Lecture 3: Application of Optimization Algorithms to Maximum Likelihood Estimator

Professor: Mauricio Sarrias

Universidad de Talca

2020

Goals

- To understand most of the functionalities of **maxLik** package in R.
- To be able to program a MLE in R using **maxLik**.
- To be able to estimate a linear model by MLE.

maxLik Package in R

- **maxLik** is designed to provide a single, unified interface for different optimization routines, and to treat the results in a way suitable for ML estimation.
- It implements NR, BHHH, BFGS and other optimization routines.
- Our only job is to code a function for the log-likelihood function (gradient and Hessian if we want), and **maxLik** will do the optimization for us.

Basic Usage

- The following command loads the **maxLik** package.

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

- `maxLik` function has two mandatory arguments,
  - `logLik`: a function that calculates the log-likelihood value as a function of the parameters,
  - `start` : vector of starting values
- I will demonstrate the usage of `maxLik` package by estimating and linear regression.
- Recall that the conditional log-likelihood function of $y_i|\mathbf{x}_i \sim \mathrm{N}(\mathbf{x}'\boldsymbol{\beta}_0, \sigma_0^2)$ is

$$\begin{aligned}\log f(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) &= -0.5 \cdot n \cdot \log(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i - \mathbf{x}_i^\intercal\boldsymbol{\beta})^2}{2\sigma^2} \\ &= -0.5 \cdot n \cdot \log(2\pi\sigma^2) - \frac{(\mathbf{y} - \mathbf{X}\widehat{\boldsymbol{\beta}})^\intercal(\mathbf{y} - \mathbf{X}\widehat{\boldsymbol{\beta}})}{2\sigma^2}\end{aligned}$$

Basic Usage

Given the LL function above, we create and R function that calculates the log-likelihood value

```r
linear.ml <- function(param, y, X){
  n <- nrow(X)
  k <- ncol(X)
  beta <- param[1:k]
  sig.sq  <- param[k + 1]
  resi  <- y - tcrossprod(X, t(beta))
  LL    <- -0.5 * n * log(2 * pi * sig.sq) - 0.5 * crossprod(resi) / sig.sq
  return(LL)
}
```

Note the first argument must be the vector of parameters to be estimated and it must return the log-likelihood value.[1] The second element `y` is the dependent variable, whereas the element `X` is the matrix $n \times K$ of independent variables.

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

Generate DGP

I first generate an artificial data set. The data generating process is the following:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \epsilon_i$$

where $\beta_0 = \beta_1 = \beta_2 = 1$ and $\epsilon_i \sim \mathrm{N}(0, 4)$. I also assume that $x_{1i} \sim U[0, 1]$ and $x_{2i} \sim \mathrm{N}(0, 1)$.

```
# Create artifitial database
set.seed(123)
N  <- 5000      # number of individuals
x1 <- runif(N)
x2 <- rnorm(N)
y <- 1 + 1 * x1 + 1 * x2 + rnorm(N, mean = 0, sd = 2) # e~N(0, 2^2)
X <- cbind(1, x1, x2) # put data in matrix form
```

Since the optimization procedure is numerical, we need the vector of starting values. My first initial guess is the following:

```
start <- c(0, 0, 0, 1)
names(start) <- c("constant", "x1", "x2", "sigma2")
```

Syntax

The syntax for the estimation using `maxLik` is the following:

```
# Estimate the model by BFGS
out1 <- maxLik(logLik = linear.ml, start = start,
               y = y, X = X, method = 'bfgs')
```

Note that we need also to include the arguments for our `linear.ml` function.

Output

The results are the following

```
summary(out1)

## --------------------------------------------
## Maximum Likelihood estimation
## BFGS maximization, 61 iterations
## Return code 0: successful convergence
## Log-Likelihood: -10611.68
## 4  free parameters
## Estimates:
##          Estimate Std. error t value Pr(> t)
## constant  0.98958    0.05693  17.383  <2e-16 ***
## x1        0.99172    0.10159   9.762  <2e-16 ***
## x2        0.99982    0.02897  34.518  <2e-16 ***
## sigma2    4.08285    0.08139  50.167  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

Note that the estimated parameters are very close to the true parameters.

Question

How does `maxLik` compute the gradient?

Other Functions

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

```
coef(out1)

##  constant        x1        x2    sigma2
## 0.9895816 0.9917197 0.9998217 4.0828547

stdEr(out1)

##   constant         x1         x2     sigma2
## 0.05692898 0.10158965 0.02896515 0.08138603
```

Computing variance-covariance matrix

Recall that we can estimate the variance-covariance matrix as:

$$\widehat{\mathbf{V}}^1 = \left(\frac{1}{n} \sum_{i=1}^{n} -\mathbf{H}(\mathbf{w}_i, \widehat{\boldsymbol{\theta}}) \right)^{-1} = -\left(\mathbf{H}(\mathbf{w}, \widehat{\boldsymbol{\theta}}) \right)^{-1}$$

Then,

```
avov <- -solve(out1$hessian) #extract the Hessian at theta hat
se   <- sqrt(diag(avov))
se

##   constant          x1          x2      sigma2
## 0.05692898 0.10158965 0.02896515 0.08138603

all.equal(se, stdEr(out1))

## [1] TRUE
```

Some Notions

- If no analytical gradient is provided, finite-difference gradient and Hessian are calculated.
- While the estimation works well in this case, this may not be the case for more complex models.
  - Finite-difference derivatives may be costly to compute, and they may turn out to be noisy and unreliable.
  - They might either slow down the estimation or even impede convergence.
- It is highly 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.

Gradient

Recall that the score function for each individual is:

$$\mathbf{s}(\mathbf{w}_i;\boldsymbol{\theta}) = \begin{pmatrix} \frac{1}{\sigma^2}\mathbf{x}_i \cdot \widehat{\epsilon}_i \\ -\frac{1}{2\sigma^2} + \frac{1}{2\sigma^4}\widehat{\epsilon}_i^2 \end{pmatrix}$$

whereas the gradient is given by:

$$\mathbf{s}(\mathbf{w};\boldsymbol{\theta}) = \begin{pmatrix} \underbrace{\frac{1}{n\sigma^2}\mathbf{X}^\intercal\widehat{\boldsymbol{\varepsilon}}}_{(K\times 1)} \\ \underbrace{-\frac{1}{2\sigma^2} + \frac{1}{2n\sigma^4}\widehat{\boldsymbol{\varepsilon}}^\intercal\widehat{\boldsymbol{\varepsilon}}}_{(1\times 1)} \end{pmatrix}$$

BHHH

- BHHH 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.
- This approximation is only valid while maximizing log-likelihood.
- In order to use the BHHH method, the user has to provide the gradient vectors by individual observations (so, a matrix $n \times K$)

Example

The gradient is included in the following way:

```
linear.mlb <- function(param, y, X){
  k <- ncol(X)
  beta <- param[1:k]
  sig.sq  <- param[k + 1]
  resi  <- y - tcrossprod(X, t(beta))
  # Log-likelihood function for each individual
  Li    <-   -0.5 * log(2 * pi * sig.sq) - 0.5 * resi ^ 2 / sig.sq

  #Score function for each individual
  grad.beta <- (1 / sig.sq) * X * drop(resi)
  grad.sig  <- -1/(2 * sig.sq) + resi ^ 2 / (2 * sig.sq ^ 2)
  grad <- cbind(grad.beta, grad.sig)
  attr(Li, 'gradient') <- grad
  Li
}
```

Example

Estimate the model using BHHH procedure

```
out2 <- maxLik(logLik = linear.mlb, start = start,
               y = y, X = X, method = 'bhhh')
summary(out2)

## --------------------------------------------
## Maximum Likelihood estimation
## BHHH maximisation, 17 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -10611.68
## 4  free parameters
## Estimates:
##          Estimate Std. error t value Pr(> t)
## constant  0.98958    0.05828  16.981  <2e-16 ***
## x1        0.99172    0.09958   9.959  <2e-16 ***
## x2        0.99982    0.02869  34.853  <2e-16 ***
## sigma2    4.08285    0.08165  50.004  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

Check

```
all.equal(coef(out1), coef(out2))

## [1] "Mean relative difference: 1.1837e-06"

all.equal(stdEr(out1), stdEr(out2))

## [1] "Mean relative difference: 0.01452157"
```

We observe very small differences in the coefficients, but they standard errors are slightly different. This is because the outer product approximation may differ from the derivative-based Hessian in finite samples.

Variance-Covariance Matrix

Recall that:

$$\widehat{\mathbf{V}}^2 = \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{s}(\mathbf{w}_i, \widehat{\boldsymbol{\theta}}) \mathbf{s}(\mathbf{w}_i; \widehat{\boldsymbol{\theta}})^\intercal \right)^{-1} = (\mathbf{G}^\intercal \mathbf{G})^{-1}$$

where $\mathbf{G}$ is a $n \times K$ matrix of score vectors for each individual.

```
G <- out2$gradientObs
GG <- crossprod(G)
se_bhhh <- sqrt(diag(solve(GG)))
se_bhhh

##   constant          x1          x2      sigma2
## 0.05827650 0.09957628 0.02868666 0.08165106

all.equal(se_bhhh, stdEr(out2))

## [1] TRUE
```

Hessian

Recall that the Hessian is:

$$\mathbf{H}(\mathbf{w}_i;\boldsymbol{\theta}) = \begin{pmatrix} -\frac{1}{\sigma^2}\mathbf{x}_i\mathbf{x}_i^\intercal & -\frac{1}{\sigma^4}\mathbf{x}_i \cdot \widehat{\epsilon}_i \\ -\frac{1}{\sigma^4}\mathbf{x}_i^\intercal \cdot \widehat{\epsilon}_i & \frac{1}{2\sigma^4} - \frac{1}{\sigma^6}\widehat{\epsilon}_i^2 \end{pmatrix} \tag{1}$$

or

$$\mathbf{H}(\mathbf{w};\boldsymbol{\theta}) = \sum_{i=1}^{n} \mathbf{H}(\mathbf{w}_i;\boldsymbol{\theta}) = \begin{pmatrix} -\frac{1}{\sigma^2}\mathbf{X}^\intercal\mathbf{X} & -\frac{1}{\sigma^4}\mathbf{X}^\intercal\widehat{\boldsymbol{\varepsilon}} \\ -\frac{1}{\sigma^4}\widehat{\boldsymbol{\varepsilon}}^\intercal\mathbf{X} & \frac{1}{2\sigma^4} - \frac{1}{\sigma^6}\widehat{\boldsymbol{\varepsilon}}^\intercal\widehat{\boldsymbol{\varepsilon}} \end{pmatrix} \tag{2}$$

Code

```r
linear.mlc <- function(param, y, X){
  k <- ncol(X)
  beta <- param[1:k]
  sig.sq  <- param[k + 1]
  if (sig.sq <= 0) return(NA) # This signals to the optimazer
  # that the attempted parameter value was out of range, and forces
  # it to find a new one closer to the previous values
  resi  <- y - tcrossprod(X, t(beta))
  Li    <-   -0.5 * log(2 * pi * sig.sq) - 0.5 * resi ^ 2 / sig.sq

  #Gradient
  grad.beta <- (1 / sig.sq) * X * drop(resi)
  grad.sig  <- -1/(2 * sig.sq) + resi ^ 2 / (2 * sig.sq ^ 2)
  grad <- cbind(grad.beta, grad.sig)
  attr(Li, 'gradient') <- grad


  #Hessian
  H <- matrix(NA, nrow = k + 1, ncol = k + 1)
  H[1:k, 1:k] <- -(1 / (sig.sq)) * crossprod(X)
  H[k + 1, k + 1] <- 1 / (2 * sig.sq ^ 2) - (1 / (sig.sq ^ 3)) * crossprod(resi)
  H[1:k, k + 1] <- -(1 / (sig.sq ^ 2))  * crossprod(X, resi)
  H[k + 1, 1:k] <- -(1 / (sig.sq ^ 2))  * crossprod(resi, X)
  attr(Li, 'hessian') <- H
  Li
}
```

Code

Now we estimate the model by NR:

```
out3 <- maxLik(logLik = linear.mlc, start = start,
               y = y, X = X, method = 'nr')
summary(out3)

## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 17 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -10611.68
## 4  free parameters
## Estimates:
##          Estimate Std. error t value Pr(> t)
## constant  0.98958    0.05699   17.36  <2e-16 ***
## x1        0.99172    0.09913   10.00  <2e-16 ***
## x2        0.99982    0.02877   34.76  <2e-16 ***
## sigma2    4.08222    0.05773   70.71  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

OLS

Now we compare with OLS

```
colnames(X) <- c("constant", "x1", "x2")
ols <- lm(y ~ X - 1)
summary(ols)
```

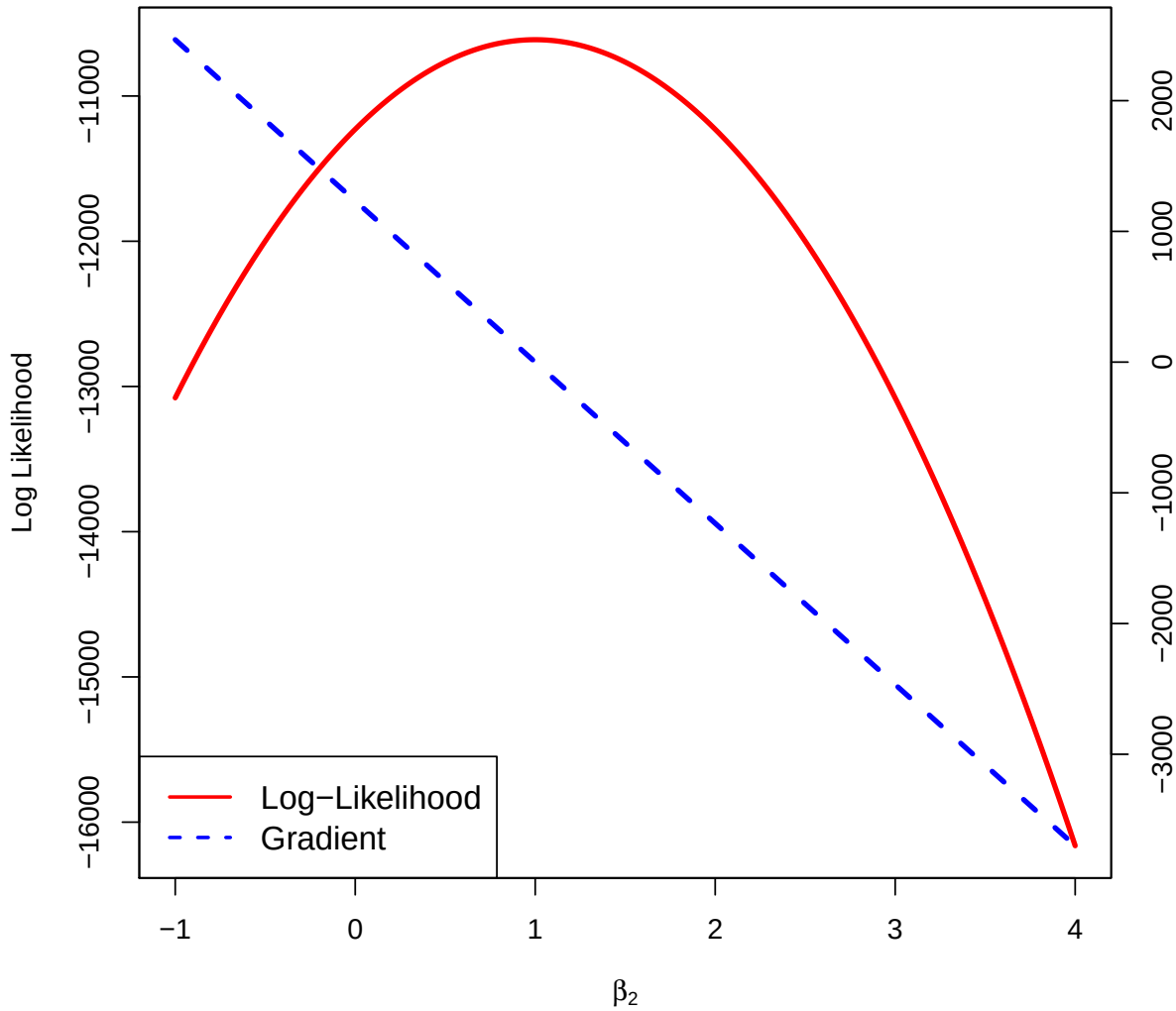
```
##
## Call:
## lm(formula = y ~ X - 1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6783 -1.3491 -0.0285  1.3982  7.7129
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## Xconstant  0.98958    0.05702   17.36   <2e-16 ***
## Xx1        0.99172    0.09917   10.00   <2e-16 ***
## Xx2        0.99982    0.02878   34.74   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.021 on 4997 degrees of freedom
## Multiple R-squared:  0.4442,Adjusted R-squared:  0.4439
## F-statistic:  1331 on 3 and 4997 DF,  p-value: < 2.2e-16
```

Example

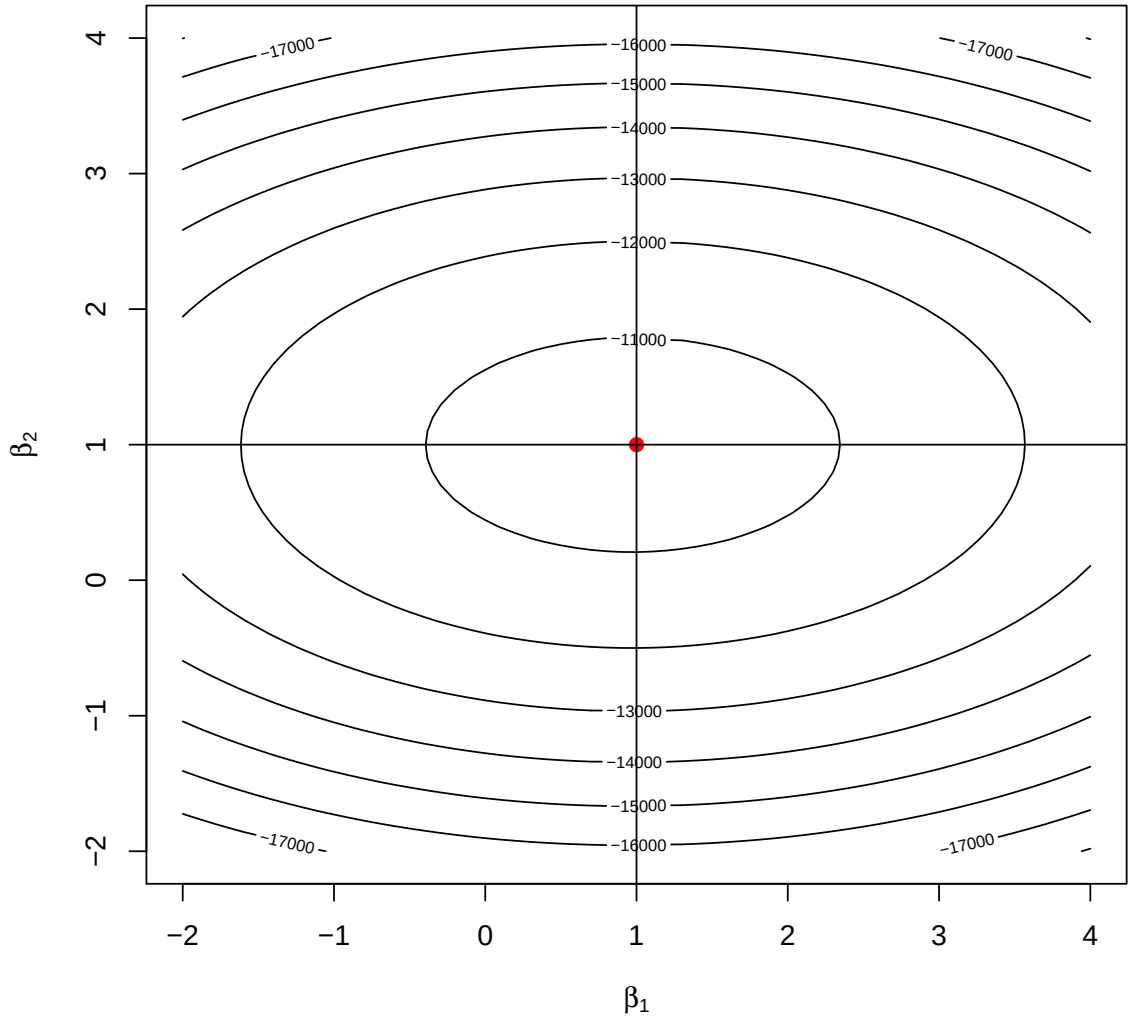
```
delta <- seq(-1, 4, .01)
grad.values <- as.numeric(lapply(delta, function(x) {
                            colSums(attr(linear.mlb(param <- c(1, 1, x, 4),
                                                    y = y,
                                                    X = X),
                                         'gradient')
                                    )[[3]]
                                      }
                                )
                            )

logl.values <- as.numeric(lapply(delta, function(x) {
   sum(linear.mlb(param <- c(1, 1, x, 4),y = y,X = X))
}))
```

Example


Log Likelihood
−11000
−12000
−13000
−14000
−15000
−16000
2000
1000
0
−1000
−2000
−3000
−1
0
1
2
3
4
β2
Log−Likelihood
Gradient

Example


−17000
−16000
−15000
−14000
−13000
−12000
−11000
−13000
−14000
−15000
−16000
−17000
−17000
−17000
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4
β2
β1

Example