A PRIMER OF MAXIMUM LIKELIHOOD PROGRAMMING IN R

Marco R. Steenbergen

7/28/2019

Abstract

R is an outstanding platform for maximum likelihood programming. These notes describe the **maxLik** package, a wrapper that gives access to the most important hill-climbing algorithms and provides a convenient way of displaying results. The notes also include a discussion of Wald confidence intervals and likelihood ratio tests.

Important: *The functions in these notes were tested on R version 3.6.1 (2019-07-05). Use of these notes is at your own risk. Correspondence should be directed to steenbergen@ipz.uzh.ch.*

Introduction

R is by now one of the most widely used programming languages in the social and behavioral sciences. A major reason is that R is a flexible and versatile language, which makes it easy to program new models. R is also very precise in its computations.

R is well-suited for programming your own maximum likelihood (ML) routines. Indeed, there are several procedures for optimizing likelihood functions. Here I shall focus on the **maxLik** package, which implements a variety of algorithms and offers a convenient way of displaying results (Henningsen and Toomet (2011)). Among the implemented hill-climbing algorithms are Newton-Raphson, BFGS, and BHHH. The package is capable of numeric differentiation and in most of our examples we shall take advantage of this feature.

Syntactic Structure

Evaluating likelihood functions entails a two-step process. First, one declares the log-likelihood function and next one optimizes this function. The log-likelihood function and optimization command may be typed interactively into the command window or they may be contained in a .R syntax file. I suggest saving log-likelihood functions into a file, especially if you plan on using them frequently. They can then be accessed using the source function.

Declaring the Log-Likelihood Function

The log-likelihood function is declared as a normal function. In R, functions take at least one argument, to wit a parameter or a vector of parameters. It is generally useful, however, also to declare a data frame or matrix. This serves as a placeholder for the actual data that will be analyzed.

The generic syntax takes the following form:

```
name <- function(pars, df) {
  declarations
}
```

Here **pars** is the name of the parameter (vector), while **df** is the name of the data frame or matrix. (In some cases, multiple data objects may need to be referenced.) The **declarations** section specifies the log-likelihood function. Depending on the needs, it may also spell out the contents of the parameter vector. The precise specification of the log-likelihood function depends on the algorithm that is being used: it is different for BHHH than for other algorithms.

Single Parameter

When the log-likelihood function includes only a single parameter and numerical derivatives are used in the estimation, then the **declarations** typically involve a single line of code. What this line looks like depends on the algorithm. For the BHHH algorithm, the log-likelihood is specified in terms of a single observation.

Example 1: *Consider the Poisson probability mass function with unit log-likelihood values of*

$$\ell_i = -\mu + y_i \ln \mu - \ln(y_i!)$$

This may be programmed in nthe following way:

```
lnl <- function(mu) {
  -1*mu + y*log(mu) - lfactorial(y)
}
```

Here **lfactorial(y)** *is* $\ln(y_i!)$*. Since the declaration of the function consists of only one line, we could have omitted the brackets. Notice that we did not make the data an argument of the function. The reason is that we do not perform any operations on the data inside the function.*

For other algorithms such as BFGS and Newton-Raphson, one needs to specify the full log-likelihood function, i.e., $\ell = \sum_{i=1}^{n} \ell_i$. This is illustrated in the following example:

Example 2: *For the Poisson distribution, the full log-likelihood is given by*

$$\ell = -n\mu + \ln \mu \sum_{i=1}^{n} y_i - \sum_{i=1}^{2} \ln(y_i!)$$

Assuming the data are stored in the dataframe **y**, *we can program the log-likelihood as*

```
lnl <- function(mu, y) {
  n <- nrow(y)
  -n*mu + log(mu)*sum(y) - sum(lfactorial(y))
}
```

Here **sum** *is the sum operator. Because the declaration of the Notice that we now include the data as an argument because we compute* n *based on the number of rows in the data frame. (If* **y** *is a vector, we should replace* **nrow** *with* **length**.*)*

Multiple Parameters

When the log-likelihood contains multipke parameters, then the R commands change only in the sense that **pars** is now a vector whose contents need to be

specified. It is generally quite easy to make this modification, as we shall illustrate for the normal distribution.

Example 3: *Imagine, $Y \sim \mathcal{N}(\mu, \sigma)$. We seek to estimate μ and σ using the Newton-Raphson algorithm. The kernel of the full log-likelihood is*

$$\ell = -n \ln \sigma - .5 \sum_{i=1}^{n} \left\{ \frac{(y_i - \mu)^2}{\sigma^2} \right\}$$

The log-likelihood function is now specified as

```
lnl <- function(pars, y) {
  mu <- pars[1]
  sigma <- pars[2]
  n <- nrow(y)
  z <- (y-mu)/sigma
  -n*log(sigma) -.5*sum(z^2)
}
```

We have now declared pars to be a vector with two elements, the first of which is μ and the second of which is σ. We again define n. We also simplify things a bit by declaring the z-transform $(y - \mu)/\sigma$; this allows us to write the log-likelihood compactly.

Optimization

Once the log-likelihood has been declared, the **maxLik** routine is invoked to perform optimization. The generic syntax is:

```
library(maxLik)
object <- maxLik(loglikelihood,
                 grad = NULL,
                 hess = NULL,
                 method = c("BFGS", "BFGSR", "BHHH", "CG",
                            "NM", "NR", "SANN"),
                 start =,
                 dataframe)
```

Log-Likelihood, Gradient, and Hessian

The argument **loglikelihood** is mandatory and is the function that you specified earlier. When using numerical derivatives, the arguments **grad** and **hess** for gradient and Hessian can both be set equal to NULL. Since this is the default, we could also have skipped the arguments altogether. Later, we shall explore examples in which we forego numeric differentiation. In those cases, we have to declare additional functions for the gradient and the Hessian.

Method

The argument **method** declares the optimizer that should be used. The current choices are:

- BFGS for Broyden-Fletcher-Goldfarb-Shanno
- BFGSR for the R implementation of Broyden-Fletcher-Goldfarb-Shanno
- BHHH for Berndt-Hall-Hall-Hausman
- CG for conjugate gradients
- NM for Nelder- Mead
- NR for Newton-Raphson
- SANN for simulated annealing

Starting Values

The argument **start** declares the starting value(s). When a single parameter is estimated, the declaration of the starting values is extremely easy:

```
start = #
```

Here, $\#$ is a particular value of the parameter. This should be chosen according to the range restrictions for the probability density or mass function on which the log-likelihood is based. For instance, in the Poisson distribution μ has to be strictly positive. Thus, a starting value of 1 is OK but one of -1 will surely cause a warning message and a convergence problem.

When there are multiple parameters, then a vector of starting values should be declared:

```
start = c(#1, #2, ...)
```

Here, $\#1$, $\#2$, etc. are successive starting values, which should be specified in the same order as the parameters and should meet the range restrictions.

It is possible to use previous estimation results as starting values, as we shall see in the worked examples later on.

Data Frame

The data frame specifies the data frame or matrix containing the data. This will be passed onto the log-likelihood function. Here it is important that the data are consistent with the support for the probability density or mass function. For example, in a Poisson regression the data points should be non-negative integers. It is thus important to check the data distribution prior to performing the estimation.

Output

maxLik allows us to display the estimates, the value of the log-likelihood, the number of estimated parameters, the variance-covariance matrix of the estimators (VCE), and messages concerning convergence. The estimates can be displayed using

```
summary(object)
```

where **object** is the estimation object that was created. The **summary** command shows the optimization routine, indicates whether the routine exited normally, gives the value of the log-likelihood function at the maximum, and shows the parameter estimates, their standard errors, t-statistics, and p-values (under the null hypothesis that the parameter is 0). We illustrate the output for the Poisson distribution.

Example 4: *We start by generating some simulated data from a $\mathcal{P}(2)$ distribution.*

```
set.seed(2310, kind = "Mersenne-Twister",
         normal.kind = "Inversion")
y <- rpois(1000, 2)
summary(y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   1.000   2.000   2.001   3.000   7.000
```

We shall employ the Newton-Raphson algorithm, so that we use the full log-likelihood function from example 2. We use a starting value of 1.

```
poisson.fit <- maxLik(lnl, grad = NULL, hess = NULL,
                      method = "NR", start = 1, y = y)
```

We obtain the following output.

```
summary(poisson.fit)
```

```
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 5 iterations
Return code 2: successive function values within tolerance limit
Log-Likelihood: -33887.01
1  free parameters
Estimates:
     Estimate Std. error t value Pr(> t)
[1,]  2.00100    0.04463   44.84  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Individual elements of the output can be obtained as follows:

```
# Log-Likelihood and Degrees of Freedom
logLik(object)
# Estimates
coef(object)
# VCE
vcov(object)
# Number of Estimated Parameters
nParam(object, free = TRUE)
# Return Code from optimizer
returnCode(object)
# Return Message from Optimizer
returnMessage(object)
```

Regarding the return codes, the following numeric values indicate convergence of the algorithm:

Algorithm	Code
BFGS	0
BFGSR	2
BHHH	2
CG	0
NM	0
NR	2
SANN	0

Fine-Tuning the Estimation

Providing Analytic Derivatives

In some cases, numeric derivatives can be unreliable or time consuming to compute and it may pay off to provide analytic derivatives instead. One could opt to provide the gradient, the Hessian, or both. The example below illustrates this for the normal distribution.

Example 5: *We decide to use the NR algorithm and thus only need to specify ℓ and its derivatives. Using the z-transform $(y_i - \mu)/\sigma$, the log-likelihood function is given by*

$$\ell = -n \ln \sigma - .5 \sum_{i=1}^{n} z_i^2$$

The gradient vector is

$$\boldsymbol{\nabla} = \begin{bmatrix} \frac{\partial \ell}{\partial \mu} \\ \frac{\partial \ell}{\partial \sigma} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma} \sum_{i=1}^{n} z_i \\ \frac{1}{\sigma} \left(\sum_{i=1}^{n} z_i^2 - n \right) \end{bmatrix}$$

Further, the Hessian is given by

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 \ell}{\partial \mu \partial \mu} & \frac{\partial^2 \ell}{\partial \mu \partial \sigma} \\ \frac{\partial^2 \ell}{\partial \sigma \partial \mu} & \frac{\partial^2 \ell}{\partial \sigma \partial \sigma} \end{bmatrix} = \begin{bmatrix} -\frac{n}{\sigma^2} & -\frac{2}{\sigma^2} \sum_{i=1}^{n} z_i \\ -\frac{2}{\sigma^2} \sum_{i=1}^{n} z_i & \frac{1}{\sigma^2} \left(n - 3 \sum_{i=1}^{n} z_i^2 \right) \end{bmatrix}$$

We need to specify separate functions for the log-likelihood and the derivatives. For the log-likelihood, we have

```
lnl <- function(pars, y) {
  n <- length(y)
  mu <- pars[1]
  sigma <- pars[2]
  z <- (y-mu)/sigma
  -n*log(sigma)-.5*sum(z^2)
}
```

The gradient is specified as follows:

```
grad <- function(pars, y) {
  n <- length(y)
  mu <- pars[1]
  sigma <- pars[2]
  z <- (y-mu)/sigma
  grad.vals <- numeric(2)
  grad.vals[1] <- (1/sigma)*sum(z)
  grad.vals[2] <- (1/sigma)*(sum(z^2) - n)
  return(grad.vals)
}
```

The Hessian is declared as

```
hessian <- function(pars, y) {
  n <- length(y)
  mu <- pars[1]
  sigma <- pars[2]
  z <- (y - mu)/sigma
  hess.vals <- matrix(0, nrow = 2, ncol = 2)
  hess.vals[1,1] <- -n/sigma^2
  hess.vals[1,2] <- -(2/sigma^2)*sum(z)
  hess.vals[2,1] <- -(2/sigma^2)*sum(z)
  hess.vals[2,2] <- (1/sigma^2)*(n - 3*sum(z^2))
  return(hess.vals)
}
```

We now generate 500 random draws from a $\mathcal{N}(2,2)$ *distribution:*

```r
set.seed(1071, kind = "Mersenne-Twister",
         normal.kind = "Inversion")
y <- rnorm(500, 2, 2)
summary(y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-3.5804  0.5131  1.9869  2.0036  3.5253  7.3610
```

Applying the Newton-Raphson algorithm with the analytical derivatives, we get:

```r
normal.fit <- maxLik(lnl, grad = grad, hess = hessian,
                     method = "NR", start = c(0,1),
                     y = y)
summary(normal.fit)
```

```
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 9 iterations
Return code 1: gradient close to zero
Log-Likelihood: -597.408
2  free parameters
Estimates:
     Estimate Std. error t value Pr(> t)
[1,]  2.00364    0.08959   22.36  <2e-16 ***
[2,]  2.00334    0.06335   31.62  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Constrained Optimization

It is easy to impose linear constraints in the estimation by using the **fixed** option in the **maxLik** function. This allows us to specify which parameters should be estimated and which ones should be fixed to their starting value. The following is an example if how to do this.

Example 6: *For the data in Example 5, imagine that we hypothesize* $H_0 : \sigma = 1$. *We can build this into the estimation using the following command (here I revert*

back to numerical derivatives):

```
constrained.fit <- maxLik(lnl, method = "NR",
                          start = c(0,1), y = y,
                          fixed = c(FALSE, TRUE))
```

*The command is identical to what we did before, except for the **fixed** option. The first element of* **fixed** *is set to FALSE, meaning that we estimate μ and use start = 0 merely as a starting value for this parameter. The second element of* **fixed** *is set to TRUE, meaning that we are fixing the value of σ to 1, which is the starting value for the parameter. The results are as follows.*

```
summary(constrained.fit)
```

```
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 3 iterations
Return code 1: gradient close to zero
Log-Likelihood: -1003.343
1  free parameters
Estimates:
     Estimate Std. error t value Pr(> t)
[1,]  2.00364    0.04472   44.81  <2e-16 ***
[2,]  1.00000    0.00000      NA      NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Note the standard error of 0 for $\hat{\sigma}$, which results from the fact that we did not estimate the parameter.

Wald Confidence Intervals

Wald confidence intervals take advantage of the fact that maximum likelihood estimators are asymptotically normally distributed. Normally, we would acquire these intervals using **confint**, but this does not work with **maxLik**. The following function will do the trick:

```r
ml.conf <- function(object, level) {
  z.crit <- qnorm((1-level)/2, mean = 0, sd = 1)
  est <- coef(object)
  se <- sqrt(diag(vcov(object)))
  K <- length(est)
  out <- matrix(NA, nrow = K, ncol = 4, byrow = TRUE)
  labels <- c("Estimate", "SE", "Lower", "Upper")
  colnames(out) <- labels
  out[,1] <- est
  out[,2] <- se
  out[,3] <- est+z.crit*se
  out[,4] <- est-z.crit*se
  return(out)
}
```

Example 7: *For the unconstrained estimation of the normal distribution, the 95% confidence interval is obtained as*

```r
ml.conf(normal.fit, 0.95)
```

```
     Estimate         SE    Lower    Upper
[1,] 2.003644 0.08959211 1.828047 2.179242
[2,] 2.003340 0.06335119 1.879174 2.127507
```

Likelihood Ratio Tests

In Example 6, we imposed a constraint on the normal probability density. We can test this constraint using a likelihood ratio test. Let ℓ_U and ℓ_R be the log-likelihoods for the unconstrained and constrained models, respectively. Then

$$LR = 2\left[\ell_U - \ell_R\right] \overset{asy}{\sim} \chi^2_q$$

Here *asy* denotes the asymptotic nature of the χ^2-sampling distribution and q denotes the number of restrictions imposed in the constrained model. We can obtain the LR-test statistic, q, and the p-value using the following function:

```r
lr.test <- function(U, R) {
  lnl.U <- logLik(U)
  lnl.R <- logLik(R)
  q <- nParam(U, free = TRUE) - nParam(R, free = TRUE)
  LR <- 2*(lnl.U - lnl.R)
  p <- pchisq(LR, q, lower.tail = FALSE)
  out <- matrix(c(lnl.U, lnl.R, LR, q, p), nrow = 1,
                ncol = 5)
  labels <- c("lnl.U", "lnl.R", "LR", "df", "p")
  colnames(out) <- labels
  return(out)
}
```

Example 8: *For the normal probability density function from Examples 5-6, the LR test yields:*

```r
lr.test(normal.fit, constrained.fit)
```

```
        lnl.U     lnl.R       LR df             p
[1,] -597.408 -1003.343 811.8705  1 1.416532e-178
```

Clearly, the hypothesis of a unit standard deviation has to be rejected.

More Complex Models

Regression Analysis

Consider the linear model $y = X\beta + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$. We seek to estimate β and σ. Let $\mu = X\beta$ be the **linear predictor**. Then $y \sim \mathcal{N}(\mu, \sigma^2 I)$, resulting in a likelihood of

$$\ell = -n \ln \sigma - .5 \frac{(y-\mu)^\top (y-\mu)}{\sigma^2}$$

Since $y - \mu = y - X\beta = \varepsilon$, this may also be written as

$$\ell = -n \ln \sigma - .5 \frac{\varepsilon^\top \varepsilon}{\sigma^2}$$

To program this, we declare the following log-likelihood:

```
lm.lnl <- function(pars, data) {
  n <- nrow(data)
  K <- ncol(data)-1
  b.hat <- pars[1:K]
  s <- pars[K+1]
  y <- data[,1]
  X <- data[,2:(K+1)]
  e <- y - X%*%b.hat
  -n*log(s)-.5*(t(e)%*%e)/s^2
}
```

We now apply the linear model to the Boston Housing data contained in the **mlbench** package. We regress crime and taxes onto the corrected median housing value. The data preparation is as follows:

```
library(mlbench)
data("BostonHousing2")
boston.sub <- cbind(BostonHousing2$cmedv,
                    1,
                    BostonHousing2$crim,
                    BostonHousing2$tax)
summary(boston.sub)
```

```
       V1              V2          V3                 V4
 Min.   : 5.00   Min.   :1   Min.   : 0.00632   Min.   :187.0
 1st Qu.:17.02   1st Qu.:1   1st Qu.: 0.08204   1st Qu.:279.0
 Median :21.20   Median :1   Median : 0.25651   Median :330.0
 Mean   :22.53   Mean   :1   Mean   : 3.61352   Mean   :408.2
 3rd Qu.:25.00   3rd Qu.:1   3rd Qu.: 3.67708   3rd Qu.:666.0
 Max.   :50.00   Max.   :1   Max.   :88.97620   Max.   :711.0
```

For the estimation we take starting values of 0 for the three regression coefficients (the constant and the partial slopes for crime and taxes). We set a starting value of 100 for σ.

```
boston.fit <- maxLik(lm.lnl, method = "NR",
                     start = c(rep(0,3), 100),
                     data = boston.sub)
summary(boston.fit)
```

```
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 14 iterations
Return code 2: successive function values within tolerance limit
Log-Likelihood: -1304.127
4  free parameters
Estimates:
     Estimate Std. error t value  Pr(> t)
[1,] 31.44741    4.57980   6.867 6.58e-12 ***
[2,] -0.18514    0.10846  -1.707   0.0878 .
[3,] -0.02021    0.01080  -1.871   0.0613 .
[4,]  7.98309    0.25066  31.849  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Note that the estimate for σ is *not* the same as the RMSEA from the **lm** command. Our estimate does not adjust for the degrees of freedom lost by estimating the regression coefficients and, as such, shows a downward bias.

Heteroskedastic Regression

Let us relax the assumption that the regression errors have a constant variance of σ^2. Instead, we adopt the approach of Harvey (1976) and assume multiplicative heteroskedasticity of the form

$$\sigma_i^2 = \exp(z_i^\top \gamma)$$

where i denotes a particular unit. Writing the unit-specific mean as $\mu_i = x_i^\top \beta$, we obtain the following log-likelihood for a single unit:

$$\ell_i = -.5 z_i^\top \gamma - .5 \exp(z_i^\top \gamma)(y_i - x_i^\top \beta)^2$$

This time, we keep the data for the outcome and the two sets of predictors separate. Thus, the log-likelihood is programmed as

```r
harvey.lnl <- function(pars, y, x, z) {
  P <- ncol(x)
  Q <- ncol(z)
  b <- pars[1:P]
  g <- pars[(P+1):(P+Q)]
  lp1 <- x%*%b
  lp2 <- z%*%g
  -.5*lp2-.5*exp(lp2)*(y-lp1)^2
}
```

We now apply the heteroskedastic model to the Boston Housing data. The predictors of the mean model are crime and taxes. The predictor of the variance model is crime.

```r
y <- BostonHousing2$cmedv
x <- cbind(1, BostonHousing2$crim, BostonHousing2$tax)
z <- cbind(1, BostonHousing2$crim)
harvey.fit <- maxLik(harvey.lnl, method = "BHHH",
                     start = rep(0, 5),
                     y = y, x = x, z = z)
summary(harvey.fit)
```

```
--------------------------------------------
Maximum Likelihood estimation
BHHH maximisation, 150 iterations
Return code 4: Iteration limit exceeded.
Log-Likelihood: -81664.63
5  free parameters
Estimates:
       Estimate Std. error t value Pr(> t)
[1,]  1.004e+01  1.112e-02  903.48  <2e-16 ***
[2,] -5.570e-02  1.112e-03  -50.11  <2e-16 ***
[3,]  3.557e-03  2.799e-05  127.08  <2e-16 ***
[4,]  4.191e-01  4.402e-04  952.13  <2e-16 ***
[5,] -1.606e-03  1.117e-04  -14.38  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Logit and Probit Models

Consider a binary outcome $Y \in \{0, 1\}$. The 1s occur with probability π_i and the 0s with probability $1 - \pi_i$. Further, $\pi_i = F(x_i^\top \beta)$, where F is either the standard logistic or the standard normal distribution function.

The log-likelihood function for a single unit derives from the Bernoulli distribution

$$f(y_i) = \pi_i^{y_i}(1 - \pi_i)^{1-y_i}$$

Specifically, by taking the natural logarithm, we obtain

$$\ell_i = y_i \ln \pi_i + (1 - y_i) \ln(1 - \pi_i) = y_i \ln F(x_i^\top \beta) + (1 - y_i) \ln \left[1 - F(x_i^\top \beta)\right]$$

This can be simplified (see Greene (1993)). Let $q_i = 2y_i - 1$, then

$$\ell_i = \ln F(q_i x_i^\top \beta)$$

The full log-likelihood is $\ell = \sum_{i=1}^n \ell_i$.

We now program the logit log-likelihood as follows:

```
logit.lnl <- function(pars, y, x) {
  K <- ncol(x)
  b <- pars[1:K]
  lp <- x%*%b
  q <- 2*y-1
  sum(log(plogis(q*lp)))
}
```

(For probit, we simply substitute **pnorm** for **plogis**.)

Let us apply the code to Ronald Fisher's famous iris data. Imagine we try to predict whether an iris is of the species virginica. We prepare the data as follows:

```
data(iris)
y <- ifelse(iris$Species=="virginica", 1, 0)
x <- cbind(1, iris$Sepal.Length, iris$Sepal.Width,
           iris$Petal.Length, iris$Petal.Width)
```

Instead of making up starting values, we now adopt the values generated by a linear probability model:

```
lpm.fit <- lm(y~x - 1)
st.val <- coef(lpm.fit)
```

(We drop the intercept because **x** already includes a constant.) The logit estimation then yields

```
logit.fit <- maxLik(logit.lnl, method = "NR",
                    start = st.val, y = y, x = x)
summary(logit.fit)
```

```
--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 19 iterations
Return code 2: successive function values within tolerance limit
Log-Likelihood: -5.949273
5  free parameters
Estimates:
   Estimate Std. error t value Pr(> t)
x1  -42.638     25.060  -1.701  0.0889 .
x2   -2.465      2.312  -1.066  0.2864
x3   -6.681      4.983  -1.341  0.1800
x4    9.429      4.565   2.066  0.0388 *
x5   18.286     11.032   1.658  0.0974 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

References

Greene, William H. 1993. *Econometric Analysis*. 2nd ed. New York: Macmillan.

Harvey, A. C. 1976. "Estimating Regression Models with Multiplicative Heteroskedasticity." *Econometrica* 44 (3): 461–65.

Henningsen, Arne, and Ott Toomet. 2011. "MaxLik: A Package for Maximum Likelihood Estimation in R." *Computational Statistics* 26 (3): 443–58.