Computer Lab 1
Computational Statistics

Linköpings Universitet, IDA, Statistik

2019/01/23

Kurskod och namn:	732A90 Computational Statistics
Datum:	2019/01/22—2019/01/31 (lab session 23 January 2019)
Delmomentsansvarig:	Krzysztof Bartoszek, Eric Herwin, Sara Johansson
Instruktioner:	This computer laboratory is part of the examination for the Computational Statistics course Create a group report, (that is directly presentable, if you are a presenting group), on the solutions to the lab as a **.PDF** file. Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments. **All R code should be included as an appendix into your report.** A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes. In the report reference **ALL** consulted sources and disclose **ALL** collaborations. The report should be handed in via LISAM (or alternatively in case of problems e–mailed to krzysztof.bartoszek@liu.se or Eric Herwin erihe068@student.liu.se or Sara Johansson sarjo775@student.liu.se), by **23:59 31 January 2019** at latest. Notice there is a final deadline of **23:59 1 April 2019** after which no submissions nor corrections will be considered and you will have to redo the missing labs next year. The seminar for this lab will take place **7 March 2019**. The report has to be written in English.

Question 1: Be careful when comparing

Consider the following two R code snippets

```
x1<-1/3;x2<-1/4
if (x1-x2==1/12){
print("Subtraction is correct")
}else{
print("Subtraction is wrong")
}
```

and

```
x1<-1;x2<-1/2
if (x1-x2==1/2){
print("Subtraction is correct")
}else{
print("Subtraction is wrong")
}
```

1. Check the results of the snippets. Comment what is going on.
2. If there are any problems, suggest improvements.

Question 2: Derivative

From the defintion of a derivative a popular way of computing it at a point x is to use a small ϵ and the formula

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}.$$

1. Write your own R function to calculate the derivative of $f(x) = x$ in this way with $\epsilon = 10^{-15}$.
2. Evaluate your derivative function at $x = 1$ and $x = 100000$.
3. What values did you obtain? What are the true values? Explain the reasons behind the discovered differences.

Question 3: Variance

A known formula for estimating the variance based on a vector of n observations is

$$\mathrm{Var}(\vec{x}) = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right)$$

1. Write your own R function, `myvar`, to estimate the variance in this way.
2. Generate a vector $x = (x_1, \ldots, x_{10000})$ with 10000 random numbers with mean 10^8 and variance 1.
3. For each subset $X_i = \{x_1, \ldots, x_i\}$, $i = 1, \ldots, 10000$ compute the difference $Y_i = \texttt{myvar}(X_i) - \texttt{var}(X_i)$, where $\texttt{var}(X_i)$ is the standard variance estimation function in R. Plot the dependence Y_i on i. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?
4. How can you better implement a variance estimator? Find and implement a formula that will give the same results as `var()`?

Question 4: Linear Algebra

The Excel file "tecator.xls" contains the results of a study aimed to investigate whether a near–infrared absorbance spectrum and the levels of moisture and fat can be used to predict the protein content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry. The worksheet you need to use is "data" (or file "tecator.csv"). It contains data from 215 samples of finely chopped meat. The aim is to fit a linear regression model that could predict protein content as function of all other variables.

1. Import the data set to R
2. Optimal regression coefficients can be found by solving a system of the type $\mathbf{A}\vec{\beta} = \vec{b}$ where $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and $\vec{b} = \mathbf{X}^T\vec{y}$. Compute $\mathbf{A}$ and $\vec{b}$ for the given data set. The matrix $\mathbf{X}$ are the observations of the absorbance records, levels of moisture and fat, while $\vec{y}$ are the protein levels.
3. Try to solve $\mathbf{A}\vec{\beta} = \vec{b}$ with default solver `solve()`. What kind of result did you get? How can this result be explained?
4. Check the condition number of the matrix $\mathbf{A}$ (function `kappa()`) and consider how it is related to your conclusion in step 3.
5. Scale the data set and repeat steps 2–4. How has the result changed and why?

CompStatsLab1

Omkar Bhutra

7 December 2018

Question 1:

Be careful when comparing

```
## [1] "subtraction is wrong"
```

Due to underflow the subtraction is displaying the same number although when the digits are increased using options we can see that the number is actually different. Underflow is the loss of significant digits.

```
## [1] "subtraction is correct"
```

```
## [1] "subtraction is correct"
```

Evaluating the results of the 2 snippets we see that in the first occasion we get the wrong print of the if-else statement.The problem lies to the fact that float numbers that have infinite numbers of decimals can't be represented exactly in the binary system in computers due to memory storage limitation.Using print(x1-x2,digits=16) and print(1/12,digits=16) we will see that the resulting floats are (0.08333333333333331,0.08333333333333333) respectfully and they are not the same causing the condition of unerflow which leads to the failure of the if statement and evaluation of else. We can adress this problem using the "all_equals()" in the if statement instead of "==" to compare the numbers and we will see that the if statement will be executed and the correctly print message will be outputed. The second statement is evaluated correctly and we get the correct print output because 1/2 has finite numbers of decimals so we don't have the occurence of underflow here.

Question 2:

Derivative

```
## [1] 1.1102230246251565
```

```
## [1] 0
```

The value of the derivative for when x=1 is 1.11022 while the value obtained for the derivative when x=100000 is 0. Looking at the equation algebraically it seems that the answer should be 0. But in the first case, when x=1 the addition of a small value epsilon retains its effect compared to the 2nd case and hence produces a different result than the expected value of 0.

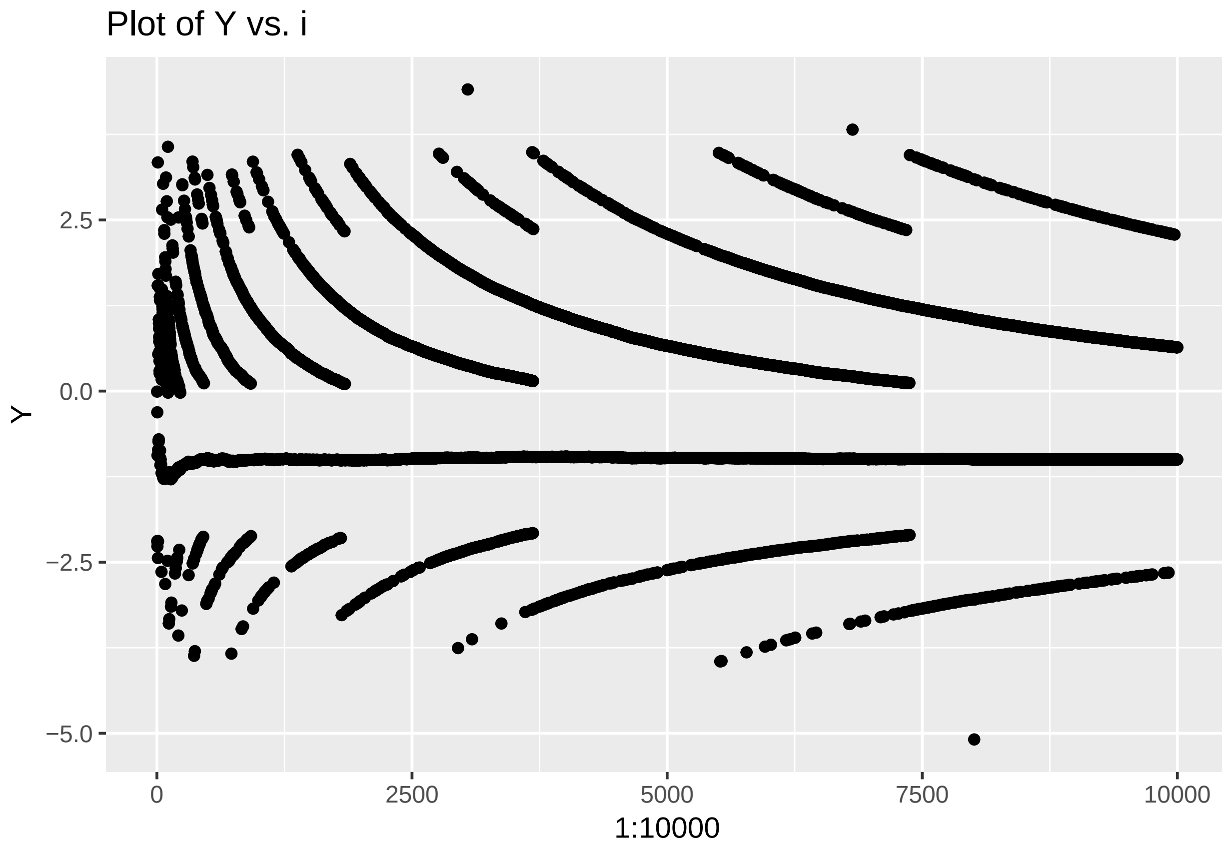
The true value for the function using the function $f(x) = x$ is $f'(x) = \frac{f(x+\epsilon)-f(x)}{\epsilon} = \frac{(x+\epsilon)-x}{\epsilon} = 1$ is always constant with value 1.Regarding the result of the derivative function we see that for $x = 100000$ R doesn't take into account the decimals after a specific number of x and rounds the number to the nearest integer which is 100000 due to underflow occurance so the numeretor of the derivative formula becomes 0 leading finally to 0.When instead $x = 1$ the numerator evaluated is 1.1102230246251565e-15 and the devision with epsilon 10^-15 is just discards the last 15 decimals resulting 1.1102230246251565.

Question 3:

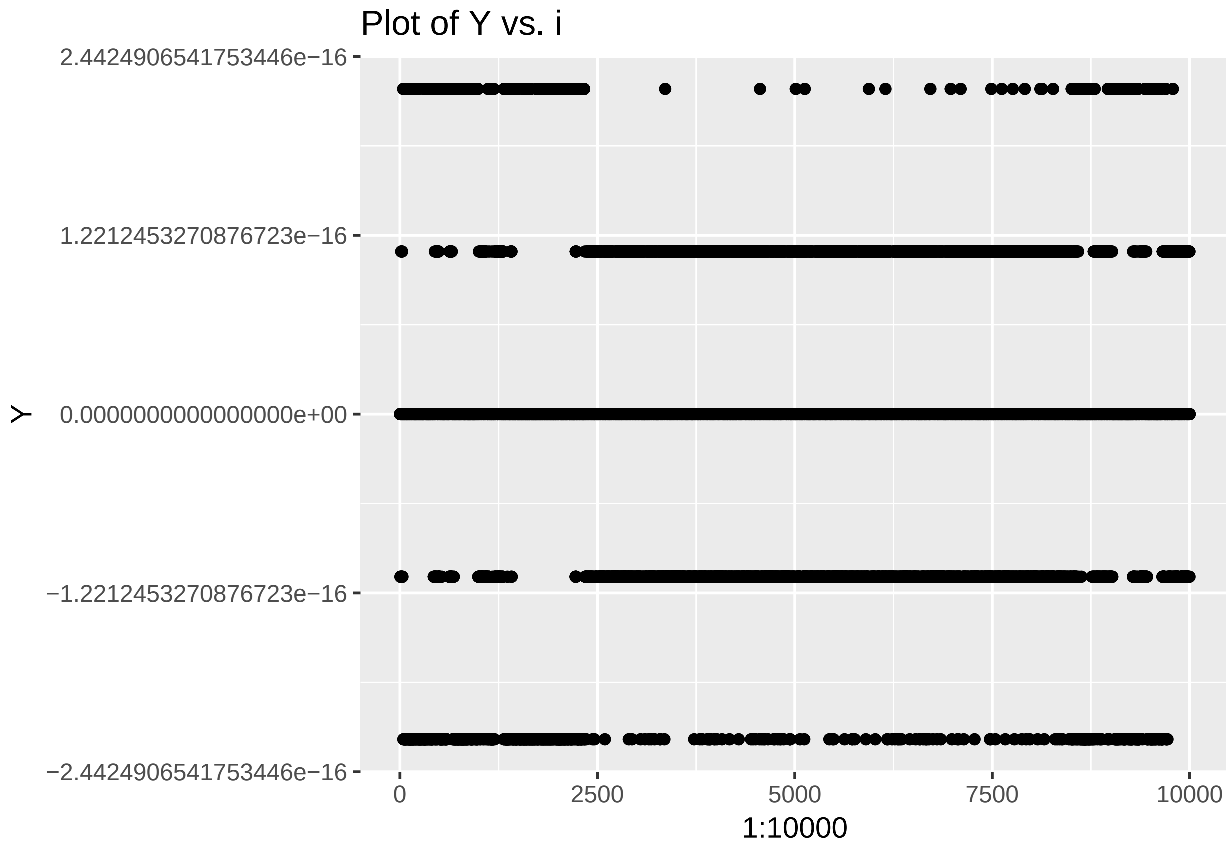
Variance

```
## [1] 1.6385638563856386
```

The plot above shows the dependence Y_i on i with the formula $Var(x) = \frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}$ where μ is the mean. Using the new formula where we center the points arround the mean we see that we have an improvement in the range of the errors and the deviation of the errors is steady and we can see an upper and a lower band with few errors lie beyond these linear bands represented with red in the plot.Also we can observe that the range of the errors is much smaller with means the formula used almost as good as the var() basic function in R.




```
## [1] 0.99969989923081803
```



The function does not perform as well as expected due to the numerical precision of the expression myvar(Xi)-var(Xi) which shows us negative values on most occurances.

Question 4:

Linear Algebra

```
## [1] 1157834236871692.2
```

Error in solve.default(A, b) : system is computationally singular: reciprocal condition number = 7.13971e-17

Printing the number of kappa for the value of A matrix we see that is very big and that implies that the matrix is said to be ill-conditioned a very small change in matrix A will cause a large error in b and makes the solution unstable.

```
##                                [,1]
## Channel1    -110.64200663177356887
## Channel2    -221.18999213628123357
## Channel3     378.00670489934344687
## Channel4    -129.70382900682116656
## Channel5     413.41802724878726849
## Channel6     -79.75199462292130193
## Channel7    -203.00600103836293897
## Channel8      82.79496481288938980
## Channel9    -132.38107625535101874
## Channel10    255.82331130982933587
```

```
## Channel11   -328.60850197387361504
## Channel12   -304.14980672129559025
## Channel13    624.12672247994066765
## Channel14   -298.89902984730480284
## Channel15     40.74336715520891516
## Channel16   -257.53594209754874100
## Channel17    169.23891908084701186
## Channel18    296.66293885907538197
## Channel19   -325.06601012322818178
## Channel20     -3.00774341513279264
## Channel21    554.56043514869395494
## Channel22  -1366.02819464972321839
## Channel23   1860.35645048260312251
## Channel24  -1416.13210988432069826
## Channel25    631.83966192630055048
## Channel26   -112.04395491771877857
## Channel27     17.01671931621323708
## Channel28   -228.93063252299120336
## Channel29    444.27242587767221949
## Channel30   -597.38053691993104621
## Channel31    438.14844153976412144
## Channel32    315.03940879430871291
## Channel33   -349.81437683657878779
## Channel34   -285.91097574921798241
## Channel35    418.58105049764253636
## Channel36    -79.10682360375271571
## Channel37   -305.94133876342527856
## Channel38    284.25434730174231390
## Channel39   -435.56578629077682763
## Channel40    819.74862823138187196
## Channel41   -885.00733142452941138
## Channel42    324.58934112061319865
## Channel43    524.58956696461768843
## Channel44   -583.44189030616257696
## Channel45   -140.17097449229814288
## Channel46    577.23617438118196787
## Channel47   -294.26844675559453890
## Channel48    -68.07512752921009280
## Channel49    -90.49228410711393167
## Channel50    404.14395235715244326
## Channel51   -698.99905269889643478
## Channel52   1258.88337827146801828
## Channel53  -1672.73084007547663532
## Channel54   1486.22991720257186898
## Channel55   -812.36134431546076939
## Channel56    192.49547990397724107
## Channel57    -32.91204662098924416
## Channel58      7.37525455331103430
## Channel59    -88.69071417608546426
## Channel60    344.87690207084136773
## Channel61   -454.35186074457743644
## Channel62    447.62052960726805395
## Channel63   -197.41868472109501909
## Channel64    222.33707920152659199
```

```
## Channel65   -399.25583177909453525
## Channel66    364.86655737276095124
## Channel67   -367.16148297452463112
## Channel68    243.92206215756519327
## Channel69    -76.29483445491032967
## Channel70   -318.19160711413701392
## Channel71    327.66533461201169075
## Channel72   -178.52315275400727046
## Channel73    119.18564986588171450
## Channel74    445.11500447600673169
## Channel75    -20.01273187169612910
## Channel76   -642.75099614710870810
## Channel77    369.48095078598106511
## Channel78    -74.90113656891863059
## Channel79    -23.48543216535520983
## Channel80   -676.86153344610886506
## Channel81   1013.45380290573928050
## Channel82   -889.76231887539347554
## Channel83    403.00656326222281223
## Channel84    424.08483028785201441
## Channel85   -801.09561546783777430
## Channel86    655.01342015748275571
## Channel87    659.18297852899979716
## Channel88  -2150.83256466095554060
## Channel89   1671.80888532636413402
## Channel90    298.69770682030031139
## Channel91   -332.17277624942408920
## Channel92   -487.36897587493234596
## Channel93    278.62773677083617940
## Channel94    201.66273526775202640
## Channel95   -609.50814557014871298
## Channel96    565.28517886262272896
## Channel97   -133.34075951392054549
## Channel98   -368.00872501373430623
## Channel99    238.20159678039942719
## Channel100    24.64181878308056639
## Fat           -1.66664028405493303
## Moisture      -0.93410994774249811
```

This happens because the tolerence returned is larger than the default threshold set by the function solve (argument tolerence) so an error returned and we cannot get a solution.The torrelance is related to conditon number by the function $tolerance = \frac{1}{conditionnumber}$ so in our case $tolerance = \frac{1}{kappa(A)} = 7.425326e - 16$ and it is bigger that the threshold of $7.425326e - 17$ that is set by solve function as we see in the printed error resulting the end of execution of the function. Using the scaled data we where able to solve the linear system and get coefficients for every feature value. Printing the number of kappa again we can see that is still high but much less that the previous used with the unscaled data and we where able to solve the linear system and get coefficient values.

When we scale the data we see that the linear system did not get any better or worse the linear dependences of the column features are still present but we manage to make the value of condition number smaller with scaling.This is happening because If we look at the definition of the condition number $k(A) = ||A|| * ||A^-1||$ and just by making the range of the columns smaller the magnitude got smaller leading to a smaller value of condition number which is below threshold value of solve function and we manage to get the solution.The tolerence now is $tolerance = \frac{1}{kappa(A1)} = \frac{1}{490471518993} = 2.038854e - 12$ which is smaller than the default

$7.425326e-17$ set by solve so now we are able to get a solution.

Apendix

```
knitr::opts_chunk$set(echo = TRUE)
library(dplyr)
library(plotly)
library(ggplot2)
library(xlsx)
library(readxl)
library(boot)
library(kableExtra)
library(knitr)
library(testthat)
options(digits=22)
x1<-1/3;x2<-1/4
if(x1-x2==1/12){
  print("subtraction is correct")
}else{
  print("subtraction is wrong")
}
x1<-1/3;x2<-1/4
if(all.equal((x1-x2),(1/12))){
  print("subtraction is correct")
}else{
  print("subtraction is wrong")
}
x1<-1;x2<-1/2
if(x1-x2==1/2){
  print("subtraction is correct")
}else{
  print("subtraction is wrong")
}
derivative<-function(x){
  f<-function(x){
    return(x)
  }
  epsilion<-10^-15
  x<-(f(x+epsilion)-f(x))/epsilion
  return(x)
}

derivative(x=1)

derivative(x=100000)

set.seed(12345)
myvar<-function(x){
n=length(x)
var<-(1/(n-1))*(sum(x^2)-((1/n)*(sum(x)^2)))
return(var)
}
```

```r
x<-rnorm(n=10000,mean=10^8,sd=sqrt(1))
myvar(x)
Y <- c()
for (i in 1:10000){
options(digits = 22)
Y[i] <- myvar(x[1:i])-var(x[1:i])
}

p1<-ggplot()+ geom_point(aes(1:10000,Y))+ labs(title="Plot of Y vs. i")
p1
set.seed(12345)
varfun <- function(x){
vari <- (1/(length(x) - 1)) * sum((x - mean(x))^2)
return(vari)
}

x<-rnorm(n=10000,mean=10^8,sd=sqrt(1))
varfun(x)

Y <- c()
for (i in 1:10000){
Y[i] <- varfun(x[1:i])-var(x[1:i])
}

p2<-ggplot() + geom_point(aes(1:10000,Y))+ labs(title="Plot of Y vs. i")
p2
tecator = read_excel("tecator.xls",sheet ="data" )
X<-as.matrix(tecator[,c(2:102,104)])
Y<-as.matrix(tecator[,c(103)])
A<-t(X)%*%X
b<-t(X)%*%Y
kappa(A)
#solve(A,b)
tecatorscale <- as.matrix(scale(tecator))
Xscale <- as.matrix(tecatorscale[,-c(1,103)])
yscale <- as.matrix(tecatorscale[,103])
Ascale <- t(Xscale)%*%Xscale
bscale <- t(Xscale)%*%yscale
solve(Ascale,bscale)
```

Computer Lab 2
Computational Statistics

Linköpings Universitet, IDA, Statistik

2019/01/25

Kurskod och namn:	732A90 Computational Statistics
Datum:	2019/01/24—2019/02/07 (lab session 25 January 2019)
Delmomentsansvarig:	Krzysztof Bartoszek and Eric Herwin
Instruktioner:	This computer laboratory is part of the examination for the Computational Statistics course Create a group report, (that is directly presentable, if you are a presenting group), on the solutions to the lab as a **.PDF** file. Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments. **All R code should be included as an appendix into your report.** A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes. In the report reference **ALL** consulted sources and disclose **ALL** collaborations. The report should be handed in via LISAM (or alternatively in case of problems e–mailed to krzysztof.bartoszek@liu.se or Eric Herwin erihe068@student.liu.se or Sara Johansson sarjo775@student.liu.se), by **23:59 7 February 2019** at latest. Notice there is a final deadline of **23:59 1 April 2019** after which no submissions nor corrections will be considered and you will have to redo the missing labs next year. The seminar for this lab will take place **7 March 2019**. The report has to be written in English.

Question 1: Optimizing a model parameter

The file `mortality_rate.csv` contains information about mortality rates of the fruit flies during a certain period.

1. Import this file to `R` and add one more variable `LMR` to the data which is the natural logarithm of `Rate`. Afterwards, divide the data into training and test sets by using the following code:

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

2. Write your own function `myMSE()` that for 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 (parameter `enp.target` in `loess()`) and then predicts the model for `Xtest`. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data. It is defined by the following Equation (for you to implement):

   $$\text{predictive MSE} = \frac{1}{\texttt{length(test)}} \sum_{i\text{th element in test set}} \left(\texttt{Ytest[i]} - \texttt{fYpred(X[i])}\right)^2,$$

   where `fYpred(X[i])` is the predicted value of `Y` if `X` is `X[i]`. Read on `R`'s functions for prediction so that you do not have to implement it yourself.

3. Use a simple approach: use function `myMSE()`, training and test sets with response LMR and predictor Day and the following λ values to estimate the predictive MSE values: $\lambda = 0.1, 0.2, \ldots, 40$

4. Create a plot of the MSE values versus λ and comment on which λ value is optimal. How many evaluations of `myMSE()` were required (read `?optimize`) to find this value?

5. Use `optimize()` function for the same purpose, specify range for search $[0.1, 40]$ and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many `myMSE()` function evaluations were required? Compare to step 4.

6. Use `optim()` function and BFGS method with starting point $\lambda = 35$ to find the optimal λ value. How many `myMSE()` function evaluations were required (read `?optim`)? Compare the results you obtained with the results from step 5 and make conclusions.

Question 2: Maximizing likelihood

The file `data.RData` contains a sample from normal distribution with some parameters μ, σ. For this question read `?optim` in detail.

1. Load the data to `R` environment.

2. Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

3. Optimize the minus log–likelihood function with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log–likelihood?

4. Did the algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

Lab2 Computational Statistics

Andreas Christopoulos Charitos (andch552) Omkar Bhutra (omkbh878)

27 jan 2019

Question 1

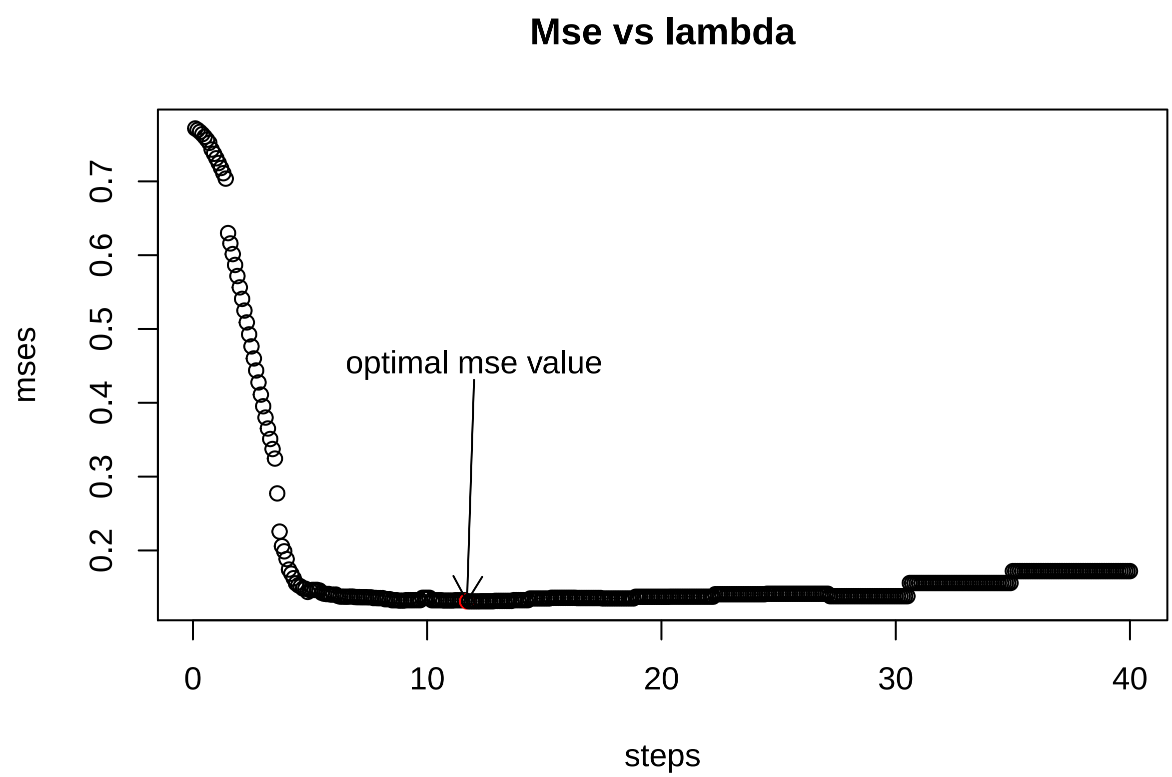
1

```
#importing and diving data
mortality<-read.csv2("mortality_rate.csv")
mortality$LMR<-log(mortality$Rate)

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

```
#defining the function "myMSE"

myMSE<-function(lambda,pars,iterCounter = T){
  model<-loess(pars$Y~pars$X,enp.target = lambda)
  preds<-predict(model,newdata=pars$X_test)
  mse<-sum((pars$Y_test-preds)^2)/length(pars$Y_test)

  # 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)
}
```

```
set.seed(123456)
steps=seq(0.1,40,0.1)
```

```r
mses<-double(length(steps))

mypars<-list(X=train$Day,Y=train$LMR,X_test=test$Day,Y_test=test$LMR)

for(i in steps){
  mses[which(i==steps)]<-myMSE(i,mypars)

}

optimal_l=steps[which.min(mses)]
optimal_mse=min(mses)

plot(x=steps,y= mses,main = "Mse vs lambda",
      col=ifelse(steps==optimal_l,"red","black"))
arrows(optimal_l+0.3, optimal_mse+0.3, x1 = optimal_l, y1 = optimal_mse,
               length = 0.15, angle = 30)
text(optimal_l+0.3,optimal_mse+0.32,labels=c("optimal mse value"))
```




```r
iters1=iterForMyMSE

results1=list("par"=optimal_l,"value"=optimal_mse)

cat("The number of iterations using brute force are :", iters1,"\n",
    "The results from the brute force are :\n")
```

```
## The number of iterations using brute force are : 400
```

```
##  The results from the brute force are :
```

```r
results1
```

```
## $par
## [1] 11.7
##
## $value
## [1] 0.131047
```

5

```r
set.seed(123456)
remove("iterForMyMSE")
xmin <- optimize(myMSE,pars=mypars,
                 interval=c(0.1,40),maximum  = FALSE,tol=0.01)
iters2=iterForMyMSE
cat("The number of iterations using lambda in [0.1,40] and accuracy 0:01 are :", iters2,"\n",
    "The output of optimize function is : \n")
```

```
## The number of iterations using lambda in [0.1,40] and accuracy 0:01 are : 18
##  The output of optimize function is :

## $minimum
## [1] 10.69361
##
## $objective
## [1] 0.1321441
```

We can see comparing the results with the previous step that the number of iterations needed for the algorithm to converge decreased dramatically from 400 to only 18.

6

```r
set.seed(123456)
remove("iterForMyMSE")
xmin1=optim(c(35), myMSE,pars=mypars,
      method = c( "BFGS"))

iters3=iterForMyMSE
cat("The new number of iterations using BFGS algorithm and lambda = 35 are :", iters3,"\n",
    "The output of optimize function is : \n")
```

```
## The new number of iterations using BFGS algorithm and lambda = 35 are : 3
##  The output of optimize function is :

## $par
## [1] 35
##
## $value
## [1] 0.1719996
##
## $counts
## function gradient
##        1        1
##
```

```
## $convergence
## [1] 0
##
## $message
## NULL
```

Comparing again the number of iterations with the previous step we can see that the iterations decreased from 18 to only 3 using BFGS.

The table below summarises the results of the 3 methods used

```r
library(knitr)

sumtable<-data.frame("Brute Force Method"=c(optimal_l,optimal_mse,iters1),
                     "Optimize function"=c(xmin$minimum,xmin$objective,iters2),
                     "Optim function BFGS"=c(xmin1$par,xmin1$value,iters3))
rownames(sumtable)<-c("lambda","mse","iters")

kable(sumtable)
```

	Brute.Force.Method	Optimize.function	Optim.function.BFGS
lambda	11.700000	10.6936107	35.0000000
mse	0.131047	0.1321441	0.1719996
iters	400.000000	18.0000000	3.0000000

As we can see from the table above the BFGS algorithm is able to converge with only 3 iterations which are lower compared to the other 2 methods but with a little higher mse value.

Question 2

2

Maximum Likehihood estimation

The probability function is given by the formula

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

Assuming that the observations from the sample are i.i.d. the likelihood formula is given by

$$f(x_1, x_2, ..., x_n|\mu, \sigma^2) = \prod_{j=1}^{n} (2\pi\sigma^2)^{-(n/2)} exp(-\frac{(x_j-\mu)^2}{2\sigma^2})$$

which is equavalent to :

$$f(x_1, x_2, ..., x_n|\mu, \sigma^2) = (2\pi\sigma^2)^{-(n/2)} exp(-\frac{\sum_{j=1}^{n}(x_j-\mu)^2)}{2\sigma^2})$$

The loglikekihood function is then

$$L = \ln[(2\pi\sigma^2)^{(-n/2)} exp(-(\frac{1}{2\sigma^2})\sum_{j=1}^{n}(x_j - \mu)^2)] =$$

$$\ln((2\pi\sigma^2)^{(-n/2)}) + \ln(exp(-(\frac{1}{2\sigma^2})\sum_{j=1}^{n}(x_j - \mu)^2)) =$$

$$-\frac{n}{2}\ln(2\pi\sigma^2) - (\frac{1}{2\sigma^2})\sum_{j=1}^{n}(x_j - \mu)^2 =$$

$$-\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - (\frac{1}{2\sigma^2})\sum_{j=1}^{n}(x_j - \mu)^2$$

Getting the derivative with respect to μ to we get :

$$\frac{\partial L}{\partial \mu} => -\frac{1}{2\sigma^2}(\sum_{j=1}^{n}(x_j^2 - 2x_j\mu + \mu^2))' =>$$

$$-\frac{1}{2\sigma^2}(\sum_{j=1}^{n}x_j^2 - 2\sum_{j=1}^{n}x_j\mu + n\mu^2)' =>$$

$$-\frac{1}{2\sigma^2}(-2\sum_{j=1}^{n}x_j + 2n\mu) => -\frac{1}{2\sigma^2}(\sum_{j=1}^{n}x_j - n\mu)(-2)(e.1)$$

Setting $(e.1)$ equal to zero we get

$$\frac{\partial L}{\partial \mu} = 0 => \mu = \frac{\sum_{j=1}^{n} x_j}{n}$$

Which is the estimator for the parameter $\hat{\mu}$

Getting the derivative with respect to σ we get :

$$\frac{\partial L}{\partial \sigma} => -\frac{n}{2}\frac{1}{\sigma^2}2\sigma - \frac{1}{2}\sum_{j=1}^{n}(x_j - \mu)^2\frac{\theta L}{\theta \sigma}(\sigma^2)^{-1} => -\frac{n}{\sigma} - \frac{1}{2}\sum_{j=1}^{n}(x_j - \mu)^2\frac{-2}{(\sigma^3)} => -\frac{n}{\sigma} + \sum_{j=1}^{n}(x_j - \mu)^2\frac{1}{(\sigma^3)}(e.2)$$

Setting $(e.2)$ equal to zero we get

$$\frac{\partial L}{\partial \sigma} = 0 => \sum_{j=1}^{n}(x_j - \mu)^2 = \frac{n\sigma^3}{\sigma} => \sigma^2 = \frac{\sum_{j=1}^{n}(x_j - \mu)^2}{n} => \sigma = \sqrt{\frac{\sum_{j=1}^{n}(x_j - \mu)^2}{n}}$$

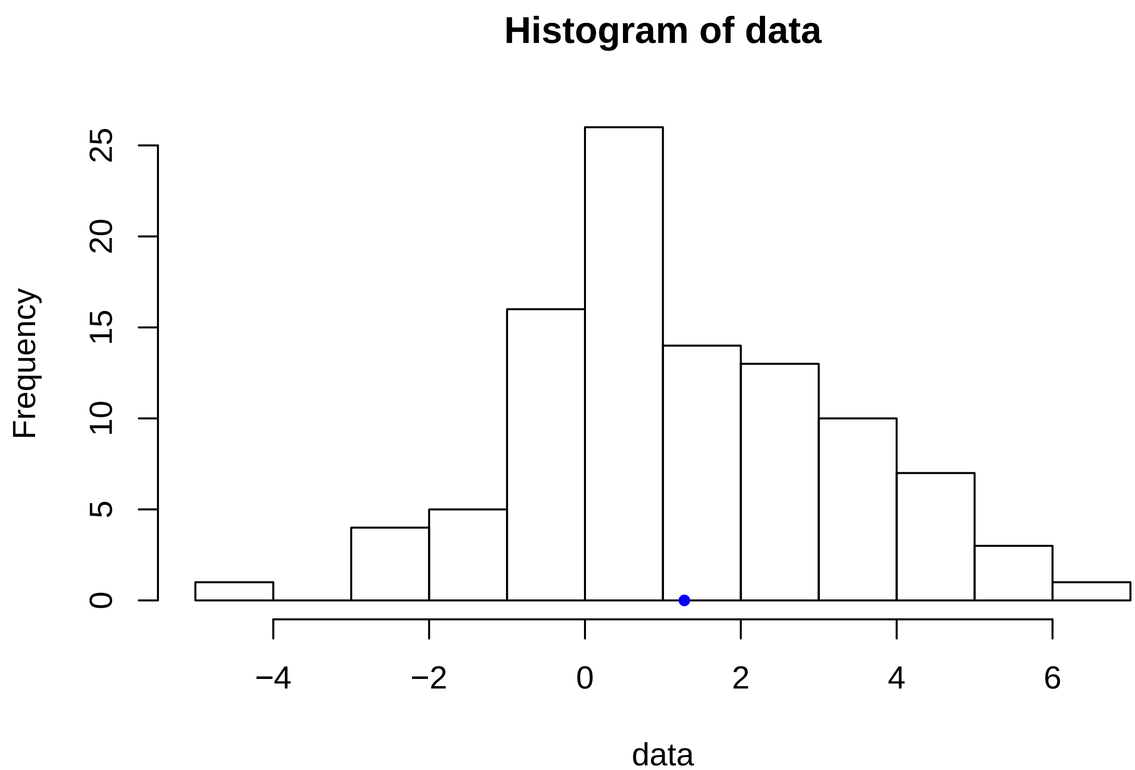
We can use the obtained formulas to obtain the mean $\hat{\mu}$ and variance $\hat{\sigma}$ analytically.

```r
#load data
load("data.RData")

#make estimators function
estimators<-function(data){
  n<-length(data)
  mean<-sum(data)/n
  sigma<-sum((data-mean)^2)/n
  return(c("mean"=mean,"sd"=sqrt(sigma)))
```

```
  }


res=estimators(data)

cat("==========================================================\n",
    "The obtained mean is :",res["mean"],"and the sd is :",res["sd"])
```

```
## ==========================================================
##  The obtained mean is : 1.275528 and the sd is : 2.005976
```

```
hist(data)
points(res["mean"],0,col="blue",pch=20,
       main="Histogram of data")
```



The above histogram shows the distribution of our data and the blue point is indicating the mean value.

3

First we optimize minus loglikehood without specifing gradient

```
loglike<-function(args,x){

  n<-length(x)
  logminus<- (-n*log(2*pi*args[2]^2)/2)-(sum((args[1]-x)^2)/(2*args[2]^2))
  logminus<- -1*logminus
  return(logminus)
```

```r
}


#with BFGS
myres1<-optim(c(0,1),fn=loglike,x=data,method = "BFGS")

#with Conjugate Gradient
myres2<-optim(c(0,1),fn=loglike,x=data,method = "CG")


cat("===================================\n",
    "The output of optim with BFGS is :\n")
```

```
## ===================================
##  The output of optim with BFGS is :
```

```r
myres1
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       41       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
cat("\n")
```

```r
cat("=================================================\n",
    "The output of optim with Conjugate Gradient is :\n")
```

```
## =================================================
##  The output of optim with Conjugate Gradient is :
```

```r
myres2
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      208       35
##
## $convergence
## [1] 0
```

```
##
## $message
## NULL
```

Second we optimize minus loglikehood specifing gradient

The gradient with respect to μ is given by the formula (e.1) we calculated erlier $\frac{\partial L}{\partial \mu} = \frac{\sum_{j=1}^{n} x_j - n\mu}{\sigma^2}$ and the derivative with respect to σ is given by the formula (e.2) $\frac{\partial L}{\partial \sigma} = -\frac{n}{\sigma} + \frac{\sum_{j=1}^{n}(x_j-\mu)}{\sigma^3}$

```
gradient<-function(args,x){
  n<-length(x)
  gr_mu<- (sum(x)-n*args[1])/(args[2]^2)

  gr_sigma<- sum((x-args[1])^2)/args[2]^3-n/args[2]

  return(c(-gr_mu,-gr_sigma))
}

#with BFGS
myres3<-optim(c(0,1),fn=loglike,gr=gradient,x=data,method = "BFGS")

#with Conjugate Gradient
myres4<-optim(c(0,1),fn=loglike,gr=gradient,x=data,method = "CG")


cat("==================================================\n",
    "The output of optim with BFGS with gradient is :\n")
```

```
## ==================================================
##  The output of optim with BFGS with gradient is :
```

```
myres3
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       39       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
cat("\n")
```

```
cat("==============================================================\n",
    "The output of optim with Conjugate Gradient with gradient is :\n")
```

```
## ==============================================================
##  The output of optim with Conjugate Gradient with gradient is :
```

```
myres4
```

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

As we can see loglikelihood is monotonically increasing function and is has the same relations of order as the likelihood $p(x|\theta_1) > p(x|\theta_2) <=> \ln(p(x|\theta_1)) > \ln(p(x|\theta_2))$ so maximizing likelihood is equivalent to maximizing log likelihood.The reason why we use $ln(x)$ is for computational convenience as we see in the calculations of loglikehood we did before.Using $ln()$ we where able to discard the exponent and also use the $ln(ab) = \ln(a) + \ln(b)$ we manage to replace multiplication with summation which is more coninient and we are able to maximize just by setting the derivatives to 0.

4

```
library(knitr)

sumdat<-data.frame("BFGS_without_gr"=c(myres1$par[1],myres1$par[2],
                                       myres1$counts[1],myres1$counts[2],myres1$convergence),
                   "CG_without_gr"=c(myres2$par[1],myres2$par[2],
                                     myres2$counts[1],myres2$counts[2],myres2$convergence),
                   "BFGS_with_gr"=c(myres3$par[1],myres3$par[2],
                                    myres3$counts[1],myres3$counts[2],myres3$convergence),
                   "CG_with_gr"=c(myres4$par[1],myres4$par[2],
                                  myres4$counts[1],myres4$counts[2],myres4$convergence)
                   )

rownames(sumdat)<-c("mean","sd","iterations function","iterations gradient","convergence")
sumdat[5,]<-ifelse(sumdat[5,]==0,"yes","no")

kable(sumdat)
```

	BFGS_without_gr	CG_without_gr	BFGS_with_gr	CG_with_gr
mean	1.27552755151932	1.27552771909709	1.27552755040258	1.27552759112531
sd	2.00597696486639	2.00597650338868	2.00597654945241	2.00597647249389
iterations function	41	208	39	53
iterations gradient	15	35	15	17
convergence	yes	yes	yes	yes

The table provides summary information using the 2 algorithms ("BFGS","CG") with and without specifing gradient function in optim function. All the algorihms converge as we can see and the results are almost identical regarding the mean and the sd.The settings that we would recommend are using BFGS algorithm with gradient because as we can see the number of iterations for both function and gradient are lower compared with the other settings.