732A90: Computational Statistics

Computer lab2 - Group11

Sofie Jörgensen, Oriol Garrobé Guilera, David Hrabovszki

04 February 2020

Question 1: Optimizing a model parameter

The aim of the first question is to perform optimization by using a data set, `mortality rate.csv`, consisting of information about the mortality rates of fruit flies during an observed period.

1.

First, we import the file to R, and then add a variable called `LMR` to the data set. The new defined variable `LMR` is the natural logarithm of `Rate`. Thereafter, we split the data into a training set and a test sets, respectively. The splitting is done using the code which is already given (see Appendix).

2.

We write our own function `myMSE()` that for any given parameters λ and list `pars` containing vectors `X`, `Y`, `Xtest`, `Ytest` fits a LOESS model with response Y and predictor X using `loess()` function with penalty λ and then predicts the model for `Xtest`.

The function computes and returns the predictive MSE, which is defined as

$$\frac{1}{length(test)} \sum_i \left(Ytest[i] - fYpred(X[i])\right)^2,$$

where $fYpred(X[i])$ is the predicted value of `Y` if `X` is `X[i]`.

The function code is defined in the Appendix.

3.

Now we use our function `myMSE()` and the training and test sets defined in Question 1.1. with response LMR and predictor Day to estimate the predictive MSE values. The penalty values are chosen as $\lambda = 0.1, 0.2, ..., 40$.

4.

Looking at Figure 1, and running an R command we can observe that 11.7 is the optimal value for λ. At this point the MSE is 0.131047.

Since the step size in λ was 0.1 and the starting point 0.1, the optimal value of 11.7 means that 117 evaluations of `myMSE()` were required to find it.

5.

In this task we use the `optimize()` function to find the optimal λ, with range [0.1,40] and accuracy 0.01.

The result for the minimum value is 0.1321441 and the function achieves this at $\lambda = 10.69361$. Therefore, we conclude that the optimizing function didn't find the optimal solution precisely, but it was very close to it. The `optimize()` called our `myMSE()` 18 times, this many evaluations were required to reach the solution. This, compared to the 117 evaluations from step 4 means that `optimize()` optimized much more efficiently, although the result is not exactly accurate.

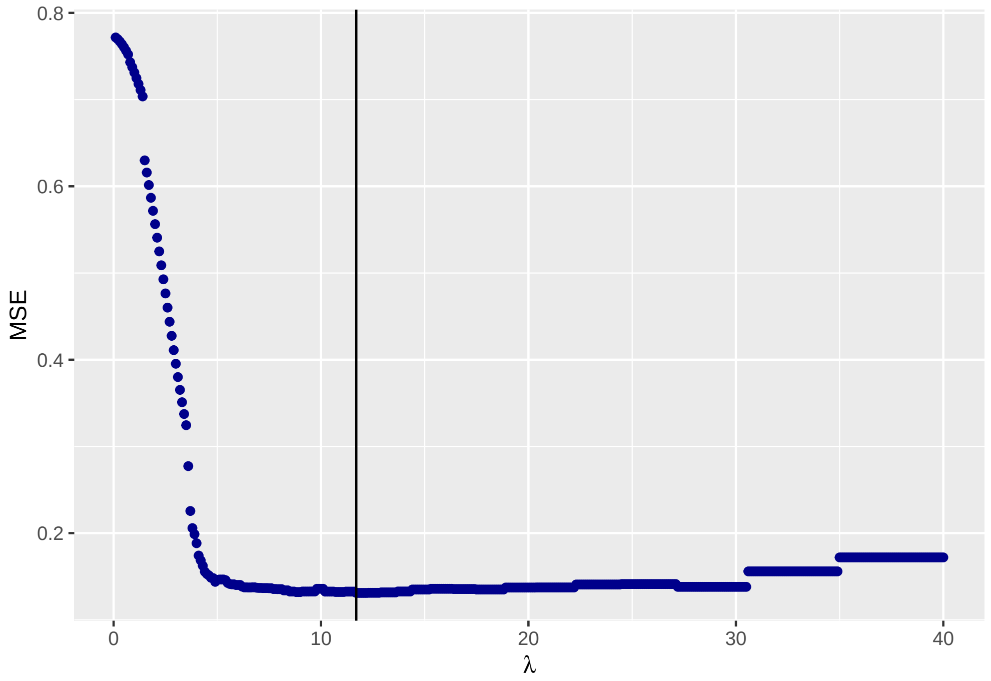

Figure 1: MSE values as a function of lambda

6.

In the last task we use the `optim()` function and BFGS method with starting point $\lambda = 35$ to find the optimal λ.

This time only 1 `myMSE()` function evaluation was required, because `optim()` returns the value obtained at the starting point. This is because the MSE function is flat around that point (see Figure 1), which means that the gradient is 0, therefore no optimization is possible with this algorithm. Also, the starting point of 35 is too high, but if we set it to a lower value, where the MSE function is not flat, the `optim()` function finds a better optimum.

The `optimize()` function in step 5 looked for the optimal λ on the interval [0.1,40], which meant that optimizing was possible, but in this step, the starting point is too high, that is why we obtained such different results.

Question 2: Maximum likelihood

1.

In this task we will use the file `data.RData` consists of a sample coming from normal distribution with parameters μ and σ. First we load the data set into R.

2.

The sample comes from a normal distribution with parameters μ and σ, where we set $\theta = (\mu, \sigma)$. Under the assumption that the sample $\boldsymbol{x} = (x_1, ..., x_{100})$ is iid, i.e. $\boldsymbol{X_i} \overset{iid}{\sim} N(\mu, \sigma^2)$, for $i = 1, ..., 100$, then the joint density function of all $n = 100$ observations can be written as

$$L(\theta; \boldsymbol{x}) = f(\boldsymbol{x}|\theta) = \prod_{i=1}^{100} f(x_i|\theta).$$

Now we let the number of observations be denoted by n in the following derivations. Using the density function of a normal distribution with parameter θ we obtain the likelihood function

$$L(\theta; \boldsymbol{x}) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x_i - \mu}{\sigma}\right)^2\right\}.$$

The log-likelihood function is given by

$$l(\theta; \boldsymbol{x}) = \log\, L(\theta; \boldsymbol{x}) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2.$$

The maximum likelihood estimators (MLEs) $\hat{\mu}_{ML}$ and $\hat{\sigma}^2_{ML}$ of μ and σ^2 are obtained by maximizing the likelihood function. This is done by differentiating the log-likelihood functions and setting them to zero. In more detail, we calculate the score functions $S(\theta; \boldsymbol{x})$ w.r.t. μ and σ seperately, and let them equal zero and solve for each parameter:

$$S(\mu) = \frac{\partial}{\partial\mu}\, l(\theta; \boldsymbol{x}) = -\frac{n(\overline{x} - \mu)}{\sigma^2} = 0,$$

where $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. From this we obtain $\hat{\mu}_{ML} = \overline{x}$. Further,

$$S(\sigma) = \frac{\partial}{\partial\sigma}\, l(\theta; \boldsymbol{x}) = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_i - \mu)^2 = 0,$$

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

Then we use the derived formulas in order to obtain the desired parameter estimates for the loaded data. So the data set with 100 observations gives the result $\hat{\mu}_{ML} = 1.275528$ and $\hat{\sigma}_{ML} = 2.005976$.

3.

The function `optim()` minimizes the function by default in R. Thus we will optimize the minus log-likelihood function in order to find the maximum of the function. We will perform two types of algorithms, Conjugate Gradient and BFGS, both with gradient specified and without, to optimize the minus log-likelihood function. It is a better idea to maximize the log-likelihood than maximize the likelihood. This is due to the large values that occurs in the likelihood, which are numerically unstable, so it is preferable to take the logarithm which gives us a better scale to work with. Also, differentiating the log-likelihood function is more computationally convenient, since the product is replaced by a sum (see Question 2.2).

4.

The results of the optimization are presented in Table 1.

From the results in Table 1, we can see that the algorithm converged in all cases, where all the obtained optimal values are the same and correspond to the estimated parameters in Question 2.2. One explanation of why all algorithms find the optimum is because the likelihood function is convex for the normal distribution. Therefore, it is guaranteed that these algorithms will find the optimal values of the parameters.

Without specifying the gradient, the Conjugate Gradient method required 297 function and 45 gradient evaluations for the algorithm to converge, while the BFGS only required 37 function and 15 gradient

Algorithm	Gradient specified	$\hat{\mu}$	$\hat{\sigma}$	Function	Gradient	Time
Conjugate Gradient	No	1.275528	2.005977	297	45	0.01877809 sec
Conjugate Gradient	Yes	1.275528	2.005976	53	17	0.004215956 sec
BFGS	No	1.275528	2.005977	37	15	0.005324841 sec
BFGS	Yes	1.275528	2.005977	39	15	0.009279966 sec

Table 1: *Result of the optimization using the algorithms Conjugate Gradient and BFGS, for the given data set. The algorithms, gradient, optimal values of parameters, number of function and gradient evaluations and time taken are presented.*

Keep in mind that the output of Sys.time() is random and it can have quite a big variance sometimes.

evaluations. Also, the time until convergence was measured by using `Sys.time()`, and we can notice that the BFGS is somewhat faster than Conjugagte Gradient. Even though the difference in time between the algorithms is small, it may have a bigger impact when having a larger data set. All this support the recommendation of choosing the algorithm BFGS in this situation. When we specified the gradient, the number of evaluations was remarkably reduced and time decreased for the Conjugate Gradient algorithm. In this case, it could be reasonable to specify the gradient and not use a finite-difference approximation. On the other hand, there are no big difference when specify the gradient for the BFGS algorithm.

In summary, the gradient adds more information to the optimization and therefore we recommend to specify the gradient if possible. If the gradient is not specified when optimizing, then the BFGS is a better choice in this case.

Appendix

```r
knitr::opts_chunk$set(echo = FALSE)
# R version
RNGversion('3.5.1')
#libraries
library(ggplot2)
# Question 1: Optimizing a model parameter
# 1.1
data <- read.csv2("mortality_rate.csv",sep=";")

data<-cbind(data,LMR=log(data$Rate))
n=dim(data)[1]
set.seed(123456)
id=sample(1:n,floor(n*0.5))
train=data[id,]
test=data[-id,]

# 1.2
myMSE <- function(lambda, pars, iterCounter = FALSE) {
  X <- pars$X
  Y <- pars$Y
  Xtest <- pars$Xtest
  Ytest <- pars$Ytest

  model <- loess(formula = Y~X, enp.target = lambda)

  Xtest_pred <- predict(model, newdata = Xtest)

  MSE <- sum((Ytest - Xtest_pred)^2) / length(Ytest)
  ####### THE FOLLOWING CODE REGARDING THE ITERATION COUNTER WAS WRITTEN BY
  #Arian Barakat AND COPIED FROM THE FILE FunctionCounter_Using_Environments_in_R.pdf #######

  # If we want a iteration counter
  if(iterCounter){
    if(!exists("iterForMyMSE")){
      # Control if the variable exists in the global environemnt,
      # if not, create a variable and set the value to 1. This
      # would be the case for the first iteration
      # We will call the variable 'iterForMyMSE'
      assign("iterForMyMSE",
             value = 1,
             globalenv())
    } else {
      # This part is for the 2nd and the subsequent iterations.
      # Starting of with obtaining the current iteration number
      # and then overwrite the current value by the incremental
      # increase of the current value
      currentNr <- get("iterForMyMSE")
      assign("iterForMyMSE",
             value = currentNr + 1,
             globalenv())
    }
  }
  return(MSE)
}

# 1.3
pars <- list(X=as.matrix(train$Day), Y = as.matrix(train$LMR),
             Xtest = as.matrix(test$Day), Ytest = as.matrix(test$LMR))
lambda <- seq(from = 0.1, to = 40, by = 0.1)

MSE <- sapply(lambda, myMSE, pars)


# 1.4
df_plot <- data.frame(lambda, MSE)
ggplot(df_plot, aes(x = lambda, y = MSE)) +
  geom_point(col = "darkblue") +
  xlab(expression(lambda)) +
  geom_vline(xintercept = 11.7)  # optimal lambda

#min(MSE) #optimal MSE value = 0.131047

# 1.5
#remove iteration variable if already exists
if (exists("iterForMyMSE")) {
  rm("iterForMyMSE")
}

MSE_min <- optimize(myMSE, interval = c(0.1, 40), tol = 0.01, pars = pars, iterCounter = TRUE)
#iterForMyMSE #18 iterations
#MSE_min$minimum #lambda = 10.69361
```

```r
#MSE_min$objective #MSE = 0.1321441

# 1.6
MSE_min2 <- optim(method = "BFGS", par = 35, fn = myMSE, pars = pars)
#MSE_min2$counts[1] #1 function call to myMSE()
#################################################################################


# Question 2: Maximizing likelihood
# 2.1
# Load the data
# Sample from normal distribution with parameters \mu and \sigma
load("data.Rdata")
x <- data

# 2.2
# Derived formulas for the MLE
mu_ml <- function(x){
  mean(x)
}

sigma_ml <- function(x){
  term <- (x - mean(x))^2
  sigma <- sum(term)/length(x)
  return(sqrt(sigma))
}

# Value of MLE
mu <- mu_ml(x)
sigma <- sigma_ml(x)

# 2.3
# Minus log-likelihood function
minus_logL <- function(x, par){
  mu <- par[1]
  sigma <- par[2]
  (length(x)/2)*log(2*pi*sigma^2) + 1/(2*sigma^2) * sum((x - mu)^2)
}

# Gradient (also minus)
gradient <- function(x, par){
  mu <- par[1]
  sigma <- par[2]
  -c((1/sigma)^2 * sum(x - mu),
     -(length(x)/sigma) + (1/sigma)^3 * sum((x - mu)^2))
}

# Optimize with initial parameters mu = 0, sigma = 1.
# Set seed
set.seed(123456)

#---------------------------------------------------------------------
# Conjugate Gradient method
# Start time
start.time <- Sys.time()

# With a finite-difference approximation
optim(par = c(0, 1), fn = minus_logL, gr = NULL, method = "CG", x = x)

# End time
end.time <- Sys.time()
time.taken <- end.time - start.time

#--------------------------------------------------------------------
# Conjugate Gradient method
# Start time
start.time <- Sys.time()

# With a specified gradient
optim(par = c(0, 1), fn = minus_logL, gr = gradient, method = "CG", x = x)

# End time
end.time <- Sys.time()
time.taken <- end.time - start.time

#--------------------------------------------------------------------
# BFGS - With a finite-difference approximation
# Start time
start.time <- Sys.time()

# With a finite-difference approximation
optim(par = c(0, 1), fn = minus_logL, gr = NULL, method = "BFGS", x = x)

# End time
end.time <- Sys.time()
time.taken <- end.time - start.time

#--------------------------------------------------------------------
# BFGS - With a specified gradient
# Start time
start.time <- Sys.time()

# Optimize
optim(par = c(0, 1), fn = minus_logL, gr = gradient, method = "BFGS", x = x)

# End time
end.time <- Sys.time()
time.taken <- end.time - start.time
```