University of Edinburgh
School of Mathematics
Incomplete Data Analysis

Supporting Materials for Lecture 5

Numerical MLEs

We show here how to use R to numerically maximise the likelihood or log likelihood function. As usually, I start by cleaning the workspace and fixing the seed.

```
rm(list=ls())
set.seed(1)
```

Let us start with the Cauchy example, simulating $n = 100$ observations from a Cauchy distribution with location 0 (which we want to estimate) and scale 1 (assumed to be known).

```
n=100
y=rcauchy(n,location=0,scale=1)
```

There are several R routines that can be used. 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.

```
require(maxLik)

#log likelihood function to be optimised
logLikFun=function(y,theta){
sum(dcauchy(y,location=theta,scale=1,log=TRUE))
}
```

Having defined the loglikelihood function, which we have called **logLikFun**, we further need to pass a starting value and the data. There is no rule of thumb for selecting the starting value; if the method of moments estimator can be easily derived, it might serve as a 'good' starting value. In this case, I have used the value 15.

```
mle=maxLik(logLik=logLikFun,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
## --------------------------------------------
```

We have obtained $\widehat{\theta} = -0.09821$ and the corresponding standard error is 0.16253. We can, alternatively, use the built-in function `optim`; for more info type help(optim)

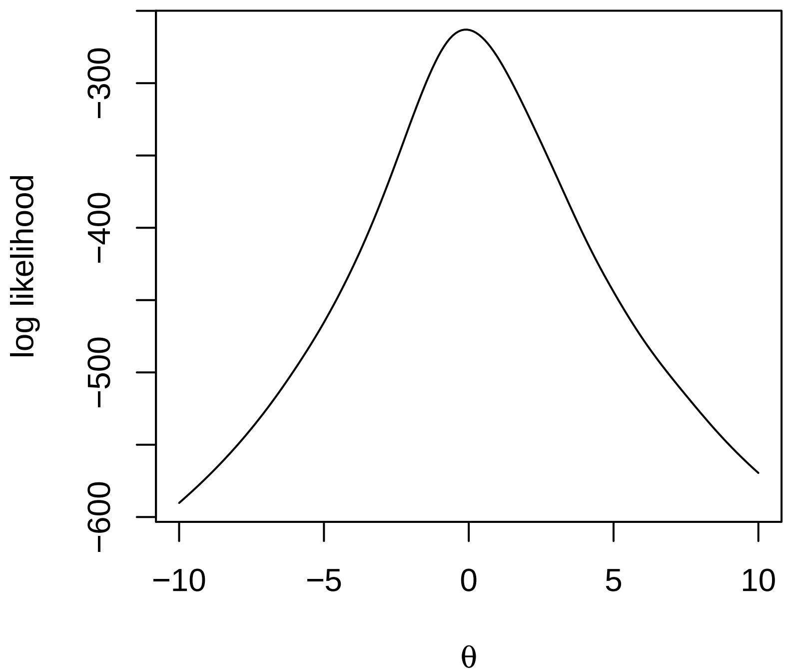
```
mleoptim=optim(par=c(15),fn=logLikFun,y=y,control=list("fnscale"=-1),hessian=TRUE)
mleoptim

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

We obtain a very similar estimate. Note that we can also extract the standard error: $1/\sqrt{37.845022} = 0.1625532$. To visually check that our estimate is correct, we can plot the log likelihood over a grid of possible values of θ.

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

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



We can also easily implement our own procedure to maximise the log likelihood function. The following function implements the Newton–Raphson for this case.

```
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)
}
```

If we apply the above function to these data with the initial value of 15 that we have used before, we get an error, which is indicative of convergence problems.

```
nr.cauchy(y=y,15,0.000001)

## Error in while (diff > eps) {:  missing value where TRUE/FALSE needed
```

Since in this case we know that the optimum is zero, we can try to pick up a starting value that is closer to the optimum.

```
nr.cauchy(y=y,3,0.000001)

## Error in while (diff > eps) {: missing value where TRUE/FALSE needed

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

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

The dependency on the starting values is a know 'problem' of the Newton–Raphson method. Let us now code the Fisher-Scoring algorithm.

```
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.

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

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

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

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

We will now focus on the Weibull example. We first generate data.

```
n=500
y=rweibull(n,shape=3,scale=2)
```

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

```
logLikFun=function(y,param){
a=param[1]
b=param[2]
sum(dweibull(y,shape=a,scale=b,log=TRUE))
}
```

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

```
mle=maxLik(logLik=logLikFun,y=y,start=c(8,10))
summary(mle)

## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 10 iterations
## Return code 1: gradient close to zero
## Log-Likelihood: -479.3393
## 2  free parameters
## Estimates:
##      Estimate Std. error t value Pr(> t)
## [1,]  3.08868    0.10924   28.27  <2e-16 ***
## [2,]  1.98682    0.03023   65.72  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

Alternatively, using the `optim` function.

```
weibopt=optim(par=c(5,5),fn=logLikFun,y=y,control=list("fnscale"=-1),hessian=TRUE)
weibopt

## $par
## [1] 3.088875 1.986933
##
## $value
## [1] -479.3393
##
## $counts
## function gradient
##       73       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]       [,2]
## [1,] -92.48647   102.7554
## [2,] 102.75542 -1208.1456

sqrt(solve(-weibopt$hessian))

##            [,1]       [,2]
## [1,] 0.10927369 0.03186826
## [2,] 0.03186826 0.03023396
```