Computational Statistics

Lab 2

Emil K Svensson and Rasmus Holm

2017-02-13

Question 1

1.1

```
mort <- read.csv2("../data/mortality_rate.csv")
mort$LMR <- log(mort$Rate)

n <- dim(mort)[1]
set.seed(123456)
id <- sample(1:n, floor(n*0.5))
train <- mort[id, ]
test <- mort[-id, ]
```

1.2

```
myMSE <- function(lambda, pars) {
    MSEcounter <<- MSEcounter + 1

    data  <- data.frame(pars$X, Y=pars$Y)
    model <- loess(formula=Y ~ ., data=data, enp.target=lambda)
    mean((pars$Ytest - predict(model, pars$Xtest))^2)
}
```

1.3

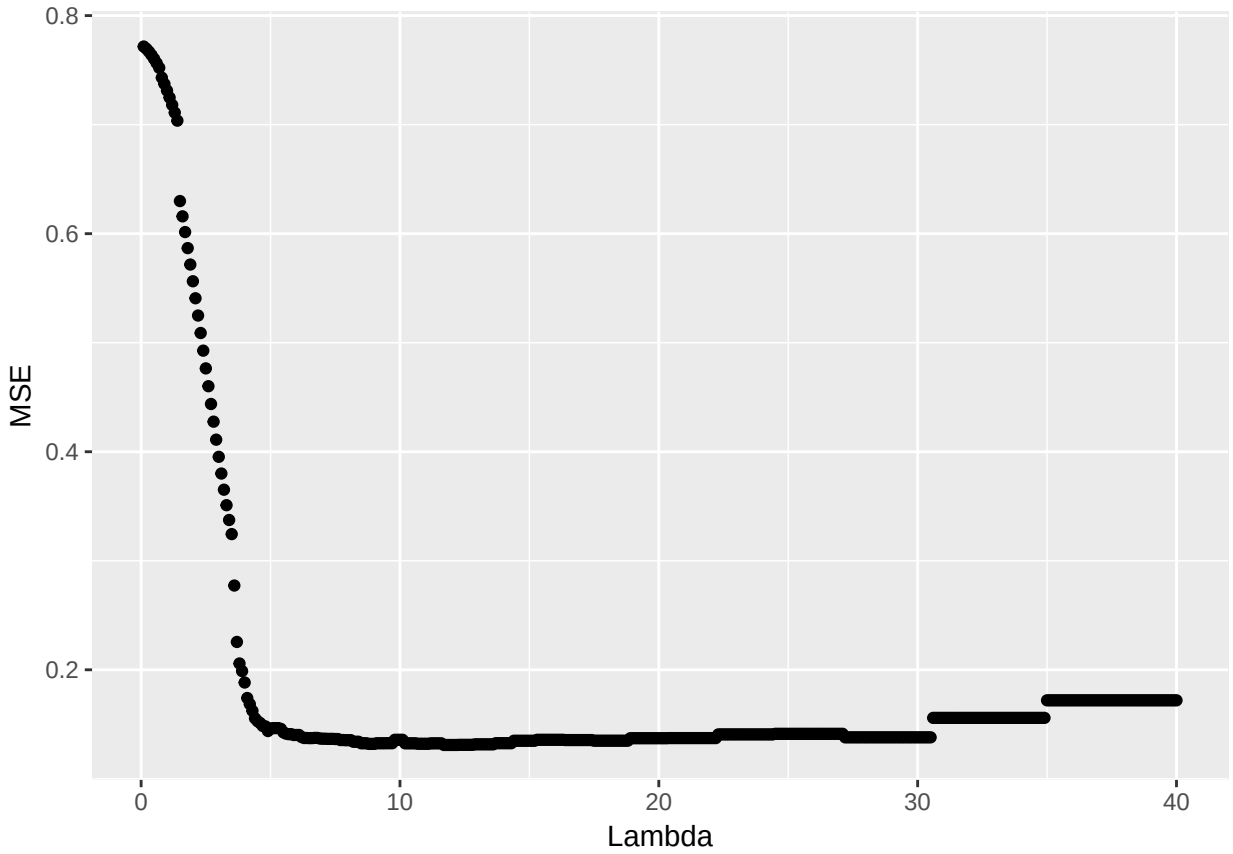
```
MSEcounter <- 0
lambdas <- seq(0.1, 40, by=0.1)
pars <- list(X=train$Day, Y=train$LMR, Xtest=test$Day, Ytest=test$LMR)

MSES <- sapply(lambdas, FUN=myMSE, pars=pars)
```

1.4

```
library(ggplot2)

ggplot() + geom_point(aes(x=lambdas, y=MSES)) +
    xlab("Lambda") + ylab("MSE")
```



The minimum MSE is achieved when lambda is 11.7 which is the optimal value of those we tried. The number of evaluations required for this task was 400, i.e. the number of lambdas that we tried.

1.5

```
MSEcounter <- 0
opt <- optimize(myMSE, lower=0.1, upper=40, tol=0.01, pars=pars)
```

The optimization algorithm fails to find the minimum MSE since it identifies it as 10.6936107 because the MSE at that point is a local minimum. The number of evaluations are lower than in previous experiment, 18 compared to 400.

1.6

```
MSEcounter <- 0

optim(par=list(lambda=35), fn=myMSE, method="BFGS", pars=pars)
```

```
## $par
## lambda
##     35
##
## $value
## [1] 0.1719996
##
```

```
## $counts
## function gradient
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
paste("The number of evaluations:", MSEcounter)
```

```
## [1] "The number of evaluations: 3"
```

The optimal lambda here was as we specified, lambda = 35, because the MSE around lambda = 35 is a plateau and the gradient (first derivative) is therefore zero and the algorithm stops since there is no direction to go.

Question 2

2.1

```
load("../data/data.RData")
```

2.2

Since the data is from a Gaussian distribution we know that

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

and the likelihood is

$$L(\mu,\sigma) = \prod_{i=1}^{n} f(x_i;\mu,\sigma)$$

where $n = 100$. The log-likelihood is then

$$\ln L(\mu,\sigma) = \sum_{i=1}^{n} \ln f(x_i;\mu,\sigma) = -\frac{n}{2}\ln(\sigma^2) - \frac{n}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2.$$

Taking the derivative of the log-likelihood with respect to μ and setting it to zero we get

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and the derivative with respect to σ^2 and setting it to zero gives

$$\hat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}(x_i-\hat{\mu})^2.$$

2.3

```
neg_llik <- function(pars, data){
    neg_llik_counter <<- neg_llik_counter + 1

    mu <- pars[1]
    sigma <- pars[2]
    n <- length(data)
    (n / 2) * log(sigma^2) + (n / 2) * log(2 * pi) + (1 / (2 * sigma^2)) * sum((data - mu)^2)
}

neg_llikgrad <- function(pars, data) {
    neg_llikgrad_counter <<- neg_llikgrad_counter + 1

    n <- length(data)
    mu <- pars[1]
    sigma <- pars[2]

    mugrad <- -sum(data - mu) / sigma^2
    sigmagrad <- (n / sigma) - sum((data - mu)^2) / sigma^3
    c(mugrad, sigmagrad)
}

mle <- function(data) {
    n <- length(data)
    mu <- sum(data) / n
    sigma <- (1 / n) * sum((data - mu)^2)
    c(mu, sigma)
}
```

The reason why choosing the log-likelihood instead of the likelihood is because of numerical precision. Since we are calculating small probabilities we don't want to multiply them and get even smaller numbers, possibly underflow. When taking the log-likelihood we turn the multiplication into addition which does not run the risk of underflow and the logarithm is a monotonic function meaning the optimum will be equivalent. Finding the derivatives also simplifies with the logarithm.

The maximum likelihood estimates are: $\mu = 1.2755276$ and $\sigma = 2.0059765$.

```
neg_llik_counter <- 0
optim(par =c(0, 1), fn=neg_llik, method="BFGS", data=data)
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       37       15
##
## $convergence
## [1] 0
##
## $message
```

```
## NULL
```

```
paste("The number of evaluations of the log-likelihood:", neg_llik_counter)
```

```
## [1] "The number of evaluations of the log-likelihood: 97"
```

```
neg_llik_counter <- 0
neg_llikgrad_counter <- 0
optim(par =c(0, 1), fn=neg_llik, gr=neg_llikgrad, method="BFGS", data=data)
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       38       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
paste("The number of evaluations of the log-likelihood:", neg_llik_counter)
```

```
## [1] "The number of evaluations of the log-likelihood: 38"
```

```
paste("The number of evaluations of the gradient:", neg_llikgrad_counter)
```

```
## [1] "The number of evaluations of the gradient: 15"
```

```
neg_llik_counter <- 0
optim(par =c(0, 1), fn=neg_llik, method="CG", data=data)
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      180       33
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
paste("The number of evaluations of the log-likelihood:", neg_llik_counter)
```

```
## [1] "The number of evaluations of the log-likelihood: 312"
```

```
neg_llik_counter <- 0
neg_llikgrad_counter <- 0
```

```r
optim(par =c(0, 1), fn=neg_llik, gr=neg_llikgrad, method="CG", data=data)
```

```
## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       53       17
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
paste("The number of evaluations of the gradient:", neg_llikgrad_counter)
```

```
## [1] "The number of evaluations of the gradient: 17"
```

```r
paste("The number of evaluations of the log-likelihood:", neg_llik_counter)
```

```
## [1] "The number of evaluations of the log-likelihood: 53"
```

2.4

All optimizations converged. The results can be seen above in the par-variable for each print out where the first value is μ and the second value is σ. The true values are $\mu = 1.2755276$, $\sigma = 2.0160822$ and we can see that both algorithms regardless of setting found the same μ, σ which are equal to the maximum likelihood estimates. These values are also very close to the true values which indicates that all algorithms found good estimates in this example. However, the number of function calls differ quite dramatically between BFGS and conjugate gradient. BFGS is clearly superior and we can see that the number of function calls are similar whether the gradient is specified or not, one less function call when gradient is not specified. Note though that when the gradient is missing each gradient is approximated by two regular function calls so the total number of function calls is actually higher, 97 vs 38. We conclude that we would prefer the BFGS with the gradient explicitly specified in this example.