Computer Lab 2

Martynas Lukosevicius, Alejo Perez Gomez, Zahra Jalil Pour

10/11/2020

Question 1: Optimizing parameters

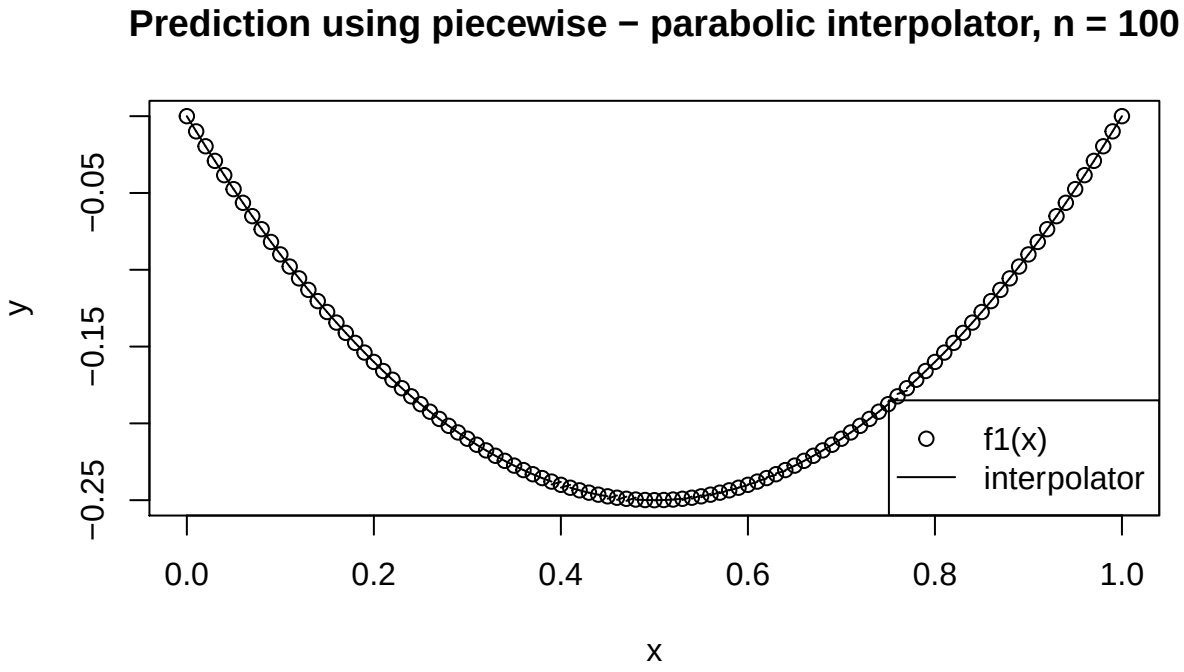
1

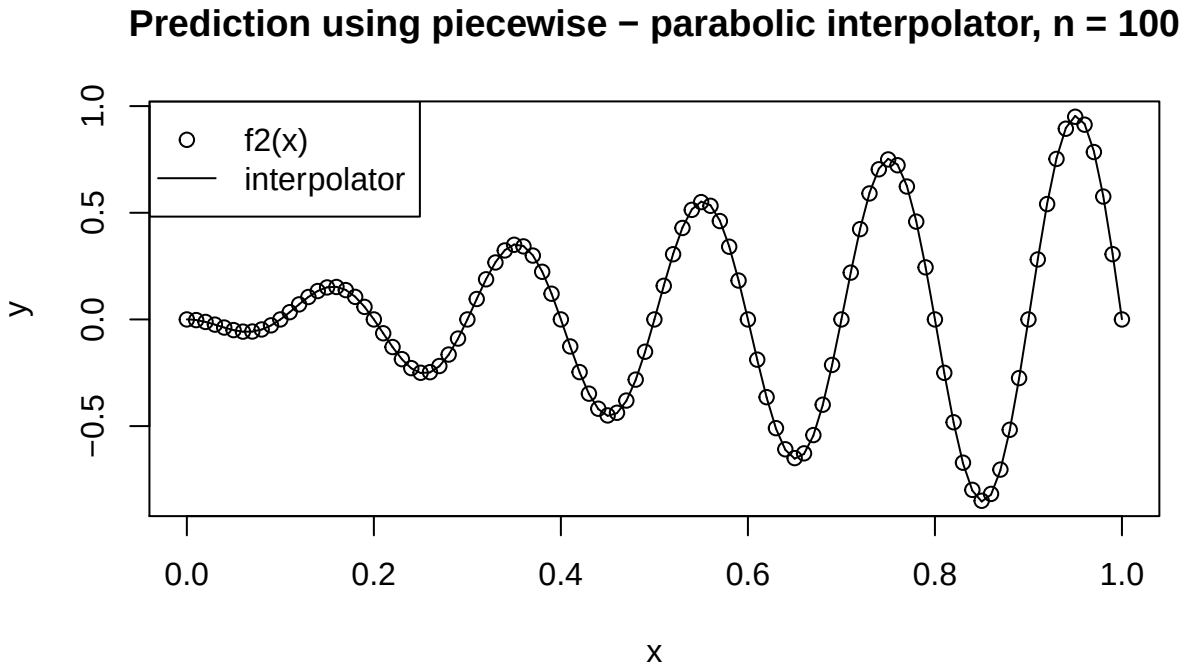
```
interpolator <- function(a, x){
  X <- as.matrix(c(1, x, x^2), ncol = 1 )
  return(as.vector(a %*% X))
}

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





From graphs we can see that piecewice - parabolic interpolator predicted values almost without any error.

Question 2

1

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

2

The sampled data is normally distributed with μ and σ

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

Likelihood of the model:

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

log - likelihood of the model:

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

MLE estimators are partial derivatives of - log-likelihood

$\hat{\mu}_{MLE}$ estimator:

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

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

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

$\hat{\sigma}_{MLE}$ estimator:

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

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

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

mean	variance
1.276	2.006

3

```
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)
}
```

The natural logarithm is a monotonically increasing function. If one value of x increases the value of y related to it will increase too and it will ensure us max log likelihood occurs at the same point where max likelihood function occurs. Also it is easier and simpler to work with log instead of original likelihood. Taking derivative from log is simpler than original likelihood that are often exponential functions.

4

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

We would recommend to use BFGS with specified gradient, because it required to evaluate less functions and gradients as a result its faster.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
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)
}
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)
}
f1 <- function(x){
  return(-x * (1-x))
}

x <- seq(0, 1, by = 0.01)
plot(x,f1(x), main="Prediction using piecewise - parabolic interpolator, n = 100", ylab = "y")

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)
}

legend("bottomright", legend=c("f1(x)", "interpolator"), lty=c(NA,1),pch=c(1,NA))

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

x <- seq(0, 1, by = 0.01)
plot(x,f2(x), main="Prediction using piecewise - parabolic interpolator, n = 100", ylab = "y")


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)
}

legend("topleft", legend=c("f2(x)", "interpolator"), lty=c(NA,1),pch=c(1,NA))
load("data.RData")
meanEst <- mean(data)
sigma <- sqrt((sum((data-meanEst)^2))/ length(data))
res <- matrix(c(meanEst, sigma), nrow = 1)
knitr::kable(res, col.names = c("mean", "variance"), digits = 3)



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

```r
  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")


mm = matrix(1:20, ncol=5, dimnames=list(c("minus loglikelihood CG",
                                          "minus loglikelihood CG gradient",
                                          "minus loglikelihood BFGS" ,
                                          "minus loglikelihood BFGS gradient"),
                                        c("converge",
                                          "mean",
                                          "variance",
                                          "n. of functions",
                                          "n. of gradients")))

mm[1, ] <- c(ifelse(test1$convergence == 0, "Yes", "No"),
             test1$par[1],
             test1$par[2],
             test1$counts[1],
             test1$counts[2])
mm[2, ] <- c(ifelse(test2$convergence == 0, "Yes", "No"),
             test2$par[1],
             test2$par[2],
             test2$counts[1],
             test2$counts[2])
mm[3, ] <- c(ifelse(test3$convergence == 0, "Yes", "No"),
             test3$par[1],
             test3$par[2],
             test3$counts[1],
             test3$counts[2])
mm[4, ] <- c(ifelse(test4$convergence == 0, "Yes", "No"),
             test4$par[1],
             test4$par[2],
             test4$counts[1],
             test4$counts[2])
knitr::kable(mm, digits = 2)
```