Computational Statistics manuscript No.
(will be inserted by the editor)

maxLik: A Package for Maximum Likelihood Estimation in R

Arne Henningsen · Ott Toomet

Received: date / Accepted: date

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

Keywords Maximum Likelihood · Optimisation

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

Arne Henningsen
Institute of Food and Resource Economics, University of Copenhagen
Rolighedsvej 25, 1958 Frederiksberg C, Denmark
Tel.: +45-353-32274
E-mail: arne@foi.dk

Ott Toomet
Department of Economics, University of Tartu (Estonia)
Department of Economics, Aarhus School of Business, University of Aarhus (Denmark)

Although *R* (R Development Core Team 2009) has built-in optimization algorithms since it's early days, a comprehensive environment for ML estimation has been missing until recently. The package **maxLik** (Toomet and Henningsen 2009) is intended to fill this gap. 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 **LambertW** (Goerg 2009), **mlogit** (Croissant 2008), **sampleSelection** (Toomet and Henningsen 2008), and **truncreg** (Croissant 2009) use the **maxLik** package for their maximum likelihood estimations.

The **maxLik** package 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 numerical 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 Section 4 concludes.

2 Implementation

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 `maxNR`. This is used as a basis of `maxBHHH`, a Berndt-Hall-Hall-Hausman type optimizer (Berndt et al 1974), popular for ML problems. In addition, various methods from the `optim` function (from package **stats**) are included by the package in a unified way (functions `maxBFGS`, `maxNM`, and `maxSANN`).

The 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 information 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 sequential unconstrained maximization technique (SUMT), implemented in the package. 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 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.

3 Using the maxLik package

3.1 Basic usage

As the other packages in *R*, 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`. It is mostly a wrapper for different optimization routines with a few additional features, useful for ML estimations. This function has two mandatory arguments

```
> maxLik(logLik, 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 fit a normal distribution by maximum likelihood. First, we generate a vector (x) of 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 probability density function of a standard normal distribution for a value x_i is

$$P(x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right). \quad (1)$$

Hence, the likelihood function for μ, σ, and the values in vector $x = (x_1, \ldots, x_N)$ is

$$L(x; \mu, \sigma) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right) \tag{2}$$

and its logarithm (i.e. the log-likelihood function) is

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

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

```
> 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. May be a solution
Log-Likelihood: -201.5839
2  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18081    0.18151  6.5057 7.735e-11 ***
sigma  1.81648    0.12840 14.1466 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

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

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

```
> coef(mle)

      mu    sigma
1.180812 1.816481

> stdEr(mle)

       mu     sigma
0.1815053 0.1284037
```

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.[2]

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

[1] TRUE
```

If no analytical gradients are provided by the user, numerical gradients and numerical Hessians are calculated by the functions `numericGradient` and `numericNHessian`, which are also included in the **maxLik** package. While the maximisation of the likelihood function of this simple model works well with numerical gradients and Hessians, this may not be the case for more complex models. Numerical derivatives may be costly to compute, and, even more, they may turn out to be noisy and unreliable. In this way numerical 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, which is not based on gradients.

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

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

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

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)
```

[2] 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 1.5e-08).

```
+     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))
> summary(mleGrad)

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 7 iterations
Return code 1: gradient close to zero. May be a solution
Log-Likelihood: -201.5839
2  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18081    0.18165  6.5005 8.003e-11 ***
sigma  1.81648    0.12844 14.1421 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
--------------------------------------------

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

[1] TRUE

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

[1] TRUE

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

[1] "Mean relative difference: 0.001259534"
```

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

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 numerically by the function `numericHessian`.[3] The elements of the Hessian matrix of the log-

[3] Only the Newton-Raphson algorithm uses the Hessian directly for obtaining the estimates. The other algorithms ignore it during the estimation but use it for computing the final variance-covariance matrix.

likelihood function for the normal distribution are

$$\frac{\partial^2 \log(L(x,\mu,\sigma))}{(\partial\mu)^2} = -\frac{N}{\sigma^2} \tag{6}$$

$$\frac{\partial^2 \log(L(x,\mu,\sigma))}{\partial\mu\,\partial\sigma} = -2\sum_{i=1}^{N}\frac{(x_i-\mu)}{\sigma^3} \tag{7}$$

$$\frac{\partial^2 \log(L(x,\mu,\sigma))}{(\partial\sigma)^2} = \frac{N}{\sigma^2} - 3\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^4}. \tag{8}$$

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))
> summary(mleHess)

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 7 iterations
Return code 1: gradient close to zero. May be a solution
Log-Likelihood: -201.5839
2  free parameters
Estimates:
      Estimate Std. error t value   Pr(> t)
mu     1.18081    0.18165  6.5005 8.003e-11 ***
sigma  1.81648    0.12844 14.1421 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 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 numerical gradients, numerical Hessians may turn out to be slow and unreliable.

3.2 Optimisation Methods

The `maxLik` function allows to select between 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 (numerical or analytical) gradients and Hessians; the BHHH and BFGS algorithms use only (numerical 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 order to make the switching the algorithms easier, but they are ignored if the selected method does not require this information, for instance, the argument `hess` for the Berndt-Hall-Hall-Hausman method.[4] In this way the user can easily change the optimisation method without changing the arguments.

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 BHHH method approximates the Hessian by the negative of the expectation of the outer product of the gradient (score) vector. In order to calculate the expectation, the algorithm needs gradient vectors by individual observations. This can be achieved either by providing a corresponding gradient function (see below) or 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, numeric 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]
```

[4] Even if the optimization method itself does not make use of the Hessian, this information is used (except for `"BHHH"`) for computing the final variance-covariance matrix.

```
+     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
BHHH maximisation, 13 iterations
Return code 2: successive function values within tolerance limit.
May be a solution
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: 2.579856e-07"

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

[1] "Mean relative difference: 0.0711091"
```

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 on 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 numerical gradients and analytical gradients are virtually identical.

3.2.2 Nelder-Mead (NM) and other methods

The other maximization methods: Nelder-Mead, Broyden-Fletcher-Goldfarb-Shannon, 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 Gradient matrix with gradients of individual observations, is accepted as well. If the user does not provide gradients, the gradients are computed numerically.

To give the reader a taste, we give an example using gradientless 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    0.18159  6.5015 7.953e-11 ***
sigma  1.81664    0.12843 14.1445 < 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: 0.001103636"
```

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

`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 fix certain parameters, we would like to handle as constants in the optimization process. This feature is implemented in `maxNR` (and hence supported by `maxBHHH` as well), which is a part of **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 fixed at 2. This is done via argument `activePar` of `maxNR`. It is a logical vector of length equal to that of the parameter vector which specifies which components are allowed to change (are "active"). The other, "non-active" parameters are treated as known constants, always retaining their initial value. So, as σ was the second parameter, we should call:

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

```
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 3 iterations
Return code 1: gradient close to zero. May be a solution
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, ML estimate of it is still the sample average) while the estimated standard error is different. Obviously, log-likelihood is lower in the constrained space, although the reader may verify that allowing σ to be free is an insignificant improvement according to the likelihood ratio test.

3.3.2 Automatic fixing of parameter values

Next, we demonstrate, how it is possible to fix a parameter automatically during the computations when using `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. We demonstrate this feature by estimating the parameters of a mixture model of two normal distributions on a sample, drawn from a single normal distribution. Note that this example is highly dependent on the initialization of random number generator and 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 a mixture of normals:

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

Hence `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

$$l = \sum_{i=1}^{N} \log(\rho\phi(x_i - \mu_1) + (1-\rho)\phi(x_i - \mu_2)), \tag{9}$$

where ρ is the percentage of the first component in the mixture and $\phi(\cdot)$ is the standard normal density function. 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 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. May be a solution
Log-Likelihood: -1536.981
3  free parameters
Estimates:
    Estimate Std. error t value   Pr(> t)
rho  0.27973    0.13546  2.0651   0.03892 *
mu1  1.35300    0.27165  4.9806 6.338e-07 ***
mu2  0.19521    0.12484  1.5637   0.11790
---
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 Section 3.1, the standard errors are rather large (note that the sample here includes 1000 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(1000)
```

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.
May be a solution
Log-Likelihood: -1413.934
3  free parameters
Estimates:
    Estimate Std. error t value Pr(> t)
rho 0.852180        Inf       0       1
mu1 0.018628        Inf       0       1
mu2 0.018630        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 fix ρ as in the example above. 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 treated as fixed. We may rewrite the log-likelihood function 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.
- Next novelty is related to checking the proximity to the region of trouble: `if(freePar[1] & (abs(mu1 - mu2) < 1e-3))`. Hence, if we haven't fixed the first parameter (ρ) 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 fix that parameter. We mark both of these parameters in the parameter vector to be fixed (`constPar <- c(1, 3)`), and provide the new values for them (`newVal <- c(1, 0)`).
- As the last step, we inform `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 fixed from now on; and `newVal` (which contains two components – indices and values) informs which parameters have new values. It is possible to fix parameters 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 = 0.1)))

       rho        mu1        mu2
1.00000000 0.01840651 0.00000000
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 11 iterations
Return code 1: gradient close to zero. May be a solution
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   0.031627   0.589  0.5559
mu2 0.000000   0.000000      NA      NA
--------------------------------------------
```

With parameters `rho` and `mu2` fixed, 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* includes general-purpose optimizers such as `nlm` and `optim` for a long time, the **maxLik** package has three important features that are not available in at least some of the alternatives: First, the covariance matrix of the estimates can be calculated automatically. Second, the user can easily switch between different optimization algorithms. Third, the package provides the Berndt-Hall-Hall-Hausman (BHHH) algorithm, a popular optimization method which is available only for likelihood-type problems.

In the future, we plan to add support for further optimisation algorithms, e.g. function `nlm`, the "L-BFGS-B" algorithm in function `optim` that allows for box constraints, function `nlminb` that uses PORT routines (Gay 1990) and also allows for box constraints, or function `ucminf` of the **ucminf** package (Nielsen and Mortensen 2009). Another future extension includes 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.

References

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

Berndt EK, Hall BH, Hall RE, Hausman JA (1974) Estimation and inference in nonlinear structural models. Annals of Economic and Social Measurement 3(4):653–665

Broyden CG (1970) The convergence of a class of double-rank minimization algorithms. Journal of the Institute of Mathematics and Its Applications 6:76–90

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

Croissant Y (2008) mlogit: Multinomial Logit Model. URL `http://CRAN.R-project.org/package=mlogit`, R package version 0.1

Croissant Y (2009) truncreg: Truncated Regression Models. URL `http://CRAN.R-project.org/package=truncreg`, R package version 0.1

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

Gay DM (1990) Usage summary for selected optimization routines. Computing Science Technical Report 153, AT&T Bell Laboratories

Goerg GM (2009) LambertW: Lambert W Parameter Estimation, Plots, Simulation (Skew Analysis). URL `http://CRAN.R-project.org/package=LambertW`, R package version 0.1.6

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

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

Nielsen HB, Mortensen SB (2009) ucminf: General-purpose unconstrained non-linear optimization. URL `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, URL `http://www.R-project.org`, ISBN 3-900051-07-0

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

Toomet O, Henningsen A (2008) Sample selection models in R: Package sampleSelection. Journal of Statistical Software 27(7):1–23, URL http://www.jstatsoft.org/v27/i07/

Toomet O, Henningsen A (2009) maxLik: Tools for Maximum Likelihood Estimation. R package version 0.5, http://CRAN.R-project.org