computational_statistics_Lab2_group14

Akshath Srinivas, Samira Goudarzi

2022-11-20

Question 1

subquestion 1 and subquestion 2

Please see the code of the functions implemented as part of the question in appendix section.

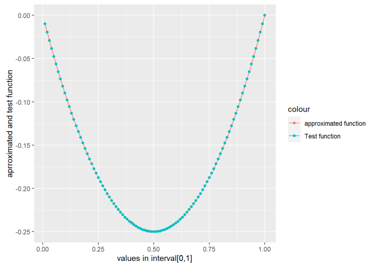
subquestion 3

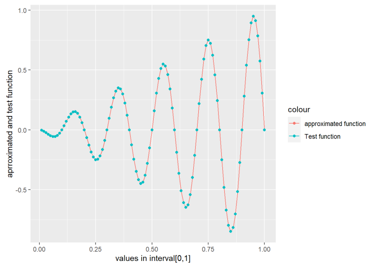
How do we say piece wise parabolic interpolater fare?

My interpolated results using test function should match with values of approximated target function(a0 + a1x + a2x^2) we chose at the beginning. To verify this we have drawn the plot.

Plot has been constructed by taking values of approximated target function(line in the plot) at y axis and values from interval[0,1] on x axis, then for the same values on x axis a line has been added for values of function that needed to be tested(points in plot) for convergence.

The piece-wise parabolic interpolation fares for both the functions. The result of test functions fitted well for values of approximated function(we can see from the plots), as we increase the sub interval the better the results we get.





Question 2

subquestion 2

The joint densitiy of the sample

$$L(\mu,\sigma)=f(y_1,y_2,\dots,y_n|\mu,\sigma)=f(y_1|\mu,\sigma)\times f(y_2|\mu,\sigma)\times\dots\times f(y_n|\mu,\sigma)$$

The above equation can be further solved to

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

log likelihood is after applying log to the above equation is shown below

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

Taking derivative with respect to μ and σ (gradient)

$$\frac{\partial ln[L(\mu,\sigma)]}{\partial\mu}=\frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i-\mu)$$

$$\frac{\partial ln[L(\mu,\sigma)]}{\partial\sigma}=\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{\sigma^3}-\frac{n}{\sigma}$$

The partial derivative on further solving with respect to μ

$$\frac{\partial\ln P(\mathbf{y}|\mu,\sigma)}{\partial\mu}=\frac{1}{\sigma^2}(\sum_{i=1}^{n}y_i-n\mu)$$

Similarly, the partial derivative on further solving with respect to σ

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

By setting partial derivatives to zero we get μ and σ

$$\hat{\mu}=\frac{1}{n}\sum_{i=1}^{n}(y_i)=\bar{y}$$

$$\hat{\sigma}=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i-\bar{y})^2}$$

We have calculated parameters(μ and σ) values for n=100

```
#log likelihood for 100 observations
#n is 100
# -((-(n/2)*log(sigma^2))-((n/2)*log(2*pi))-(sum((y-mu)^2)/(2*sigma^2)))

load('C:/Users/aksha/Downloads/732A90_ComputationalStatisticsHT2022_Lab02/data.RData')

y<-data
n<-100
mu<- sum(y)/n
sigma<-sqrt(sum((y-mu)^2)/n)

mu
```

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

```
sigma
```

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

subquestion 3

BFGS method with and without gradient

First code chunk below is without gradient and second one is with gradient.

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       37       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

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

Conjugate gradient method with and without gradient

First code chunk below is without gradient and second one is with gradient.

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      297       45
##
## $convergence
## [1] 0
##
## $message
## NULL
```

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

likelihood is the products of probabilities of joint probability distribution, this process is exponential computing which is more computationally demanding so, using only maximizing likelihood is bad idea . Applying log(maximizing log likelihood) gives the convenience of sums of probabilities which is easier to analyze without changing the graph with respect to parameter. Moreover, differentiating log of likelihood will give the same estimated parameter because of monotonic property of log.

subquestion 4

Algorithms converged to zero in all the cases above.

optimal values found for mu is 1.275528 and sigma is 2.005977 using BFGS method(with and without gradient)

optimal values found for mu is 1.275528 and sigma is 2.005976 using CG method(with and without gradient)

For BFGS without gradient, number of functions and gradient required to converge

```
## function gradient
##       37       15
```

For BFGS with gradient, number of functions and gradient required to converge.

```
## function gradient
##       39       15
```

For CG without gradient, number of functions and gradient required to converge

```
## function gradient
##      297       45
```

For CG with gradient, number of functions and gradient required to converge

```
## function gradient
##       53       17
```

All the above methods give same parameters values, we would recommend BFGS method without gradient because of fewer evaluations(function and gradient).

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

######question 1- optimizing parameters####

error_func<-function(par,...){
  a0<-par[1]
  a1<-par[2]
  a2<-par[3]
  x0<-...[1]
  x1<-...[2]
  x2<-...[3]

  min_error<-((my_func(x0)-(a0+(a1*x0)+(a2*(x0^2))))^2)+((my_func(x1)-(a0+(a1*x1)+(a2*(x1^2))))^2)+
    ((my_func(x2)-(a0+(a1*x2)+(a2*(x2^2))))^2)

  return(min_error)
}


optimization<-function(x){
  op<-optim(par=c(0,0,0),fn=error_func,x=x)
  a0<-op$par[1]
  a1<-op$par[2]
  a2<-op$par[3]

  return(c(a0,a1,a2))
}

#approximation function takes interval vector values and function to approximate as input
approximation<-function(vec,fn2){
  test<-c()
  approx<-c()
  for (i in 1:length(vec)){
    tes<-fn2(vec[i])
    test<-c(test,tes)
    a0<-optimization(c(vec[i]-0.01,vec[i]-0.005,vec[i]))[1]
    a1<-optimization(c(vec[i]-0.01,vec[i]-0.005,vec[i]))[2]
    a2<-optimization(c(vec[i]-0.01,vec[i]-0.005,vec[i]))[3]
    appr<-(a0+(a1*vec[i])+(a2*(vec[i]^2)))
    approx<-c(approx,appr)
  }
  return(list(test,approx))
}


# first test function given
my_func<-function(x_val){
  ac<--x_val*(1-x_val)
  return(ac)
}

# plotting test function on top of approx function to verify.

#intervals vector

vec<-seq(0.01,1,by=0.01)

#aprroximation function written in above subquestion

pl_1<-approximation(vec,my_func)

library(ggplot2)
ggplot()+geom_line(aes(x=vec,y=pl_1[[2]],colour='approximated function'))+geom_point(aes(x=vec,y=pl_1[[1]],colour
='Test function'))+xlab('values in interval[0,1]')+ylab('aprroximated and test function')


#second test function given
my_func<-function(x_val){
  ac<--x_val*(sin(10*pi*x_val))
  return(ac)
}

#intervals vector
vec<-seq(0.01,1,by=0.01)

#aprroximation function written in above subquestion
pl_2<-approximation(vec,my_func)

library(ggplot2)
ggplot()+geom_line(aes(x=vec,y=pl_2[[2]],colour='approximated function'))+geom_point(aes(x=vec,y=pl_2[[1]],colour
='Test function'))+xlab('values in interval[0,1]')+ylab('aprroximated and test function')

#log likelihood for 100 observations
#n is 100
# -((-(n/2)*log(sigma^2))-((n/2)*log(2*pi))-(sum((y-mu)^2)/(2*sigma^2)))

load('C:/Users/aksha/Downloads/732A90_ComputationalStatisticsHT2022_Lab02/data.RData')

y<-data
n<-100
mu<- sum(y)/n
sigma<-sqrt(sum((y-mu)^2)/n)

mu
sigma

###### question 2- maximum likelihood#####

#negative maximum likelihood
load('C:/Users/aksha/Downloads/732A90_ComputationalStatisticsHT2022_Lab02/data.RData')
neg_max_likelihood<-function(x){
  y<-data
  mu<-x[1]
  sigma<-x[2]
  n<-100
  lik<-n / 2  * log(2 * pi * sigma^2) + (1 / (2 * sigma^2)) * sum((y - mu)^2)
  return(lik)
}

gradient<-function(x){
  y<-data
  mu<-x[1]
  sigma<-x[2]
  n<-100
  return(c(-((sum(y-mu))/(sigma^2)), -(((sum((y-mu)^2))/(sigma^3))-(n/sigma))))
}

#BFGS method with and without gradient
optim(par=c(0,1),fn=neg_max_likelihood,method = "BFGS")
optim(par=c(0,1),fn=neg_max_likelihood,method = "BFGS",gr=gradient)

#Conjugate gradient method with and without gradient
optim(par=c(0,1),fn=neg_max_likelihood,method = "CG")
optim(par=c(0,1),fn=neg_max_likelihood,method = "CG",gr=gradient)

# number of gradients and functions required to converge
optim(par=c(0,1),fn=neg_max_likelihood,method = "BFGS")$counts
optim(par=c(0,1),fn=neg_max_likelihood,method = "BFGS",gr=gradient)$counts
optim(par=c(0,1),fn=neg_max_likelihood,method = "CG")$counts
optim(par=c(0,1),fn=neg_max_likelihood,method = "CG",gr=gradient)$counts
```