computational statistic lab2

Group 12

11/13/2021

This font is code

This font is comment

Red color is print message

Example :

this is font for comment comment

print(cg\_gr[['counts']]) code

## function gradient print message  
## 53 17

## Question 1

### question 1:

The ‘use\_optim()’ function takes there parameters: ‘a’ : initial value of a ‘x’ :the given point ‘v’ :value of original function at the given x.

library(ggplot2)  
library(reshape2)

square\_error <- function(v,x,a)  
{   
 res <- c()   
 for(i in x)  
 {  
 px <- c(0,i,i\*\*2)  
 res <- c(res,t(px)%\*%a)  
 }  
 return(sum((v-res)\*\*2))  
}  
  
use\_optim <-function(par\_x,init\_a,v)  
{  
 result <- optim(init\_a, fn = square\_error,x=par\_x,v=v)  
 return(result)  
}

### question 2:

The ‘sub\_interval()’ function takes two parameters:

‘original\_function’ : the function which will be approximated.

‘number\_of\_interval’: number of intervals for each interval.

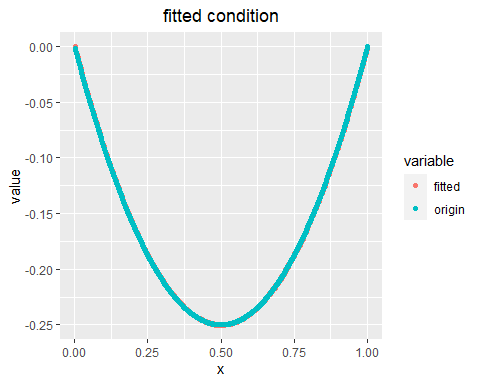
sub\_interval <- function(original\_function,numbers\_of\_interval)  
{  
 interval\_point\_list <- (0:numbers\_of\_interval)/numbers\_of\_interval  
 value\_interval <- c()  
 for (i in 1:(length(interval\_point\_list)-1))  
 {   
 start <- interval\_point\_list[i]  
 end <- interval\_point\_list[i+1]  
 mid <- (start+end)/2  
 x <- c(start,mid,end)  
 fn1 <- function(x) original\_function(x)  
 v <- fn1(x)  
 res <- use\_optim(x,c(1,1,1),v)  
 a <- res[['par']]  
 fitted <- c()  
 for(i in x)  
 {  
 px <- c(0,i,i\*\*2)  
 fitted <- c(fitted,t(px)%\*%a)  
 }  
 value\_interval <- c(value\_interval,fitted)  
 }  
 aex <- 1:(3\*(length(interval\_point\_list)-1))/(3\*(length(interval\_point\_list)-1))  
   
 # reshape data so that 2 plot lines can be plotted in a single graph  
 df <- data.frame(x = aex,fitted=value\_interval,origin=fn1(aex))  
 # plot  
 df1 <- melt(df,id.vars='x')  
 p1 <-ggplot(df1,aes(x=x,y=value))+  
 geom\_point(aes(color=variable))+  
 ggtitle('fitted condition')+  
 theme(plot.title = ggplot2::element\_text(hjust=0.5))  
 print(p1)  
}

### question 3:

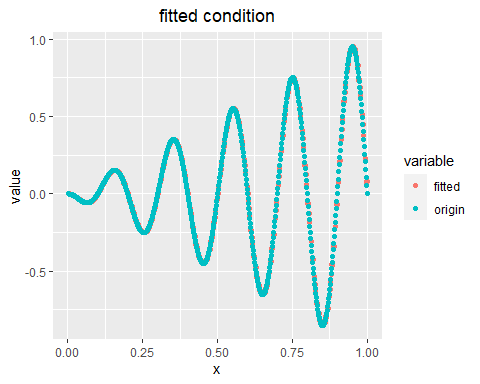
The piece wise parabolic fitted very well. The more intervals we have, the better fitted result we will have.

f1 <- function(x)  
{  
 return(-x\*(1-x))  
}  
f2 <- function(x)  
{  
 return(-x\*sin(10\*pi\*x))  
}

# use the function to get the plot.  
sub\_interval(f1,200)



sub\_interval(f2,200)



## Question 2

### question 1 and 2:

First, the normal density function is :

for X in , we multiply them together:

L() =

To get log-likelihood, we apply log into function L:

In(L()) = nIn()+nIn()+

Now we do derivation to :

……………..(1)

Let (1) equals to 0, we have:

Now for :

……………………………………(2)

Let (2) equals to 0 :

We will use in the following questions.

load('data.RData')  
log\_likelihood <- function(data)  
{  
 n <- length(data)  
 miu <- sum(data)/n  
 sigma <- sqrt(1/n\*sum((data-miu)\*\*2))  
 return(c(miu,sigma))  
}

### question 3:

It’s a bad idea to maximize likelihood function. In the likelihood function, for each , is less than 1, so multiply them may lead to underflow. And the code below is about solution for question 3.

minus\_log\_likelihood <- function(data,par)  
{   
 n <- length(data)  
 first\_part <- -n\*log(1/(par[2]\*sqrt(2\*pi)))  
 second\_part <- 1/2\*sum(((data-par[1])/par[2])\*\*2)  
 return(first\_part+second\_part)  
}  
  
gr\_function <- function(par,data){  
 mu <- sum(par[1]-data)/par[2]\*\*2  
 sd <- length(data)/par[2]-sum((data-par[1])\*\*2)/(par[2]\*\*3)  
 return(c(mu,sd))  
}  
  
conjugate\_gradient <- function(par,minus\_log\_likelihood,data)  
{  
 res <- optim(par,fn=minus\_log\_likelihood,data=data,method = 'CG')  
 return(res)  
}  
  
conjugate\_gradient\_withgr <- function(par,minus\_log\_likelihood,data,gr)  
{  
 res <- optim(par,fn=minus\_log\_likelihood,data=data,method = 'CG',gr=gr)  
 return(res)  
}  
  
  
BFGS <- function(par,minus\_log\_likelihood,data)  
{  
 res <- optim(par,fn=minus\_log\_likelihood,data=data,method = 'BFGS')  
 return(res)  
}  
  
BFGS\_withgr <- function(par,minus\_log\_likelihood,data,gr)  
{  
 res <- optim(par,fn=minus\_log\_likelihood,data=data,method = 'BFGS',gr=gr)  
 return(res)  
}  
  
cg <- conjugate\_gradient(c(0,1),minus\_log\_likelihood,data)

bf <- BFGS(c(0,1),minus\_log\_likelihood,data)

cg\_gr <-conjugate\_gradient\_withgr(c(0,1),minus\_log\_likelihood,data,gr\_function)

bf\_gr <- BFGS\_withgr(c(0,1),minus\_log\_likelihood,data,gr\_function)

print(bf[['par']])

## [1] 1.275528 2.005977

### question 4:

All the algorithms converge in all cases. These 4 algorithms return the same value of optimal parameters. so optimal u = 1.275528, sigma = 2.005977.

print(bf[['par']])

## [1] 1.275528 2.005977

For CG method:

if there is a given gradient,function evaluation : 53 gradient evaluation : 17

print(cg\_gr[['counts']])

## function gradient   
## 53 17

if there is no given gradient, function evaluation: 111 , gradient evaluation : 23

print(cg[['counts']])

## function gradient   
## 111 23

For BFGS method:

if there is a given gradient,function evaluation : 39 gradient evaluation : 15

print(bf\_gr[['counts']])

## function gradient   
## 39 15

if there is no given gradient, function evaluation: 37 , gradient evaluation : 15

print(bf[['counts']])

## function gradient   
## 37 15

Convergence :

In the model output, the parameter `convergence` is used to indicate whether algorithm converge successfully.`0` means success. From the results below, we can see all the algorithms works well.

print(bf[['convergence']])

## [1] 0

print(cg\_gr[['convergence']])

## [1] 0

print(cg[['convergence']])

## [1] 0

print(bf\_gr[['convergence']])

## [1] 0

Since all the algorithms can get same results (they all converge and return same results), I think BFGS method without a given gradient is good, just because of fewer evaluations.