Lab2

Martynas Lukosevicius, Alejo Perez Gomez, Zahra Jalil Pour

10/11/2020

Question 1

1

```
interpolator <- function(a, x){
  X <- as.matrix(c(1, x, x^2), ncol = 1 )
  return(as.vector(a %*% X))
}

SSE <- function(x, a, method1, method2){
  real <- sapply(x, method1)
  pred <- sapply(x, method2, a = a)
  return(sum((real-pred)^2))
}

optimiser <- function(x, method){
  aInit <- c(0, 0, 0)
  res <- optim(aInit, SSE, x = x, method1 = method, method2 = interpolator)
  return(res$par)
}
```

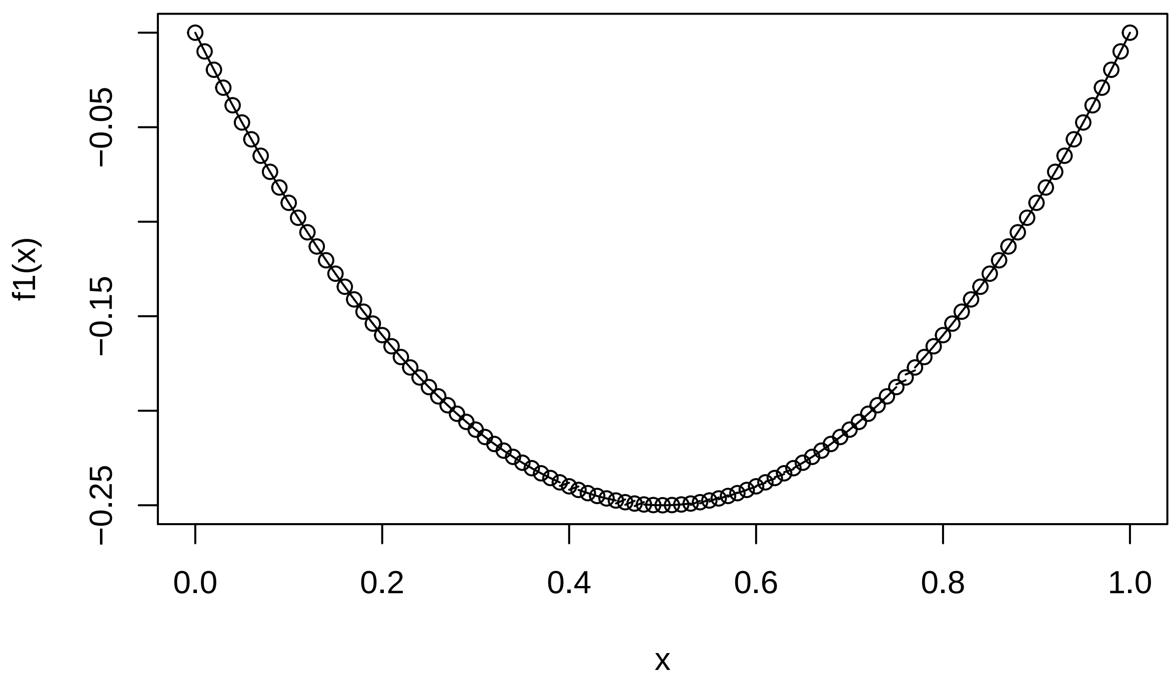
2

```
aproximate <- function(n, method){
  scale <- 1/n
  midVal <- scale / 2
  result <- list()
  for (i in 1:n) {
    x <- c((scale*i)-scale, (scale * i) - midVal, scale * i)
    result <- append(result, list(optimiser(x, method)))
  }
  return(result)
}
```

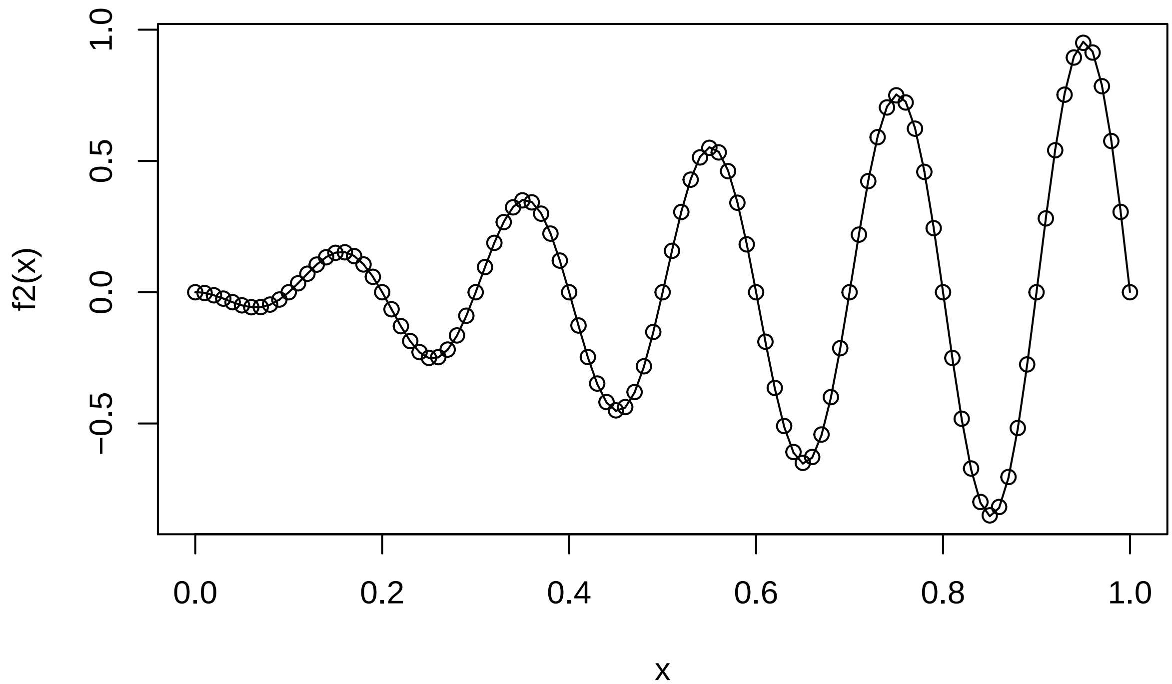
3

```
f1 <- function(x){
  return(-x * (1-x))
}


x <- seq(0, 1, by = 0.01)
```

```r
plot(x,f1(x))

test <- aproximate(100, f1)
scale <- 1/length(test)
for (i in 1:length(test)) {
  x <- seq((scale*i)- scale, scale*i, by = 0.01/length(test))
  yint <- sapply(x, interpolator, a = test[[i]])
  lines(x,yint)
}
```




```r
f2 <- function(x){
  return(-x * sin(10 * pi * x))
}

x <- seq(0, 1, by = 0.01)
plot(x,f2(x))


test <- aproximate(100, f2)
scale <- 1/length(test)
for (i in 1:length(test)) {
  x <- seq((scale*i)- scale, scale*i, by = 0.01/length(test))
  yint <- sapply(x, interpolator, a = test[[i]])
  lines(x,yint)
}
```



Question 2

1

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

2

$$y \sim N(\mu, \sigma^2)$$

$$L(p(\mu, \sigma^2|y)) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} = \frac{1}{(\sqrt{2\pi\sigma^2})^n} e^{-\sum_{i=1}^{n} \frac{(y_i-\mu)^2}{2\sigma^2}}$$

$$lnL(p(\mu, \sigma|y)) = -\frac{n}{2} ln(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i-\mu)^2}{2\sigma^2}$$

$$\frac{\partial lnL(p(\mu, \sigma|y))}{\partial \mu} = -\frac{1}{2\sigma^2} \frac{\partial(\sum y_i^2 - 2\mu \sum y_i + n\mu^2)}{\partial \mu} = -\frac{1}{2\sigma^2}(0 - 2\sum y_i + 2n\mu) = \frac{\sum y_i - n\mu}{\sigma^2}$$

$$MLE => \frac{\sum y_i - n\mu}{\sigma^2} = 0$$

$$\hat{\mu}_{MLE} = \frac{\sum y_i}{n}$$

$$\frac{\partial lnL(p(\mu, \sigma|y))}{\partial \sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^{n} (y_i - \mu)^2$$

$$MLE => -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(y_i - \mu)^2 = 0$$

$$\hat{\sigma}^2_{MLE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu)^2$$

```
meanEst <- mean(data)
print(meanEst)
```

```
## [1] 1.275528
```

```
sqrt((sum((data-meanEst)^2))/ length(data))
```

```
## [1] 2.005976
```

3

$$lnL(p(\mu, \sigma|y)) = -\frac{n}{2}ln(2\pi\sigma^2) - \sum_{i=1}^{n}\frac{(y_i - \mu)^2}{2\sigma^2}$$

$$\frac{\sum(y-\mu)^2}{2\sigma^2} = \frac{\sum y^2 - 2(\sum y)\mu + n\mu^2}{2\sigma^2} = \frac{\sum(\mu - y_i)}{\sigma^2}$$

```
minusLogLikelihood <- function(x){
  n <- length(data)
  data <- data
  mu <- x[1]
  sigma <- x[2]
  part1 <-  ((n/2) * log(2*pi * (sigma^2)))
  part2 <- (sum((data - mu)^2)) / (2 * sigma^2)
  return(part1 + part2)
}

gradient <- function(x){
  mu <- x[1]
  sigma <- x[2]
  n <- length(data)
  gMu <- sum(mu - data)/(sigma^2)
  gSigma <- (n/sigma) - ((1/(sigma^3)) * sum((data-mu)^2))
  return(c(gMu,gSigma))
}

init <- c(0, 1)
test1 <- optim(c(0, 1), minusLogLikelihood,  method = "CG")
test2 <- optim(c(0, 1), minusLogLikelihood, gr = gradient,  method = "CG")
test3 <- optim(c(0, 1), minusLogLikelihood, method = "BFGS")
test4 <- optim(c(0, 1), minusLogLikelihood, gr = gradient, method = "BFGS")
```

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

```
mm = matrix(1:20, ncol=5, dimnames=list(c("minus loglikelihood CG", "minus loglikelihood CG gradient","m
mm[1, ] <- c(ifelse(test1$convergence == 0, "Yes", "No"), test1$par[1], test1$par[2], test1$counts[1], t
mm[2, ] <- c(ifelse(test2$convergence == 0, "Yes", "No"), test2$par[1], test2$par[2], test2$counts[1], t
mm[3, ] <- c(ifelse(test3$convergence == 0, "Yes", "No"), test3$par[1], test3$par[2], test3$counts[1], t
mm[4, ] <- c(ifelse(test4$convergence == 0, "Yes", "No"), test4$par[1], test4$par[2], test4$counts[1], t
as.table(mm)
```

```
##                                    converge mean             variance
## minus loglikelihood CG             Yes      1.27552771909709 2.00597650338868
## minus loglikelihood CG gradient    Yes      1.27552759112531 2.00597647249389
## minus loglikelihood BFGS           Yes      1.27552755151932 2.00597696486639
## minus loglikelihood BFGS gradient  Yes      1.27552755040258 2.00597654945241
##                                    n. of functions n. of gradients
## minus loglikelihood CG             208             35
## minus loglikelihood CG gradient    53              17
## minus loglikelihood BFGS           41              15
## minus loglikelihood BFGS gradient  39              15
```

Questions

- Do we have to estimate for sigma squared or sigma
- Our parabolic function approximation looks weird, are we missing sth?
- are our partial derivatives well calculated?, therefore MLE estimates for mu and sigma?
- Are the gradients of our function gradient well expressed? minus sign?
- Our optim function calls in #3 are returning such mismatched values, one parameter huge and the other really small, our parameters are not similar as the estimates
  - How should we apply this optim, to minimaze or maximize MLL function?