University of Edinburgh, School of Mathematics
Incomplete Data Analysis, 2020/2021
Numerical methods for maximum likelihood estimation

Vanda Inácio

In some cases, it will not be possible to obtain a closed form expression for the maximum likelihood estimator. In such cases, we need to resort to numerical iterative procedures. There are several numerical procedures that one can employ in order to calculate mle estimates and, among the most popular, are the Newton–Raphson/Fisher-Scoring method, the method of gradient descent and the EM algorithm. Which one is more appropriate depends on the specific problem. What it is common to all of them is that they are iterative: they start at a given input value and iterate some operation until the convergence criterion is attained. Luckily, `R` has very good optimisation tools

The Newton–Raphson method is based on a first order Taylor approximation of the score function. Newton's method for finding the solution $\widehat{\boldsymbol{\theta}}$ to $U(\boldsymbol{\theta}) = \mathbf{0}$ can be described as

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \left[J\left(\boldsymbol{\theta}^{(t)}\right)\right]^{-1} U\left(\boldsymbol{\theta}^{(t)}\right).$$

However, the behaviour of $J\left(\boldsymbol{\theta}^{(t)}\right)$ can be problematic if $\boldsymbol{\theta}^{(t)}$ is far from the mle $\widehat{\boldsymbol{\theta}}$. As a remedy, instead of using the observed Fisher information $J(\boldsymbol{\theta})$, we can use the expected Fisher information $I(\boldsymbol{\theta})$. The updating equations now take the following form

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \left[I\left(\boldsymbol{\theta}^{(t)}\right)\right]^{-1} U\left(\boldsymbol{\theta}^{(t)}\right).$$

This algorithm is called the Fisher scoring method.

We will consider, firstly, applying the above methods to the Cauchy distribution. Let us suppose that $Y_1, \ldots, Y_n$ are iid random variables following the Cauchy distribution with unknown location θ ($\in \mathbb{R}$) and scale equal to one, whose density function is given by

$$f(y;\theta) = \frac{1}{\pi(1+(y-\theta)^2)}, \qquad y \in \mathbb{R}.$$

The log likelihood function in this case is

$$\log L(\theta;\mathbf{y}) = -n\log\pi - \sum_{i=1}^{n}\log\left\{1+(y_i-\theta)^2\right\}.$$

The root of the score equation has no closed form expression, as one can appreciate below

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\log L(\theta;\mathbf{y}) = 0 \Rightarrow 2\sum_{i=1}^{n}\frac{y_i-\theta}{1+(y_i-\theta)^2} = 0.$$

To implement the Newton–Raphson and Fisher scoring methods we need to calculate the second derivative of the log likelihood, which is given by

$$\frac{\mathrm{d}^2}{\mathrm{d}\theta^2}\log L(\theta;\mathbf{y}) = 2\sum_{i=1}^{n}\frac{(y_i-\theta)^2-1}{\{1+(y_i-\theta)^2\}^2}.$$

Having the second derivative of the log likelihood is all we need to the Newton–Raphson method (because we only need the observed Fisher information). For the Fisher scoring method we need to obtain the expected Fisher information, which after some (read: a lot) calculations, turned out to be $I(\theta) = n/2$. For instance, it was deduced here

http://wwwf.imperial.ac.uk/ das01/MyWeb/M3S3/Handouts/WorkedExample-Cauchy.pdf

(note that the author obtains 1/2 because they are only calculating it for a single observation).

I have simulated $n = 100$ observations from a Cauchy distribution with location $\theta = 0$ (which we want to estimate) and scale 1 (assumed to be known).

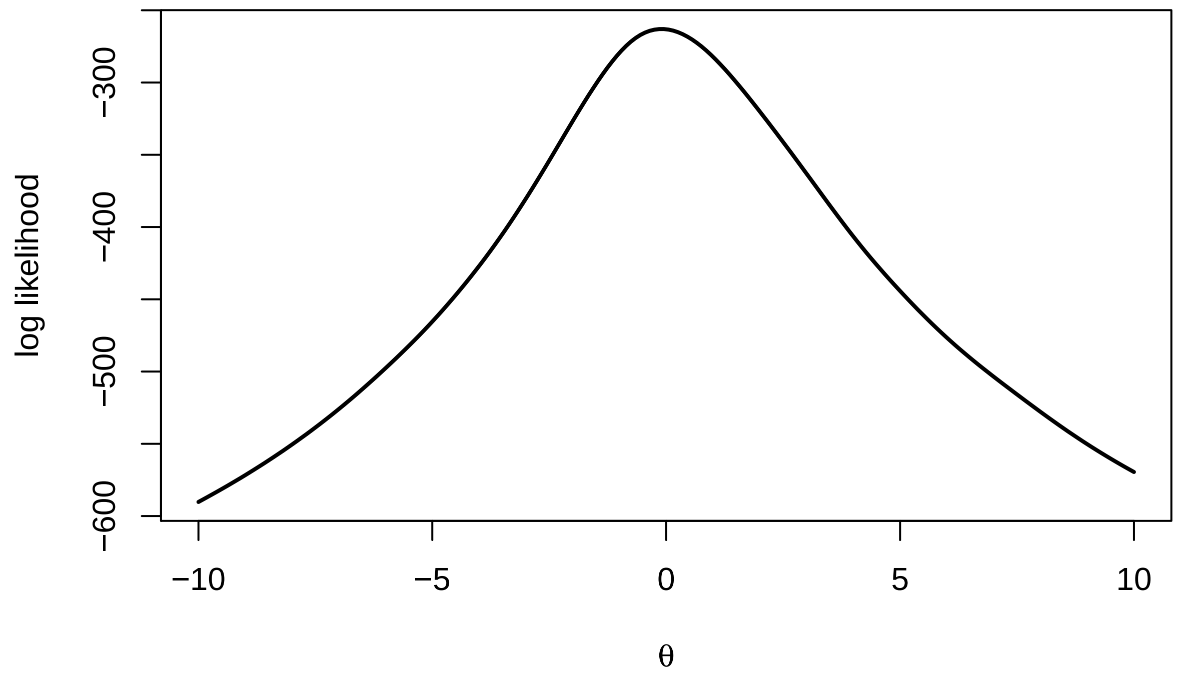
```
set.seed(1)
n <- 100
y <- rcauchy(n, location = 0, scale = 1)
```

Because I have simulated data I obviously know the true value but let us look at the log likelihood function, which is always a good practice and feasible in this one dimensional case.

```
log_like_cauchy <- function(y, theta){
  sum(dcauchy(y, location = theta, scale = 1, log = TRUE))
}

thetagrid <- seq(-10, 10, len = 200); ntheta <- length(thetagrid)
res <- numeric(ntheta)
for(i in 1:ntheta){
res[i] <- log_like_cauchy(y = y, theta = thetagrid[i])
}

plot(thetagrid, res, type = "l", xlab=expression(theta), ylab = "log likelihood", lwd = 2)
```



The following function implements the Newton–Raphson for this case. It is a very simple function and, obviously, there is a lot of room for improvements. For monitoring convergence or, possibly better stated, for our stopping criterion, we use $|\theta^{(t+1)} - \theta^{(t)}| < \epsilon$ and we denote bu `theta0` the initial value $\theta^{(0)}$ to start the iterative procedure.

```r
nr.cauchy <- function(y, theta0, eps){

theta <- theta0
diff <- 1

while(diff > eps){
theta.old <- theta

lprime <- 2*sum((y-theta)/(1+(y-theta)^2))
l2prime <- 2*sum((((y-theta)^2)-1)/((1+(y-theta)^2)^2))
theta <- theta-lprime/l2prime

diff <- abs(theta-theta.old)
}
list(theta, l2prime)
}
```

We now apply the above function with an initial value $\theta^{(0)} = 1$ and $\epsilon = 0.000001$.

```r
nr.cauchy(y = y, 1, 0.000001)
```

```
## [[1]]
## [1] -0.09820963
##
## [[2]]
## [1] -37.82092
```

However, if we try with a starting value which is more distant of the true value, e.g., $\theta^{(0)} = 3$ or $\theta^{(0)} = 10$, the function will give us an error, due to convergence problems. Let us now try the Fisher scoring and check whether this sesitivity to the initial value persists. The function is very similar to the previous one.

```r
fs.cauchy <- function(y, theta0, eps){

theta <- theta0
diff <- 1

while(diff > eps){
theta.old <- theta

lprime <- 2*sum((y-theta)/(1+(y-theta)^2))
info.fisher <- n/2
theta <- theta+lprime/info.fisher

diff <- abs(theta-theta.old)
}
list(theta)
}
```

As we see, this method is much less sensitive to the choice of the starting value.

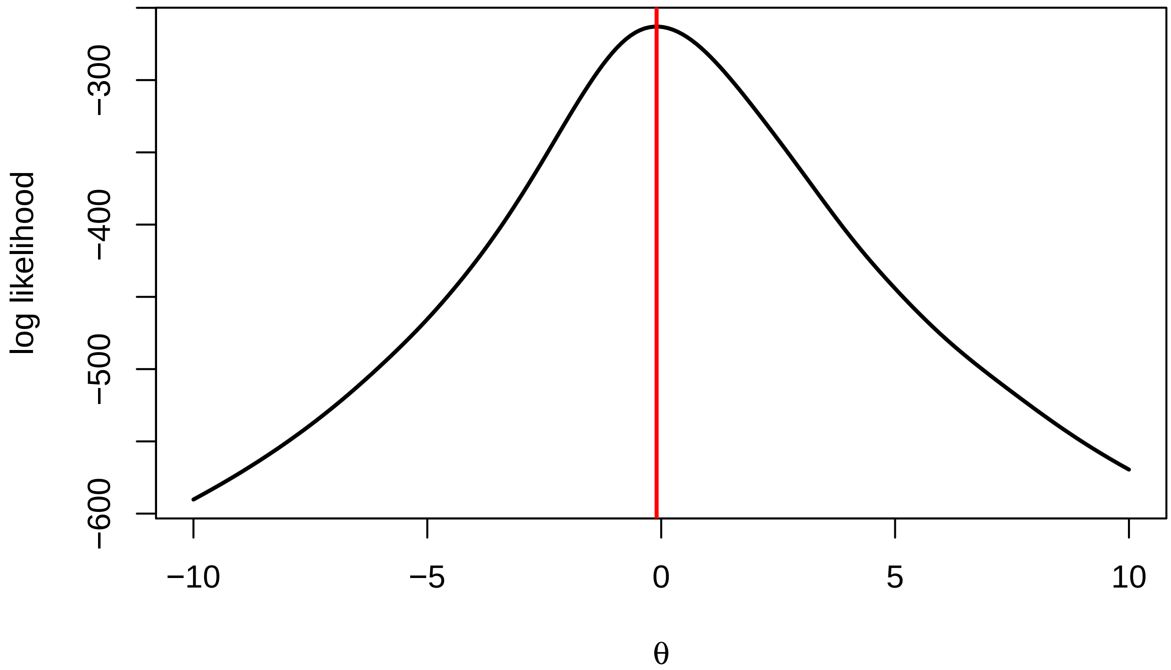
```r
fs.cauchy(y = y, 15, 0.000001)
```

```
## [[1]]
## [1] -0.09820936
```

```r
fs.cauchy(y = y, 100, 0.000001)
```

```
## [[1]]
## [1] -0.09820945
```

```
plot(thetagrid, res, type = "l", xlab=expression(theta), ylab = "log likelihood", lwd = 2)
abline(v = -0.09820945, lwd = 2, col = "red")
```



Let us now learn how to use packages or built in functions that can do the optimisation automatically. We start by illustrating the use of the package `maxLik`, which needs to be installed in advance. The package requires us to pass the log likelihood function, which we have already defined before. We further need to pass to the function a starting value and the data.

```
require(maxLik)
mle <- maxLik(logLik = log_like_cauchy, y = y, start = c(15))
summary(mle)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 7 iterations
## Return code 1: gradient close to zero
## Log-Likelihood: -262.9641
## 1  free parameters
## Estimates:
##      Estimate Std. error t value Pr(> t)
## [1,] -0.09821    0.16253  -0.604   0.546
## --------------------------------------------
```

What we have obtained is almost identical to what we have obtained before with our own coded function. The estimate is $\widehat{\theta} = -0.09821$. We also obtain that the satndard error of the estimate is 0.16253. Note that when running the Newton–Raphson we have asked for the output to include the second derivative evaluated at the last iteration. This value was -37.82092 and because we know the observed Fihser information is the negative of this value and that the variance of the mle estimator is the inverse of the observed Fisher information. The variance is 1/37.82092 and the standard error is the square root of this value, which is 0.162605.

```
sqrt(1/37.82092)
```

```
## [1] 0.162605
```

We can, alternatively, use the built-in function `optim`; for more info type `help(optim)`. The function has converged (`convergence=0`) but has ‘complained’, because we only have one parameter and so we can use the `optimize` function (this is `method = Brent`), which also requires bounds to be provided by the parameter estimate.

```
mleoptim <- optim(par = c(15), fn = log_like_cauchy, y = y,
                  control = list("fnscale"=-1), hessian = TRUE)
```

```
## Warning in optim(par = c(15), fn = log_like_cauchy, y = y, control = list(fnscale = -1), : one-dimens
## use "Brent" or optimize() directly
```

```
mleoptim
```

```
## $par
## [1] -0.09960938
##
## $value
## [1] -262.9641
##
## $counts
## function gradient
##       30       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]
## [1,] -37.84502
```

```
mleoptim_brent <- optim(par = c(15), fn = log_like_cauchy, y = y,
                        method = "Brent", lower = -10, upper = 10,
                        control = list("fnscale"=-1), hessian = TRUE)
mleoptim_brent
```

```
## $par
## [1] -0.09820959
##
## $value
## [1] -262.9641
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]
## [1,] -37.82092
```

Let us now study a problem with two parameters. The Weibull distribution is widely used in survival analysis. Its density function is given by

$$f(y;\theta) = \frac{\alpha}{\beta}\left(\frac{y}{\beta}\right)^{\alpha-1} \exp\left\{-\left(\frac{y}{\beta}\right)^{\alpha}\right\}, \quad y > 0, \quad \alpha > 0, \quad \beta > 0.$$

Here α is a shape parameter and β is a scale parameter, and $\theta = (\alpha, \beta)$. The log likelihood function is given by

$$\log L(\boldsymbol{\theta}; \mathbf{y}) = n\log(\alpha) - n\alpha\log(\beta) + (\alpha - 1)\sum_{i=1}^{n}\log(y_i) - \frac{1}{\beta^{\alpha}}\sum_{i=1}^{n} y_i^{\alpha}.$$

Differentiation leads to the system of equations:

$$\begin{cases} -\frac{n\alpha}{\beta} + \frac{\alpha}{\beta}\sum_{i=1}^{n}\left(\frac{y_i}{\beta}\right)^{\alpha} = 0 \\ \frac{n}{\alpha} + \sum_{i=1}^{n}\log\left(\frac{y_i}{\beta}\right) - \sum_{i=1}^{n}(\frac{y_i}{\beta})^{\alpha}\log\left(\frac{y_i}{\beta}\right) = 0 \end{cases}$$

To compute the solutions of this system we resort to iterative numerical methods. Let us now simulate some data; we have considered $\alpha = 3$ and $\beta = 2$.

```r
set.seed(1)
n <- 500
y <- rweibull(n, shape = 3, scale = 2)
```

As in the previous example, we start by defining the log likelihood.

```r
log_like_weibull <- function(y, param){
alpha <- param[1]
beta <- param[2]
sum(dweibull(y, shape = alpha, scale = beta, log = TRUE))
}
```

We then use the `maxLik` function, using as starting values 8 (shape) and 10 (scale).

```r
mle <- maxLik(logLik = log_like_weibull, y = y, start = c(8,10))
summary(mle)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 12 iterations
## Return code 1: gradient close to zero
## Log-Likelihood: -469.0514
## 2  free parameters
## Estimates:
##      Estimate Std. error t value Pr(> t)
## [1,]  3.17556    0.11233   28.27  <2e-16 ***
## [2,]  1.99107    0.02947   67.56  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

Alternatively, using the `optim` function.

```
weibopt <- optim(par = c(8,10), fn = log_like_weibull, y = y,
                 control = list("fnscale"=-1), hessian = TRUE)
weibopt
```

```
## $par
## [1] 3.174831 1.990649
##
## $value
## [1] -469.0515
##
## $counts
## function gradient
##      131       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]       [,2]
## [1,] -87.56616   103.0438
## [2,] 103.04380 -1272.7949
```

```
sqrt(solve(-weibopt$hessian))
```

```
##             [,1]       [,2]
## [1,] 0.11234972 0.03196713
## [2,] 0.03196713 0.02946870
```

The entries in the main diagonal of this last matrix are the standard errors of α and β, respectively (almost identical to the ones reported by the `maxLik` function).