are optimized over. Contains only `TRUE`-s if no parameters are fixed.

`\link{nIter}` number of iterations.

`\link{maximType}`
character string, type of maximization.

`\link{maxControl}`
the optimization control parameters in the form of a `MaxControl` object.

## Author(s)

Ott Toomet, Arne Henningsen

## References

Bottou, L.; Curtis, F. & Nocedal, J.: Optimization Methods for Large-Scale Machine Learning *SIAM Review*, 2018, **60**, 223–311.

Goodfellow, I.; Bengio, Y.; Courville, A. (2016): Deep Learning, *MIT Press*

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

## See Also

A good starting point to learn about the usage of stochastic gradient ascent in **maxLik** package is the vignette "Stochastic Gradient Ascent in maxLik".

The other related functions are `maxNR` for Newton-Raphson, a popular Hessian-based maximization; `maxBFGS` for maximization using the BFGS, Nelder-Mead (NM), and Simulated Annealing (SANN) method (based on `optim`), also supporting inequality constraints; `maxLik` for a general framework for maximum likelihood estimation (MLE); `optim` for different gradient-based optimization methods.

## Examples

```
## estimate the exponential distribution parameter by ML
set.seed(1)
t <- rexp(100, 2)
loglik <- function(theta, index) sum(log(theta) - theta*t[index])
## Note the log-likelihood and gradient are summed over observations
gradlik <- function(theta, index) sum(1/theta - t[index])
## Estimate with full-batch
a <- maxSGA(loglik, gradlik, start=1, control=list(iterlim=1000,
            SG_batchSize=10), nObs=100)
            # note that loglik is not really needed, and is not used
            # here, unless more print verbosity is asked
summary(a)
##
## demonstrate the usage of index, and using
## fn for computing the objective function on validation data.
## Create a linear model where variables are very unequally scaled
##
## OLS loglik function: compute the function value on validation data only
loglik <- function(beta, index) {
   e <- yValid - XValid %*% beta
```

```
   -crossprod(e)/length(y)
}
## OLS gradient: compute it on training data only
## Use 'index' to select the subset corresponding to the minibatch
gradlik <- function(beta, index)  {
   e <- yTrain[index] - XTrain[index,,drop=FALSE] %*% beta
   g <- t(-2*t(XTrain[index,,drop=FALSE]) %*% e)
   -g/length(index)
}
N <- 1000
## two random variables: one with scale 1, the other with 100
X <- cbind(rnorm(N), rnorm(N, sd=100))
beta <- c(1, 1)  # true parameter values
y <- X %*% beta + rnorm(N, sd=0.2)
## training-validation split
iTrain <- sample(N, 0.8*N)
XTrain <- X[iTrain,,drop=FALSE]
XValid <- X[-iTrain,,drop=FALSE]
yTrain <- y[iTrain]
yValid <- y[-iTrain]
##
## do this without momentum: learning rate must stay small for the gradient not to explode
cat("  No momentum:\n")
a <- maxSGA(loglik, gradlik, start=c(10,10),
           control=list(printLevel=1, iterlim=50,
                        SG_batchSize=30, SG_learningRate=0.0001, SGA_momentum=0
                        ), nObs=length(yTrain))
print(summary(a))  # the first component is off, the second one is close to the true value
## do with momentum 0.99
cat("  Momentum 0.99:\n")
a <- maxSGA(loglik, gradlik, start=c(10,10),
           control=list(printLevel=1, iterlim=50,
                        SG_batchSize=30, SG_learningRate=0.0001, SGA_momentum=0.99
                        # no momentum
                        ), nObs=length(yTrain))
print(summary(a))  # close to true value
```

---

maxValue *Function value at maximum*

---

## Description

Returns the function value at (estimated) maximum.

## Usage

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

### Arguments

| | |
|---|---|
| x | a statistical model, or a result of maximisation, created by maxLik, maxNR or another optimizer. |
| ... | further arguments for other methods |

### Value

numeric, the value of the objective function at maximum. In general, it is the last calculated value in case the process did not converge.

### 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)
maxValue(a)
```

---

## 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`. `nobs` users the standard generic from **stats4**, `nObs` (deprecated) uses the S3 generic from **miscTools**.

### Usage

```
## S3 method for class 'maxLik'
nObs(x, ...)
## S4 method for signature 'maxLik'
nobs(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | a statistical model estimated by Maximum Likelihood using `maxLik`. |
| `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, 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
```

---

`objectiveFn` *Optimization Objective Function*

---

## Description

This function returns the optimization objective function from a 'maxim' object.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | an optimization result, inheriting from class 'maxim' |
| `...` | other arguments for methods |

## Value

function, the function that was optimized. It can be directly called, given that all necessary variables are accessible from the current environment.

## Author(s)

Ott Toomet

## Examples

```
hatf <- function(theta) exp(- theta %*% theta)
res <- maxNR(hatf, start=c(0,0))
print(summary(res))
print(objectiveFn(res))
print(objectiveFn(res)(2)) # 0.01832
```

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 describes either the the convergence (stopping condition), or the problem if any.

The known codes and the related messages are:

**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.

**10** Lost patience: the optimizer has hit an inferior value too many times (see maxSGA for more information)

**100** Initial value out of range.

## 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)
```

---

# storedValues *Return the stored values of optimization*

---

## Description

Retrieve the objective function value for each iteration if stored during the optimization.

## Usage

```
storedValues(x, ...)
## S3 method for class 'maxim'
storedValues(x, ...)
storedParameters(x, ...)
## S3 method for class 'maxim'
storedParameters(x, ...)
```

### Arguments

| | |
|---|---|
| x | a result of maximization, created by maxLik, maxSGA or another optimizer. |
| ... | further arguments for other methods |

### Details

These is a generic method. If asked by control parameter storeValues=TRUE or storeParameters=TRUE, certain optimization methods store the objective function value and the parameter value at each epoch. These methods retrieves the stored values.

### Value

- storedValues: a numeric vector, one value for each iteration
- storedParameters: a numeric matrix with rows corresponding to the iterations and columns to the parameter components.

In both cases, the first value stored corresponds to the initial parameter.

### Author(s)

Ott Toomet

### See Also

maxSGA, maxControl

### Examples

```
## Estimate the exponential distribution parameter
t <- rexp(100, 2)
loglik <- function(theta, index) sum(log(theta) - theta*t[index])
## Estimate with numeric gradient and numeric Hessian
a <- maxSGA(loglik, start=1,
            control=list(storeValues=TRUE, storeParameters=TRUE, iterlim=10),
            nObs=100)
storedValues(a)
storedParameters(a)
```

---

## summary.maxim *Summary method for maximization*

---

### Description

Summarizes the general maximization results in a way that does not assume the function is log-likelihood.

## Usage

```
## S3 method for class 'maxim'
summary( object, hessian=FALSE, unsucc.step=FALSE, ... )
## S3 method for class 'summary.maxim'
print(x,
                                max.rows=getOption("max.rows", 20),
                                max.cols=getOption("max.cols", 7),
                                ... )
```

## 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 |
| x | object of class summary.maxim, summary of maximization result. |
| max.rows | maximum number of rows to be printed. This applies to the resulting coefficients (as those are printed as a matrix where the other column is the gradient), and to the Hessian if requested. |
| max.cols | maximum number of columns to be printed. Only Hessian output, if requested, uses this argument. |
| ... | currently not used. |

## Value

Object of class summary.maxim, intended to be printed with corresponding print method.

## 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  # concave parabola
  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.
res <- maxNR( f, start = c(0,0) )
summary(res)
summary(res, hessian=TRUE)
```

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 for maximum likelihood estimation, confint for confidence intervals, and tidy and glance for alternative quick summaries of the ML results.

## 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 = 5,
SUMTRho0 = NULL,
SUMTMaxIter = 100,
control = NULL, ...)
```

## 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 5. Larger numbers achieve faster convergence but may throw the inner optimizer off.

`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.

`SUMTMaxIter` Maximum SUMT iterations

`control` either a list of control parameters, or a 'MaxControl' object to specify the parameters for the inner unconstrained optimizer. See `maxNR` for details.

`...` 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 $\varrho$ 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))
```

---

# tidy.maxLik *tidy and glance methods for maxLik objects*

---

## Description

These methods return quick summary information about the estimated model, either one parameter per row (tibble), or on a single row (glance). Both require the **tibble** package to be installed.

## Usage

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

## Arguments

| | |
|---|---|
| x | object of class 'maxLik' or 'maxim'. |
| ... | Not used. |

## Value

`tidy()` for 'maxLik' objects returns a tibble (data frame) with one row per parameter, and with the following columns:

**term** The name of the estimated parameter (parameters are sequentially numbered if names missing).

**estimate** The estimated parameter.

**std.error** The standard error of the estimate.

**statistic** The $z$-statistic of the estimate.

**p.value** The $p$-value.

This is essentially the same table as `summary`-method prints, just in form of a tibble (data frame).

For 'maxim' objects, the corresponding tibble contains

**term** The name of the estimated parameter (parameters are sequentially numbered if names missing).

**estimate** The estimated parameter.

**gradient** The corresponding gradient value.

`glance()` for 'maxLik' objects returns an one-row tibble with columns:

**df** The degrees of freedom of the model.

**logLik** The log-likelihood of the model.

**AIC** Akaike's Information Criterion for the model.

**nobs** The number of observations, if this is available, otherwise `NA`.

For 'maxim' objects the tibble contains

**value** The function value at optimum.

**iterations** The number of iterations spent (what exactly is "iteration" depends on the optimizer).

**gradient** Norm of gradient at the optimum.

**code** optimizer return code (see `returnCode`).

## Author(s)

David Hugh-Jones

## See Also

The functions `tidy` and `glance` in package **generics**, and `summary` to display the "standard" summary information.

## Examples

```
## Example with a single parameter
t <- rexp(100, 2)
loglik <- function(theta) log(theta) - theta*t
a <- maxLik(loglik, start=2)
tidy(a)
glance(a)
## Example with a parameter vector
x <- rnorm(100)
loglik <- function(theta) {
   dnorm(x, mean=theta[1], sd=theta[2], log=TRUE)
}
a <- maxLik(loglik, start=c(mu=0, sd=1))
tidy(a)
glance(a)
```

---

`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.

If unable to compute the matrix (e.g. Hessian was not computed by the model), then it returns a matrix of `NA`-s of correct size.

## 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


∗ **debugging**
  condiNumber, 10
∗ **math**
  compareDerivatives, 7
  condiNumber, 10
  numericGradient, 42
∗ **methods**
  activePar, 4
  AIC.maxLik, 5
  bread.maxLik, 6
  gradient, 14
  hessian, 15
  logLik.maxLik, 16
  maximType, 24
  maxValue, 38
  nIter, 39
  nObs.maxLik, 40
  nParam.maxim, 41
  objectiveFn, 44
  returnCode, 45
  storedValues, 46
  summary.maxim, 47
  vcov.maxLik, 54
∗ **models**
  summary.maxLik, 49
∗ **optimize**
  activePar, 4
  fnSubset, 12
  hessian, 15
  maxBFGS, 17
  maximType, 24
  maxLik, 25
  maxNR, 27
  maxSGA, 33
  objectiveFn, 44
  sumt, 50
∗ **print**
  summary.maxim, 47
∗ **utilities**
  compareDerivatives, 7
  condiNumber, 10
  fnSubset, 12
  MaxControl-class, 21
  numericGradient, 42
  returnCode, 45

activePar, 4, *15, 16*
AIC, *26*
AIC.maxLik, 5

bread, *6, 7*
bread *(*bread.maxLik*)*, 6
bread.maxLik, 6

coef, *3*, *26*, *30*
coef.maxim *(*AIC.maxLik*)*, 5
coef.maxLik *(*AIC.maxLik*)*, 5
coef.summary.maxLik *(*summary.maxLik*)*, 49
compareDerivatives, *3*, 7, *31*, *43*
condiNumber, *3*, 10, *16*
confint, *12*, *50*
confint *(*confint.maxLik*)*, 11
confint.maxLik, 11
constrOptim2, *19*, *29*, *34*

deriv, *9*, *43*
dlmMLE, *13*

estfun, *14*
estfun *(*gradient*)*, 14

fnSubset, 12

glance, *50*, *54*
glance.maxim *(*tidy.maxLik*)*, 52
glance.maxLik *(*tidy.maxLik*)*, 52
gradient, 14

hessian, *14*, 15

kappa, *10, 11*

logLik, *26*
logLik.maxLik, 16
logLik.summary.maxLik *(*logLik.maxLik*)*, 16

maxAdam *(*maxSGA*)*, 33
maxBFGS, *3*, 17, *26, 27*, *32*, *37*, *46*
maxBFGSR, *20*, *26*
maxBFGSR *(*maxNR*)*, 27
maxBHHH, *3*, *18–20*, *26*, *32*
maxBHHH *(*maxNR*)*, 27
maxCG, *26*
maxCG *(*maxBFGS*)*, 17
MaxControl, *20*, *31*, *37*
maxControl, *36*, *47*
maxControl *(*MaxControl-class*)*, 21
maxControl,MaxControl-method *(*MaxControl-class*)*, 21
maxControl,maxim-method *(*MaxControl-class*)*, 21
maxControl,missing-method *(*MaxControl-class*)*, 21
MaxControl-class, 21
maximType, 24
maxLik, *3, 4*, *7*, *10, 11*, *13, 14*, *16, 17*, 25, *32*, *37*, *39–41*, *47*, *50*, *54, 55*
maxLik-package, 2, *23*
maxNM, *26*
maxNM *(*maxBFGS*)*, 17
maxNR, *4*, *13*, *18*, *20–22*, *25–27*, 27, *37*, *39, 40*, *46*, *48*, *50, 51*
maxSANN, *26*
maxSANN *(*maxBFGS*)*, 17
maxSGA, *3*, 33, *46, 47*
maxValue, *30*, 38

nIter, *26*, 39
nlm, *20*, *27*, *32*
nObs, *4*, *41, 42*
nobs, *41*
nobs,maxLik-method *(*nObs.maxLik*)*, 40
nObs.maxLik, 40
nParam, *41*
nParam.maxim, 41
numericGradient, *9*, 42
numericHessian *(*numericGradient*)*, 42
numericNHessian *(*numericGradient*)*, 42

objectiveFn, 44
optim, *3*, *13*, *17–22*, *27*, *32*, *37*

print.maxim *(*summary.maxim*)*, 47
print.maxLik *(*maxLik*)*, 25
print.summary.maxim *(*summary.maxim*)*, 47

returnCode, *30*, 45, *48*, *53*
returnMessage, *48*
returnMessage *(*returnCode*)*, 45

show,MaxControl-method *(*MaxControl-class*)*, 21
show,maxim-method *(*summary.maxim*)*, 47
show,maxLik-method *(*maxLik*)*, 25
stdEr, *3*, *12*, *26*
stdEr.maxLik *(*AIC.maxLik*)*, 5
storedParameters, *23*, *35*
storedParameters *(*storedValues*)*, 46
storedValues, *23*, *35*, 46
summary, *3*, *12*, *26*, *54*
summary.maxim, 47
summary.maxLik, 49
sumt, *19*, *29*, *34*, 50, *52*

tidy, *50*, *54*
tidy.maxim *(*tidy.maxLik*)*, 52
tidy.maxLik, 52

vcov, *26*, *55*
vcov.maxLik, 54