Lab Assignment 2

Shwetha Vandagadde, Suhani Ariga and Hoda Fakharzadeh

11/16/2020

Question 1: Optimizing parameters

1

```
parabolic_inter = function(a,x){
  a0 = a[1]
  a1 = a[2]
  a2 = a[3]
  res = a0 + a1*x + a2*(x^2)
  return(res)
}
sum_square_error = function(a,x,func){
  x0 = x[1]
  x1 = x[2]
  x2 = x[3]
  res = sum((func(x0)-parabolic_inter(a,x0))^2,(func(x1)-parabolic_inter(a,x1))^2,
            (func(x2)-parabolic_inter(a,x2))^2)
  return(res)
}
opt = function(x,func){
  a = c(0,0,0)
  res = optim(a,sum_square_error,x = x,func = func)
  res = res$par
  return(res)
}
```
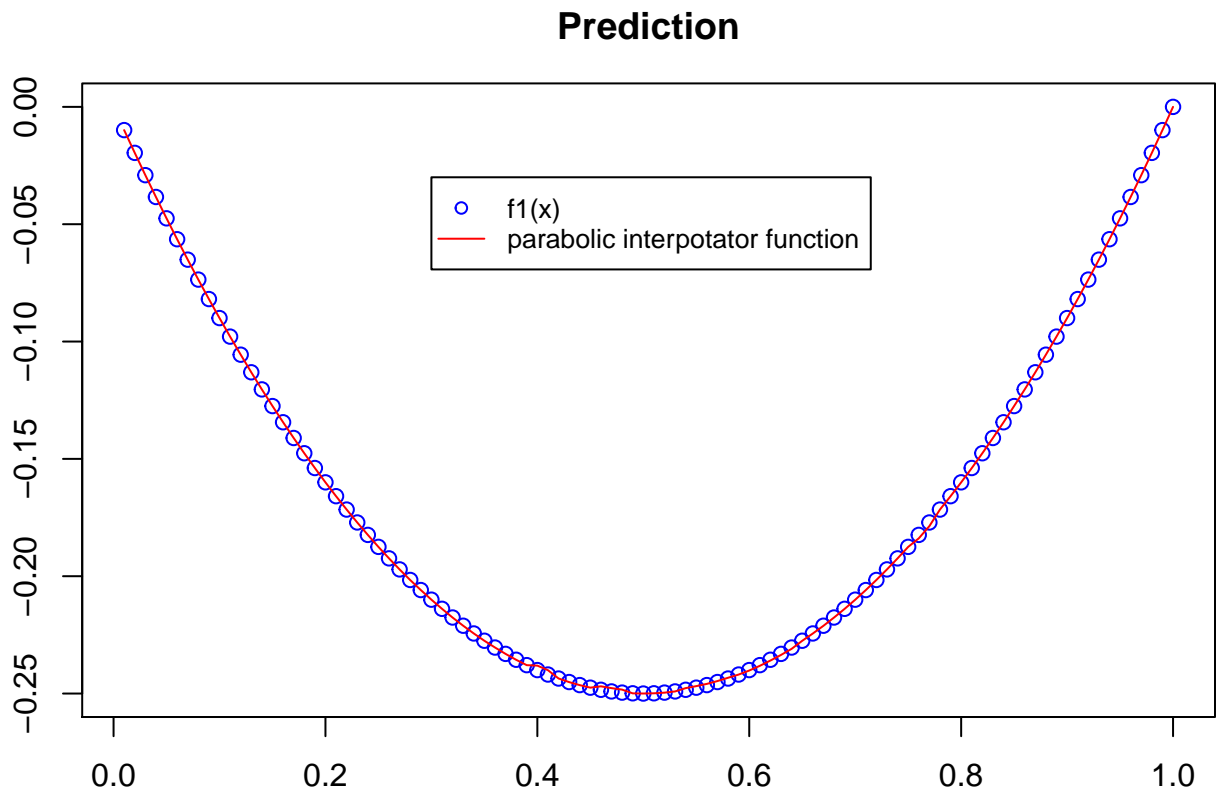
2

```
approx = function(n,func){
 interval = 1/n
 midpoint = 1/(2*n)
 res = data.frame()
 for(i in 1:n){
   end_point = i/n
   x = c((end_point - interval),(end_point - midpoint),end_point)
   res = append(res,as.data.frame(opt(x,func)))
 }
```
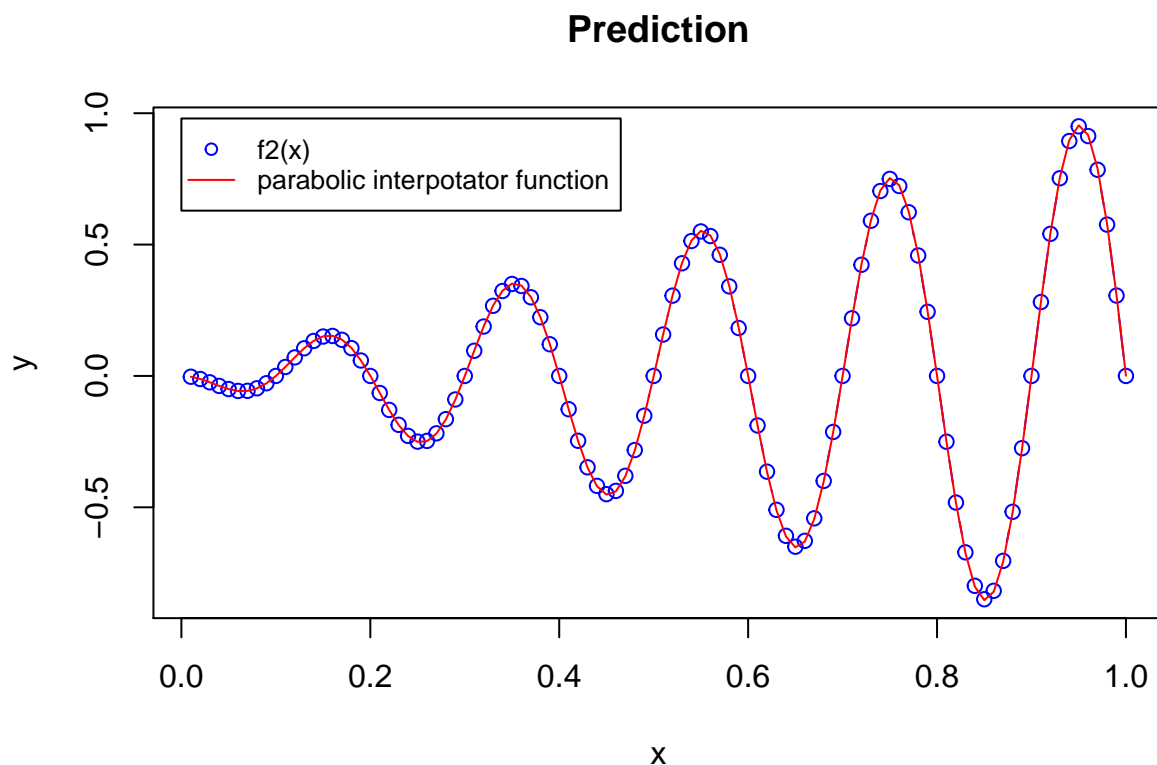
```
  return(res)
}
```

3

Comparing the result from f1(x) and parabolic interpolator function for 100 subintervals.

Prediction

Comparing the result from f2(x) and piecewise parabolic interpolator function for 100 subintervals.

Prediction

Observing the above two plots , we can say that the prediction from piecewise parabolic interpolater was fair as the predictions has very low errors.

Question 2: Maximizing likelihood

1. Load the data to R environment.

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

2. A sample from normal distribution with some parameters μ, σ.

The probability density function of normal distribution is:

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

We have the following n = 100 i.i.d observations:

$$x_1, x_2, ..., x_n$$

The likelihood function:

$$f(x_1, x_2, ..., x_n | \sigma, \mu) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} = (\frac{1}{\sigma\sqrt{2\pi}})^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}$$

The log likelihood function:

$$log(f(x_1, x_2, ..., x_n | \sigma, \mu)) = log((\frac{1}{\sigma\sqrt{2\pi}})^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2})$$

3

$$= nlog\frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$= -\frac{n}{2}log(2\pi) - nlog\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

Let's call

$$log(f(x_1, x_2, ..., x_n|\sigma, \mu))$$

as L and derivation of the log-likelihood by μ by setting partial derivatives to zero

$$\frac{\partial L}{\partial \mu} = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2 \mid \mu = 0$$

Solve this equation, we get

$$\frac{1}{2\sigma^2}\sum_{i=1}^{n}(2\hat{\mu} - 2x_i) = 0$$

Because σ^2 should be larger than 0, Mean of the sample

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

Similarly, derivation of the log-likelihood by σ

$$\frac{\partial L}{\partial \sigma} = -\frac{n}{\sigma} + \sum_{i=1}^{n}(x_i - \mu)^2\sigma^{-3} = 0$$

Variance of the sample

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

Standard Deviation of the sample

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

```r
mu_hat =  sum(data)/length(data)
# formula for sigma
sigma_hat = sqrt(sum((data - mu_hat)^2)/length(data))
cat(paste("Mean of data", mu_hat))
```

Mean of data 1.27552760103638

```r
cat(paste("Standard Deviation of data", sigma_hat))
```

Standard Deviation of data 2.00597647210603

3. Minus log–likelihood function with initial parameters μ = 0, = 1.

```r
minusloglikenormal = function(theta, x=data) {
  mu<-theta[1]                    # mean of normal distribution
  sigma2<-theta[2]                # variance of normal distribution
  n = length(x)
  a1 = (n/2)*log(2*pi)-(n/2)*log(sigma2)
  a2 = (sum((x - mu)^2)) / (2 * sigma2^2)
  ans = a1+a2
  return(ans)
}
minusloglikenormal(c(0,1),data)
```

```
[1] 374.4395
```

Gradient method

```r
gradient = function(theta, x = data) {
  mu<-theta[1]
  sigma2<-theta[2]
  n = length(x)
  dmu = sum(mu - x)/sigma2^2
  dsig = (n/sigma2) - sum((x - mu)^2)/sigma2^3
  gr = c(dmu,dsig)
  return(gr)
}
gradient(c(0,1),data)
```

```
[1] -127.5528 -465.0912
```

```r
# Conjugate Gradient method without gradient method:
opt_cg <- optim(par = c(0, 1), fn = minusloglikenormal, method = "CG")

# Conjugate Gradient method without gradient method:
opt_cg_gr <- optim(par = c(0, 1), fn = minusloglikenormal, gr = gradient,  method = "CG")

# BGFS method without gradient method:
opt_bfgs <- optim(par = c(0, 1), fn = minusloglikenormal, method = "BFGS")

# BGFS method with gradient method:
opt_bfgs_gr <- optim(par = c(0, 1), fn = minusloglikenormal, gr = gradient, method = "BFGS")
```

It is more convenient to maximize the log of the likelihood function. Because the logarithm is monotonically increasing function of its argument, maximization of the log of a function is equivalent to maximization of the function itself. Taking the log not only simplifies the subsequent mathematical analysis, but it also helps numerically because the product of a large number of small probabilities can easily underflow the numerical precision of the computer, and this is resolved by computing instead the sum of the log probabilities.

4. Algorithms converge

```r
cg1 = c(opt_cg$convergence, opt_cg$par, opt_cg$value, opt_cg$counts[1],
        opt_cg$counts[2])
cg2 = c(opt_cg_gr$convergence, opt_cg_gr$par, opt_cg_gr$value,
        opt_cg_gr$counts[1], opt_cg_gr$counts[2])
```

```r
bfgs1 = c(opt_bfgs$convergence, opt_bfgs$par, opt_bfgs$value,
          opt_bfgs$counts[1], opt_bfgs$counts[2])
bfgs2 = c(opt_bfgs_gr$convergence, opt_bfgs_gr$par, opt_bfgs_gr$value,
          opt_bfgs_gr$counts[1], opt_bfgs_gr$counts[2])
df <- as.data.frame(matrix(c(cg1, cg2, bfgs1, bfgs2), nrow=6))
rownames(df) <- c("Convergence", "Mean", "Standard Deviation", "minusloglikelihood",
                  "function evaluation", "gradient evaluation")
colnames(df) <- c("CG", "BFGS", "CG with gradient", "BFGS with gradient")
knitr::kable(x = df, caption = "Comparison table of the different algorithms")
```

Table 1: Comparison table of the different algorithms

	CG	BFGS	CG with gradient	BFGS with gradient
Convergence	1.0000	0.0000	0.0000	0.0000
Mean	122.8803	127.5528	-17266.3035	127.5528
Standard Deviation	628.7519	466.0912	54456.2258	466.0912
minusloglikelihood	-228.4387	-211.6542	-448.3364	-211.6542
function evaluation	201.0000	64.0000	17.0000	40.0000
gradient evaluation	101.0000	5.0000	16.0000	2.0000

Appendix

```r
knitr::opts_chunk$set(echo = TRUE, comment = NA)
parabolic_inter = function(a,x){
  a0 = a[1]
  a1 = a[2]
  a2 = a[3]
  res = a0 + a1*x + a2*(x^2)
  return(res)
}
sum_square_error = function(a,x,func){
  x0 = x[1]
  x1 = x[2]
  x2 = x[3]
  res = sum((func(x0)-parabolic_inter(a,x0))^2,(func(x1)-parabolic_inter(a,x1))^2,
            (func(x2)-parabolic_inter(a,x2))^2)
  return(res)
}
opt = function(x,func){
  a = c(0,0,0)
  res = optim(a,sum_square_error,x = x,func = func)
  res = res$par
  return(res)
}
approx = function(n,func){
 interval = 1/n
 midpoint = 1/(2*n)
 res = data.frame()
 for(i in 1:n){
```

```r
    end_point = i/n
    x = c((end_point - interval),(end_point - midpoint),end_point)
    res = append(res,as.data.frame(opt(x,func)))
 }
 return(res)
}
f1 = function(x){
  res = -x *(1-x)
  return(res)
}
f2 = function(x){
  res = -x * sin(10*pi*x)
  return(res)
}
x = c()
for(i in 1:100){x[i] = i/100}

approx1 = approx(100,f1)
interpolate_result1 = c()
for(i in 1:length(approx1)){
  a = as.vector(approx1[[i]])
  interpolate_result1 = append(interpolate_result1,parabolic_inter(a,x[i]))
}
par(mar = c(3,3,3,0))
plot(x,f1(x),xlab = "x",ylab = "y",col = "blue", type = "b")
lines(x,interpolate_result1,col = "red")
title("Prediction")
legend(0.3, -0.03, legend=c("f1(x)", "parabolic interpotator function"),
       col=c("blue", "red"), pch=c(1,NA),lty=c(0,1), cex=0.8)

approx2 = approx(100,f2)
interpolate_result2 = c()
for(i in 1:length(x)){
  a = as.vector(approx2[[i]])
  interpolate_result2 = append(interpolate_result2,parabolic_inter(a,x[i]))
}
plot(x,f2(x),xlab = "x",ylab = "y",col = "blue", type = "b")
lines(x,interpolate_result2,col = "red")
title("Prediction")
legend(0, 0.98, legend=c("f2(x)", "parabolic interpotator function"),
       col=c("blue", "red"), pch=c(1,NA),lty=c(0,1), cex=0.8)
load("data.RData")
mu_hat =  sum(data)/length(data)
# formula for sigma
sigma_hat = sqrt(sum((data - mu_hat)^2)/length(data))
cat(paste("Mean of data", mu_hat))
cat(paste("Standard Deviation of data", sigma_hat))
minusloglikenormal = function(theta, x=data) {
  mu<-theta[1]                    # mean of normal distribution
  sigma2<-theta[2]                # variance of normal distribution
  n = length(x)
  a1 = (n/2)*log(2*pi)-(n/2)*log(sigma2)
  a2 = (sum((x - mu)^2)) / (2 * sigma2^2)
```

```
    ans = a1+a2
    return(ans)
}
minusloglikenormal(c(0,1),data)
gradient = function(theta, x = data) {
  mu<-theta[1]
  sigma2<-theta[2]
  n = length(x)
  dmu = sum(mu - x)/sigma2^2
  dsig = (n/sigma2) - sum((x - mu)^2)/sigma2^3
  gr = c(dmu,dsig)
  return(gr)
}
gradient(c(0,1),data)
# Conjugate Gradient method without gradient method:
opt_cg <- optim(par = c(0, 1), fn = minusloglikenormal, method = "CG")

# Conjugate Gradient method without gradient method:
opt_cg_gr <- optim(par = c(0, 1), fn = minusloglikenormal, gr = gradient,  method = "CG")

# BGFS method without gradient method:
opt_bfgs <- optim(par = c(0, 1), fn = minusloglikenormal, method = "BFGS")

# BGFS method with gradient method:
opt_bfgs_gr <- optim(par = c(0, 1), fn = minusloglikenormal, gr = gradient, method = "BFGS")
cg1 = c(opt_cg$convergence, opt_cg$par, opt_cg$value, opt_cg$counts[1],
        opt_cg$counts[2])
cg2 = c(opt_cg_gr$convergence, opt_cg_gr$par, opt_cg_gr$value,
        opt_cg_gr$counts[1], opt_cg_gr$counts[2])
bfgs1 = c(opt_bfgs$convergence, opt_bfgs$par, opt_bfgs$value,
          opt_bfgs$counts[1], opt_bfgs$counts[2])
bfgs2 = c(opt_bfgs_gr$convergence, opt_bfgs_gr$par, opt_bfgs_gr$value,
          opt_bfgs_gr$counts[1], opt_bfgs_gr$counts[2])
df <- as.data.frame(matrix(c(cg1, cg2, bfgs1, bfgs2), nrow=6))
rownames(df) <- c("Convergence", "Mean", "Standard Deviation", "minusloglikelihood",
                  "function evaluation", "gradient evaluation")
colnames(df) <- c("CG", "BFGS", "CG with gradient", "BFGS with gradient")
knitr::kable(x = df, caption = "Comparison table of the different algorithms")
```