Comput Stat (2011) 26:443–458
DOI 10.1007/s00180-010-0217-1

ORIGINAL PAPER

maxLik: A package for maximum likelihood estimation in R

Arne Henningsen · Ott Toomet

Received: 15 October 2009 / Accepted: 7 September 2010 / Published online: 22 September 2010

Abstract This paper describes the package **maxLik** for the statistical environment R. The package is essentially a unified wrapper interface to various optimization routines, offering easy access to likelihood-specific features like standard errors or information matrix equality (BHHH method). More advanced features of the optimization algorithms, such as forcing the value of a particular parameter to be fixed, are also supported.

Keywords Maximum likelihood · Optimization

JEL Classification C87

1 Introduction

The Maximum Likelihood (ML) method is one of the most important techniques in statistics and econometrics. Most statistical and econometric software packages include ready-made routines for maximum likelihood estimations of many standard models such as logit, probit, sample-selection, count-data, or survival models. However, if practitioners and researchers want to estimate non-standard models or develop

A. Henningsen (✉)
Institute of Food and Resource Economics, University of Copenhagen,
Rolighedsvej 25, 1958 Frederiksberg C, Denmark
e-mail: arne@foi.dk; arne.henningsen@gmail.com

O. Toomet
Department of Economics, Aarhus School of Business, University of Aarhus,
Hermodsvej 22, 8230 Åbyhøj, Denmark
e-mail: ottt@asb.dk

O. Toomet
Department of Economics, University of Tartu,
Narva 4, 51009 Tartu, Estonia

new estimators, they have to implement the routines for the maximum likelihood estimations themselves. Several popular statistical packages include frameworks for simplifying the estimation, allowing the user to easily choose between a number of optimization algorithms, different ways of calculating variance-covariance matrices, and easy reporting of the results. The examples include the `ml` command in stata and the `maxlik` library for GAUSS.

The free software environment for statistical computing and graphics R (R Development Core Team 2009) has included built-in optimization algorithms since its early days. The first general-purpose ML framework, function `mle` in the built-in package **stats4**, was added in 2003, and an extension, `mle2` in package **bbmle** (Bolker 2009), in 2007. However, both of these packages are based on a general-purpose optimizer `optim` which does not include an option to use the Newton-Raphson algorithm. In particular, its variant, the Berndt-Hall-Hall-Hausman algorithm (Berndt et al. 1974), is very popular for ML problems. The R package **maxLik** (Toomet and Henningsen 2010) is intended to fill this gap.[1] The package can be used both by end-users, developing their own statistical methods, and by package developers, implementing ML estimators for specific models. For instance, the packages **censReg** (Henningsen 2010), **mhurdle** (Carlevaro et al. 2010), **mlogitBMA** (Sevcikova and Raftery 2010), **pglm** (Croissant 2010), **sampleSelection** (Toomet and Henningsen 2008), and **truncreg** (Croissant 2009) use the **maxLik** package for their maximum likelihood estimations.

The **maxLik** package (currently version 1.0) is available from CRAN (http://cran.r-project.org/package=maxLik), R-Forge (http://r-forge.r-project.org/projects/maxlik/), and its homepage (http://www.maxLik.org/). This paper focuses on the maximum likelihood related usage of the package; the other features (including finite-difference derivatives and optimization) are only briefly mentioned.

The paper proceeds as follows: in the next section we explain the implementation of the package. Section 3 describes the usage of the package, including the basic and more advanced features, and Sect. 4 concludes.

2 Implementation

The R package **maxLik** is designed to provide a single, unified interface for different optimization routines, and to treat the results in a way suitable for maximum likelihood (ML) estimation. The package implements a flexible multi-purpose Newton-Raphson type optimization routine in function `maxNRCompute`. This internal function is not intended to be called by users but function `maxNR` provides a convenient user-interface and calls `maxNRCompute` for the actual optimization. This Newton-Raphson type algorithm is also used as the basis of function `maxBHHH`, which implements a Berndt-Hall-Hall-Hausman type algorithm (Berndt et al. 1974) that is popular for ML problems. In addition, the Broyden-Fletcher-Goldfarb-Shanno algorithm (Broyden 1970; Fletcher 1970; Goldfarb

[1] The other ML-related packages offer a few capabilities that are missing in **maxLik** (e.g. likelihood profiling). This may be suboptimal from the user's perspective who might prefer a single "ultimate" ML package. However, a number of partially overlapping projects seems to characterize a large part of the free software world.

1970; Shanno 1970), the Nelder-Mead routine (Nelder and Mead 1965), and a simulated annealing method (Bélisle 1992) are available in a unified way in functions `maxBFGS`, `maxNM`, and `maxSANN`, respectively. These three functions are predominantly wrapper functions around the internal function `maxOptim`, which calls function `optim` (from the built-in package **stats**) for the actual optimization. The **maxLik** package provides a further implementation of the BFGS optimizer, `maxBFGSR`, which —unlike the `optim`-based methods—is written solely in R.[2]

The **maxLik** package is designed in two layers. The first (innermost) is the optimization (maximization) layer: all the maximization routines are designed to have a unified and intuitive interface which allows the user to switch easily between them. All the main arguments have identical names and similar order; only method-specific parameters may vary. These functions can be used for different types of optimization tasks, both related and not related to the likelihood. They return an S3 object of class `maxim` including both estimated parameters and various diagnostics information.

The second layer is the likelihood maximization layer. The most important tool of this layer is the function `maxLik`. Its main purpose is to treat the inputs and maximization results in a ML-specific way (for instance, computing the variance-covariance matrix based on the estimated Hessian). The `maxBHHH` function belongs to this layer as well, being essentially a call for `maxNR` using the information matrix equality as the way to approximate the Hessian matrix. A new class `maxLik` is added to the returned maximization object for automatic selection of the ML-related methods.

The maximization layer supports linear equality and inequality constraints. The equality constraints are estimated using the sequential unconstrained maximization technique (SUMT), which is also implemented in the **maxLik** package. This is achieved by adding a (initially tiny) penalty term, related to violation of the constraints, to the objective function. Thereafter the problem is repeatedly solved while the penalty is increased for every new repetition. The inequality constraints are delegated to `constrOptim` in the package **stats**. The `maxLik` function is aware of the constraints and is able to select a suitable optimization method; however, no attempt is made to correct the resulting variance-covariance matrix (just a warning is printed). As the constrained optimization should still be considered as experimental, we refer the reader to the documentation of the package for examples.

The **maxLik** package is implemented using S3 classes. Corresponding methods can handle the likelihood-specific properties of the estimate including the fact that the inverse of the negative Hessian is the approximate variance-covariance matrix of the estimated parameters. The most important methods for objects of class `"maxLik"` are: `summary` for returning (and printing) summary results, `coef` for extracting the estimated parameters, `vcov` for calculating the variance covariance matrix of the estimated parameters, `stdEr` for calculation standard errors of the estimates, `logLik` for extracting the log-likelihood value, and `AIC` for calculating the Akaike information criterion.

[2] The `maxBFGSR` optimizer supplies more debugging information compared to `optim`-based methods. We are grateful to Yves Croissant for providing the core of this code.

3 Using the maxLik package

3.1 Basic usage

Like other R packages, the **maxLik** package must be installed and loaded before it can be used. The following command loads the **maxLik** package:

```
> library("maxLik")
```

The most important user interface of the **maxLik** package is a function with the (same) name `maxLik`. As explained above, `maxLik` is mostly a wrapper for different optimization routines with a few additional features, useful for ML estimations. This function has two mandatory arguments, `logLik` and `start`. The first argument (`logLik`) must be a function that calculates the log-likelihood value as a function of the parameter (usually parameter vector). The second argument (`start`) must be a vector of starting values.

We demonstrate the usage of the **maxLik** package by a simple example: we estimate the parameters of a normal distribution based on a random sample. First, we generate a vector (x) of $N = 100$ draws from a normal distribution with a mean of $\mu = 1$ and a standard deviation of $\sigma = 2$:

```
> set.seed(123)
> x <- rnorm(100, mean = 1, sd = 2)
```

The logarithm of the probability density of the sample (i.e. the log-likelihood function) is

$$\log(L(x;\mu,\sigma)) = -\frac{1}{2}N\log(2\pi) - N\log(\sigma) - \frac{1}{2}\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^2}. \qquad (1)$$

Given the log-likelihood function above, we create an R function that calculates the log-likelihood value. Its first argument must be the vector of the parameters to be estimated and it must return the log-likelihood value.[3] The easiest way to implement this log-likelihood function is to use the capabilities of the function `dnorm`:

```
> logLikFun <- function(param) {
+     mu <- param[1]
+     sigma <- param[2]
+     sum(dnorm(x, mean = mu, sd = sigma, log = TRUE))
+ }
```

For the actual estimation we set the first argument (`logLik`) equal to the log-likelihood function that we have defined above (`logLikFun`) and we use the parameters of a standard normal distribution ($\mu = 0, \sigma = 1$) as starting values (argument `start`). Assigning names to the vector of starting values is not required but has the

[3] Alternatively, it could return a numeric vector where each element is the log-likelihood value corresponding to an (independent) individual observation (see below).

advantage that the returned estimates have also names, which improves the readability of the results.[4]

```
> mle <- maxLik(logLik = logLikFun, start = c(mu = 0, sigma = 1))
> summary(mle)

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 7 iterations
Return code 1: gradient close to zero
Log-Likelihood: -201.5839
2  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18081    0.18168  6.4996 8.055e-11 ***
sigma  1.81648    0.12843 14.1433 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

For convenience, the estimated parameters can be accessed by the `coef` method and standard errors by the `stdEr` method:[5]

```
> coef(mle)

      mu    sigma
1.180812 1.816481

> stdEr(mle)

       mu     sigma
0.1816755 0.1284338
```

As expected, the estimated parameters are equal to the mean and the standard deviation (without correction for degrees of freedom) of the values in vector x.[6]

```
> all.equal(coef(mle), c(mean(x), sqrt(sum((x - mean(x))^2)/100)),
+     check.attributes = FALSE)

[1] TRUE
```

If no analytical gradient is provided by the user, finite-difference gradient and Hessian are calculated by the functions `numericGradient` and `numericNHessian`, which are also included in the **maxLik** package. While the maximization of the likelihood function of this simple model works well with finite-difference gradients

[4] Alternatively, if all the components of the parameter vector have standardized names, one may prefer using the command `with( as.list(param), sum(dnorm(x, mean=mu, sd=sigma, log=TRUE)))` for evaluating the likelihood expression. We are grateful to a referee for this suggestion.

[5] The generic function `stdEr` is defined in package **miscTools** (Henningsen and Toomet 2010).

[6] The function `all.equal` considers two elements as equal if either the mean absolute difference or the mean relative difference is smaller than the tolerance (defaults to `.Machine$double.eps^0.5`, usually around $1.5 \cdot 10^{-8}$).

and Hessians, this may not be the case for more complex models. Finite-difference derivatives may be costly to compute, and, even more, they may turn out to be noisy and unreliable. In this way finite-difference derivatives might either slow down the estimation or even impede the convergence. In these cases, the user is recommended to either provide analytical derivatives or switch to a more robust estimation method, such as Nelder-Mead or SANN, which is not based on gradients.

The gradients of the log-likelihood function with respect to the two parameters are

$$\frac{\partial \log(L(x, \mu, \sigma))}{\partial \mu} = \sum_{i=1}^{N} \frac{(x_i - \mu)}{\sigma^2} \tag{2}$$

$$\frac{\partial \log(L(x, \mu, \sigma))}{\partial \sigma} = -\frac{N}{\sigma} + \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma^3}. \tag{3}$$

This can be calculated in R by the following function:

```
> logLikGrad <- function(param) {
+     mu <- param[1]
+     sigma <- param[2]
+     N <- length(x)
+     logLikGradValues <- numeric(2)
+     logLikGradValues[1] <- sum((x - mu)/sigma^2)
+     logLikGradValues[2] <- -N/sigma + sum((x - mu)^2/sigma^3)
+     return(logLikGradValues)
+ }
```

Now we call the `maxLik` function and use argument `grad` to specify the function that calculates the gradients:

```
> mleGrad <- maxLik(logLik = logLikFun, grad = logLikGrad,
+     start = c(mu = 0, sigma = 1))
> all.equal(logLik(mleGrad), logLik(mle))

[1] TRUE

> all.equal(coef(mleGrad), coef(mle))

[1] TRUE

> all.equal(stdEr(mleGrad), stdEr(mle))

[1] "Mean relative difference: 0.0001230090"
```

Providing analytical gradients has no (relevant) effect on the estimates but their standard errors are slightly different.

Instead of writing a separate gradient function, the user may prefer to compute the gradient value in the log-likelihood function itself, because this might be a lot faster under certain circumstances. The computed gradient value may be added to the log-likelihood value as attribute "gradient", analogously to the case of function `nlm` of the `stats` package.

The analytic Hessian of the log-likelihood function can be provided by the argument `hess`. If the user provides a function to calculate the gradients but does not use argument `hess`, the Hessians are calculated by function `numericHessian` using the finite-difference approach. The elements of the Hessian matrix of the log-likelihood function for the normal distribution are

$$\frac{\partial^2 \log(L(x,\mu,\sigma))}{(\partial\mu)^2} = -\frac{N}{\sigma^2} \tag{4}$$

$$\frac{\partial^2 \log(L(x,\mu,\sigma))}{\partial\mu\ \partial\sigma} = -2\sum_{i=1}^{N}\frac{(x_i-\mu)}{\sigma^3} \tag{5}$$

$$\frac{\partial^2 \log(L(x,\mu,\sigma))}{(\partial\sigma)^2} = \frac{N}{\sigma^2} - 3\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^4}. \tag{6}$$

They can be calculated in R using the following function:

```
> logLikHess <- function(param) {
+     mu <- param[1]
+     sigma <- param[2]
+     N <- length(x)
+     logLikHessValues <- matrix(0, nrow = 2, ncol = 2)
+     logLikHessValues[1, 1] <- -N/sigma^2
+     logLikHessValues[1, 2] <- -2 * sum((x - mu)/sigma^3)
+     logLikHessValues[2, 1] <- logLikHessValues[1, 2]
+     logLikHessValues[2, 2] <- N/sigma^2 - 3 * sum((x -
+         mu)^2/sigma^4)
+     return(logLikHessValues)
+ }
```

Now we call the `maxLik` function with argument `hess` set to this function:

```
> mleHess <- maxLik(logLik = logLikFun, grad = logLikGrad,
+     hess = logLikHess, start = c(mu = 0, sigma = 1))
> all.equal(list(logLik(mleHess), coef(mleHess), vcov(mleHess)),
+     list(logLik(mleGrad), coef(mleGrad), vcov(mleGrad)))
```

```
[1] TRUE
```

Providing an analytical Hessian has no (relevant) effect on the outcome of the ML estimation in our simple example. However, as in the case of finite-difference gradients, calculating finite-difference Hessians may turn out to be slow and unreliable. If the user prefers to pre-compute the Hessian matrix instead of supplying argument `hess`, this can be done by setting attribute "hessian" of the object returned by the log-likelihood function to the computed Hessian matrix.

3.2 Optimization methods

The `maxLik` function allows the user to select among five optimization algorithms by argument `method`. It defaults to `"NR"` for the Newton-Raphson algorithm. The other

options are "BHHH" for Berndt-Hall-Hall-Hausman (Berndt et al. 1974), "BFGS" for Broyden-Fletcher-Goldfarb-Shanno (Broyden 1970; Fletcher 1970; Goldfarb 1970; Shanno 1970), "NM" for Nelder-Mead (Nelder and Mead 1965), and "SANN" for simulated annealing (Bélisle 1992). The Newton-Raphson algorithm uses (finite-difference or analytical) gradients and Hessians; the BHHH and BFGS algorithms use only (finite-difference or analytical) gradients; the NM and SANN algorithms use neither gradients nor Hessians but only function values. The gradients and Hessians provided by the user through the arguments grad and hess are always accepted. In this way the user can easily switch the optimization method without changing the arguments. If arguments grad or hess are provided but the selected method does not require this information— for instance for the Nelder-Mead method— they are ignored during the optimization. However, even if the optimization method itself does not make use of the Hessian, this information is used for computing the (final) variance-covariance matrix of the parameters (except for the "BHHH" method.)[7]

In general, it is advisable to use all the available information, e.g. to use the "NR" method if both analytical gradients and Hessians are available, one of the gradient-based methods (either "BHHH" or "BFGS") if analytical gradients but no Hessians are available, and to resort to the value-only methods only if gradients are not provided.

3.2.1 Berndt-hall-hall-Hausman (BHHH)

The idea of the BHHH method is based on information matrix equality, replacing the Hessian by the negative of the sum over the outer products of the gradients of individual (independent) observations (see e.g. Greene 2008, p. 490). Note that this approximation is only valid while maximizing log-likelihood and hence this method is usually not included in general-purpose optimizers. In order to use the BHHH method, the user has to provide gradient vectors by individual observations. This can be achieved either by providing a corresponding gradient function or attribute (see below) or by providing a vector of individual observation-specific likelihood values by the log-likelihood function itself (if no analytical gradients are provided). In the latter case, finite-difference gradients are used to calculate the Hessian.

We modify our example above accordingly: instead of returning a single summary value of log-likelihood, we return the values by individual observations by simply removing the sum operator:

```
> logLikFunInd <- function(param) {
+     mu <- param[1]
+     sigma <- param[2]
+     dnorm(x, mean = mu, sd = sigma, log = TRUE)
+ }
> mleBHHH <- maxLik(logLik = logLikFunInd, start = c(mu = 0,
+     sigma = 1), method = "BHHH")
> summary(mleBHHH)
--------------------------------------------
Maximum Likelihood estimation
```

[7] The user can force maxLik to use the BHHH method for computing the final Hessian for other optimizers as well, see documentation for the argument finalHessian.

```
BHHH maximisation, 13 iterations
Return code 2: successive function values within tolerance limit
Log-Likelihood: -201.5839
2  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18081    0.18183  6.4941 8.354e-11 ***
sigma  1.81648    0.13408 13.5473 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

```
> all.equal(logLik(mleBHHH), logLik(mle))
```

```
[1] TRUE
```

```
> all.equal(coef(mleBHHH), coef(mle))
```

```
[1] "Mean relative difference: 2.510385e-07"
```

```
> all.equal(vcov(mleBHHH), vcov(mle))
```

```
[1] "Mean relative difference: 0.06980237"
```

While the estimated parameters and the corresponding log-likelihood value are virtually identical to the previous estimates, the covariance matrix of the estimated parameters is slightly different. This is because the outer product approximation may differ from the derivative-based Hessian in finite samples (Calzolari and Fiorentini 1993).

If the user chooses to provide analytical gradients, the function that calculates the gradients (argument `grad`) must return a numeric matrix, where each column represents the gradient with respect to the corresponding element of the parameter vector and each row corresponds to an individual observation. Note that in this case, the log-likelihood function itself does not have to return a vector of log-likelihood values by observations, as the gradient by observation is supplied by the `grad` function. In the following example, we define a function that calculates the gradient matrix and we estimate the model by BHHH method using this gradient matrix and the single summed log-likelihood from the Newton-Raphson example.

```
> logLikGradInd <- function(param) {
+     mu <- param[1]
+     sigma <- param[2]
+     logLikGradValues <- cbind((x - mu)/sigma^2, -1/sigma +
+         (x - mu)^2/sigma^3)
+     return(logLikGradValues)
+ }
> mleGradBHHH <- maxLik(logLik = logLikFun, grad = logLikGradInd,
+     start = c(mu = 0, sigma = 1), method = "BHHH")
> all.equal(list(logLik(mleBHHH), coef(mleBHHH), vcov(mleBHHH)),
+     list(logLik(mleGradBHHH), coef(mleGradBHHH), vcov(mleGradBHHH)))
```

```
[1] TRUE
```

Estimates based on finite-difference gradients and analytical gradients are virtually identical in our simple example.

3.2.2 Nelder-Mead (NM) and other methods

The other maximization methods: Nelder-Mead, Broyden-Fletcher-Goldfarb-Shanno, and Simulated Annealing, are implemented by a call to the the `optim` function in package **stats**. In order to retain compatibility with the BHHH method, all these methods accept the log-likelihood function returning a vector of individual likelihoods (these are summed internally). A function to compute a gradient matrix with gradients of individual observations is accepted as well. If the user does not provide gradients, the gradients are computed by finite-difference approach.

We give an example using the gradient-free Nelder-Mead method:

```
> mleNM <- maxLik(logLik = logLikFun, start = c(mu = 0,
+     sigma = 1), method = "NM")
> summary(mleNM)

--------------------------------------------
Maximum Likelihood estimation
Nelder-Mead maximisation, 63 iterations
Return code 0: successful convergence
Log-Likelihood: -201.5839
2  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18061    0.18168  6.4984 8.116e-11 ***
sigma  1.81664    0.12849 14.1379 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------

> logLik(mleNM) - logLik(mleGrad)

[1] -1.361201e-06

> all.equal(coef(mleNM), coef(mleGrad))

[1] "Mean relative difference: 0.0001198403"

> all.equal(vcov(mleNM), vcov(mleGrad))

[1] "Mean relative difference: 0.001083386"
```

The estimates and the covariance matrix obtained from the Nelder-Mead algorithm slightly differ from previous results using other algorithms and the fit (log-likelihood value) of the model is slightly worse (smaller) than for the previous models.

Note that although the `summary` method reports the number of iterations for all the methods, the meaning of "iteration" may be completely different for different optimization techniques.

3.3 More advanced usage

The `maxLik` function supports a variety of other arguments, most of which are passed to the selected optimizer. Among the most important ones is `print.level` which

controls the output of debugging information (0 produces no debugging output, larger numbers produce more output). Optimization methods may also support various additional features, such as the temperature-related parameters for `maxSANN`. Those will not be discussed here; the interested reader is referred to the documentation of the corresponding optimizer.

3.3.1 Fixed parameter values

Below, we demonstrate how it is possible to keep certain parameters fixed as constants in the optimization process. This feature is implemented in all optimization methods supported by `maxLik`.

Let us return to our original task of estimating the parameters of a normal sample. However, assume we know that the true value of $\sigma = 2$. Instead of writing a new likelihood function, we may use the existing one while specifying that σ is kept fixed at 2. This is done via argument `fixed` of `maxLik`.[8] This argument allows for specifying the fixed parameters in three different ways: First, it can be a logical vector of length equal to that of the parameter vector, which specifies which components are not allowed to change, i.e. stay fixed at their starting values. Second, argument `fixed` can be an index vector that indicates the positions of the fixed parameters. Third, this argument can be a vector of character strings indicating the names of the fixed parameters, where the parameter names are taken from argument `start`. So, as σ was the second parameter, we may call:

```
> summary(maxLik(logLikFun, start = c(mu = 0, sigma = 2),
+     fixed = 2))

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 3 iterations
Return code 1: gradient close to zero
Log-Likelihood: -202.4536
1  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18081    0.20007   5.902 3.591e-09 ***
sigma  2.00000    0.00000      NA        NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

As we can see, the σ is indeed exactly 2. Its standard error is set to zero while the t-value is not defined. Note also that the estimate of μ is unchanged (indeed, its

[8] In earlier version of the **maxLik** package ($\leq$0.6), parameters could be fixed only in functions `maxNR` and `maxBHHH` by using argument `activePar`. This argument was a logical vector indicating, which parameters should *not* be fixed. Since version 0.7 of the **maxLik** package, it is recommended to fix parameters by argument `fixed`, because this has several advantages, e.g. it is easier to fix just a few out of many parameters and this works also with the BFGS, NM, and SANN method. However, argument `activePar` of function `maxNR` is kept for backward compatibility.

ML estimate is still the sample average) while the estimated standard error is different. Obviously, the log-likelihood value is lower in the constrained space, although the reader may verify that allowing σ to vary freely is an insignificant improvement according to the likelihood ratio test.

3.3.2 Automatic transformation of parameters to fixed constants

Next, we demonstrate, how it is possible to turn a parameter automatically to a fixed constant during the computations when using the `maxNR` optimizer. This may be useful when estimating a large number of similar models where parameters occasionally converge toward the boundary of the parameter space or another problematic region. Most popular optimization algorithms do not work well in such circumstances. In some cases, a solution is to replace the initial model with a simpler submodel, for instance replacing a density mixture with a single density component. Note that this problem cannot be easily handled by constrained maximization either, as the mixture parameters are not identified, if the weight of one component goes to zero. Below, we demonstrate this problem by estimating the parameters of a normal mixture on a sample, drawn from a single normal distribution. Note that this example is highly dependent on the initialization of the random number generator and the initial values for the estimation. This happens often with mixture models.

First, we demonstrate the outcome on a mixture of two distinct components. We generate $N = 1,000$ values from two different normal distributions:

```
> xMix <- c(rnorm(500), rnorm(500, mean = 1))
```

Variable `xMix` is a 50–50% mixture of two normal distributions: the first one has mean equal to 0 and the second has mean 1 (for simplicity, we fix the standard deviations to 1). The log-likelihood of a mixture is simply

$$l = \sum_{i=1}^{N} \log(\rho\phi(x_i - \mu_1) + (1 - \rho)\phi(x_i - \mu_2)), \tag{7}$$

where ρ is the proportion of the first component in the mixture and $\phi(\cdot)$ is the density function of the standard normal distribution. We implement this in R:

```
> logLikMix <- function(param) {
+     rho <- param[1]
+     if (rho < 0 || rho > 1)
+         return(NA)
+     mu1 <- param[2]
+     mu2 <- param[3]
+     ll <- log(rho * dnorm(xMix - mu1) + (1 - rho) * dnorm(xMix -
+         mu2))
+ }
```

Note that the function includes checking for feasible parameter values. If $\rho \notin [0, 1]$, it returns `NA`. This signals to the optimizer that the attempted parameter value was out of range, and forces it to find a new one (closer to the previous value). This is a way

of implementing box constraints in the log-likelihood function. The results look like the following:

```
> summary(m1 <- maxLik(logLikMix, start = c(rho = 0.5,
+     mu1 = 0, mu2 = 0.01)))

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 11 iterations
Return code 1: gradient close to zero
Log-Likelihood: -1536.981
3  free parameters
Estimates:
    Estimate Std. error t value   Pr(> t)
rho  0.27973    0.13488  2.0739   0.03809 *
mu1  1.35300    0.27035  5.0047 5.595e-07 ***
mu2  0.19521    0.12448  1.5682   0.11683
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

The estimates replicate the true parameters within the confidence intervals; however compared to the examples in Sect. 3.1, the standard errors are rather large (note also that the sample here includes 1,000 observations instead of mere 100 above). This is a common outcome while estimating mixture models.

Let us now replace the mixture by a pure normal sample

```
> xMix <- rnorm(1,000)
```

and estimate it using the same log-likelihood implementation:

```
> summary(m2 <- maxLik(logLikMix, start = c(rho = 0.5,
+     mu1 = 0, mu2 = 0.01)))

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 11 iterations
Return code 2: successive function values within tolerance limit
Log-Likelihood: -1413.934
3  free parameters
Estimates:
    Estimate Std. error t value Pr(> t)
rho 0.810615        Inf       0       1
mu1 0.018629        Inf       0       1
mu2 0.018628        Inf       0       1
--------------------------------------------
```

Although the estimates seem to be close to the correct point in the parameter space: mixture of 100% normal with mean 0 and 0% with mean 1, the Hessian matrix is singular and hence standard errors are infinite. This is because both components of the mixture converge to the same value and hence ρ is not identified. Hence we have no way establishing whether the common mean of the sample is, in fact, significantly different from 0. If the estimation is done by hand, it would be easy to treat ρ as fixed

as in the example in Sect. 3.3.1. However, this may not be a suitable approach if we want to run a large number of similar computations on different samples. In that case the user may want to consider signalling the `maxNR` routine that the parameters should be kept fixed. We may rewrite the function for calculating the log-likelihood value as follows:

```
> freePar <- rep(TRUE, 3)
> logLikMix1 <- function(param) {
+     rho <- param[1]
+     if (rho < 0 | rho > 1)
+         return(NA)
+     mu1 <- param[2]
+     mu2 <- param[3]
+     constPar <- NULL
+     if (freePar[1] & (abs(mu1 - mu2) < 0.001)) {
+         rho <- 1
+         constPar <- c(1, 3)
+         newVal <- c(1, 0)
+         fp <- freePar
+         fp[constPar] <- FALSE
+         assign("freePar", fp, inherits = TRUE)
+     }
+     ll <- log(rho * dnorm(xMix - mu1) + (1 - rho) * dnorm(xMix -
+         mu2))
+     if (!is.null(constPar)) {
+         attr(ll, "constPar") <- constPar
+         attr(ll, "newVal") <- list(index = constPar,
+             val = newVal)
+     }
+     ll
+ }
```

We have introduced three changes into the log-likelihood function.

- First, while changing the fixed parameters at run-time, we have to keep track of the process. This is why we introduce `freePar` *outside* the function itself, as it has to retain its value over successive calls to the function.
- The next novelty is related to checking the proximity to the region of trouble: `if(freePar[1] & (abs(mu1 - mu2) < 1e-3))`. Hence, if we have not set the first parameter (ρ) to a constant yet (this is what `freePar[1]` keeps track of), and the estimated means of the components are close to each other, we set ρ to 1. This means we assume the mixture contains only component 1. Note that because μ_2 is undefined as $\rho = 1$, we also have to keep that parameter fixed. We mark both of these parameters in the parameter vector to as fixed (`constPar <- c(1, 3)`), and provide the new values for them (`newVal <- c(1, 0)`).
- As the last step, we inform the `maxNR` algorithm of our decision by setting respective attributes to log-likelihood. Two attributes are used: `constPar` informs the algorithm that corresponding parameters in the parameter vector must be treated as constants from now on; and `newVal` (which contains two components – indices and values) informs which parameters have new values. It is possible to set

parameters to constants without changing the values by setting the `constPar` attribute only.

Now the estimation results look like:

```
> summary(m <- maxLik(logLikMix1, start = c(rho = 0.5,
+     mu1 = 0, mu2 = 0.01)))

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 12 iterations
Return code 1: gradient close to zero
Log-Likelihood: -1413.934
1  free parameters
Estimates:
    Estimate Std. error t value Pr(> t)
rho 1.000000   0.000000      NA      NA
mu1 0.018628   0.031623  0.5891  0.5558
mu2 0.000000   0.000000      NA      NA
--------------------------------------------
```

With parameters `rho` and `mu2` treated as constants, the resulting one-component model has small standard errors.

4 Summary and outlook

The **maxLik** package fills an existing gap in the R statistical environment and provides a convenient interface for maximum likelihood estimations—both for end users and package developers. Although R has included general-purpose optimizers and more specific Maximum Likelihood tools for a long time, the **maxLik** package has three important features that are not available in at least some of the alternatives: First, the package provides the Berndt-Hall-Hall-Hausman (BHHH) algorithm, a popular optimization method which is available only for likelihood-type problems. Second, the covariance matrix of the estimates can be calculated automatically. Third, the user can easily switch between different optimization algorithms.

In the future, we plan to add support for further optimization algorithms, e.g. function `nlm` of the built-in **stats** package that uses a Newton-type algorithm, the "L-BFGS-B" algorithm in function `optim` that allows for box constraints, function `nlminb` of the **stats** package that uses PORT routines (Gay 1990) and also allows for box constraints, function `ucminf` of the **ucminf** package (Nielsen and Mortensen 2009) that uses an improved quasi-Newton type algorithm, and function `DEoptim` of the **DEoptim** package (Ardia and Mullen 2010) that performs evolutionary global optimization via the differential evolution algorithm.

Another future extension includes a more comprehensive handling of constrained maximum likelihood problems.

We hope that these improvements will make the **maxLik** package even more attractive for users and package writers.

Acknowledgments The authors are grateful to two anonymous referees and to the participants of the sixth international workshop on Directions in Statistical Computing (DSC) in Copenhagen, Denmark, 13–14 July 2009 for giving them valuable comments and suggestions regarding the **maxLik** package and this paper. Arne Henningsen is grateful to the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) for financially supporting this research. Ott Toomet gratefully aknowledges financial support from Nordic Centre of Excellence in Empirical Labour Economics, and Estonian Ministry of Education and Research (Target Financing SF0180037s08). Of course, all errors are the sole responsibility of the authors.

References

Ardia D, Mullen K (2010) DEoptim: global optimization by differential evolution. http://CRAN.R-project.org/package=DEoptim. R package version 2.0–4

Bélisle CJP (1992) Convergence theorems for a class of simulated annealing algorithms on R^d. J Appl Probab 29:885–895

Berndt EK, Hall BH, Hall RE, Hausman JA (1974) Estimation and inference in nonlinear structural models. Annalysis Soc Meas 3(4):653–665

Bolker B (2009) bbmle: Tools for general maximum likelihood estimation. http://CRAN.R-project.org/package=bbmle. R package version 0.9.3

Broyden CG (1970) The convergence of a class of double-rank minimization algorithms. J Inst Math Appl 6:76–90

Calzolari G, Fiorentini G (1993) Alternative covariance estimators of the standard tobit model. Econ Lett 42(1):5–13

Carlevaro F, Croissant Y, Hoareau S (2010) mhurdle: estimation of models with limited dependent variables. R package version 0.1. http://CRAN.R-project.org/package=mhurdle

Croissant Y (2009) truncreg: truncated regression models. http://CRAN.R-project.org/package=truncreg. R package version 0.1

Croissant Y (2010) pglm: panel generalized linear model. R package version 0.1. http://CRAN.R-project.org/package=pglm

Fletcher R (1970) A new approach to variable metric algorithms. Comput J 13:317–322

Gay DM (1990) Usage summary for selected optimization routines. Computing science technical report 153, AT&T Bell Laboratories. http://netlib.bell-labs.com/cm/cs/cstr/153.pdf

Goldfarb D (1970) A family of variable metric updates derived by variational means. Math Comput 24:23–26

Greene WH (2008) Econometric analysis. 6. Prentice Hall, Englewood Cliffs

Henningsen A (2010) censReg: censored regression (Tobit) models. R package version 0.5. http://CRAN.R-project.org/package=censReg

Henningsen A, Toomet O (2010) miscTools: miscellanneous small tools and utilities. R package version 0.6. http://cran.r-project.org/package=miscTools

Nelder JA, Mead R (1965) A simplex algorithm for function minimization. Comput J 7:308–313

Nielsen HB, Mortensen SB (2009) ucminf: general-purpose unconstrained non-linear optimization. http://CRAN.R-project.org/package=ucminf. R package version 1.0

R Development Core Team (2009) R: A language and environment for statistical computing. R foundation for statistical computing. Vienna, Austria. http://www.R-project.org, ISBN 3-900051-07-0

Sevcikova H, Raftery A (2010) mlogitBMA: Bayesian Model Averaging for Multinomial Logit Models. http://CRAN.R-project.org/package=mlogitBMA. R package version 0.1-2

Shanno DF (1970) Conditioning of quasi-newton methods for function minimization. Math Comput 24:647–656

Toomet O, Henningsen A (2008) Sample selection models in R: package sampleSelection. J Stat Softw 27(7):1–23. http://www.jstatsoft.org/v27/i07/

Toomet O, Henningsen A (2010) maxLik: tools for maximum likelihood estimation. R package version 1.0. http://CRAN.R-project.org/package=maxLik