Computational Statistics Lab2 Group 29

Jin Yan (jinya425), Yaning Wang (yanwa579)

2022-11-17

Question 1: Optimizing parameters

1) Use optim() function to find values of (a0; a1; a2).optim() should minimize the squared error.

```
#a is a vector that contains three elements a0,a1,a2.
#They are the parameters of the piecewise quadratic function.
#The purpose of the function below is to find the value of a.
#So that the value of the squared error is the minimum.
piecewise_function<-function(x,a){
  n<-length(x)
  reslut<-c()
  for(i in 1:n){
    x_new<-c(0,x[i],x[i]^2)
    reslut<-append(reslut,t(x_new)%*%a)
  }
  return(reslut)
}

# The original function is f.When the input is a vector x,the response of the
#function f is f(x).We assume the response is response_f
the_squared_error<-function(response_f,x,a){
  squared_error<-sum((response_f-piecewise_function(x,a))^2)
  return(squared_error)
}

optim_squared_error<-function(x,response_f,initial_a){
  op_value<-optim(par=initial_a,fn=the_squared_error,x=x,response_f=response_f)
  return(op_value)
}
```

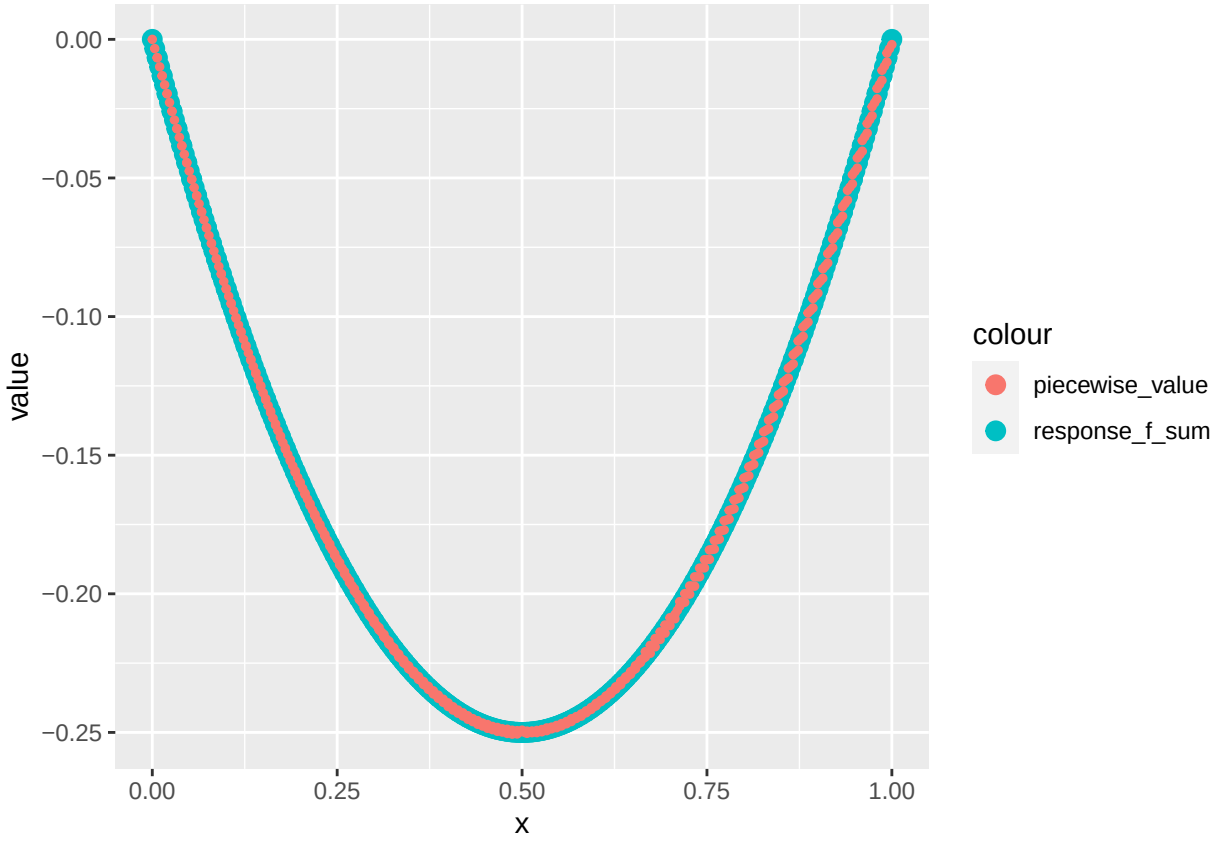
2) Construct a function that approximates a function defined on the interval [0; 1].

```
subinterval <- function(f,number_interval)
{
 #x_interval is a vector contains the start and end points of all the intervals
  x_interval<-seq(from=0,to=1,length.out=number_interval+1)
  sum_a<-c()
  piecewise_value<-c()
  response_f_sum<-c()
  sum_x<-c()
  for(i in 1:number_interval){
```

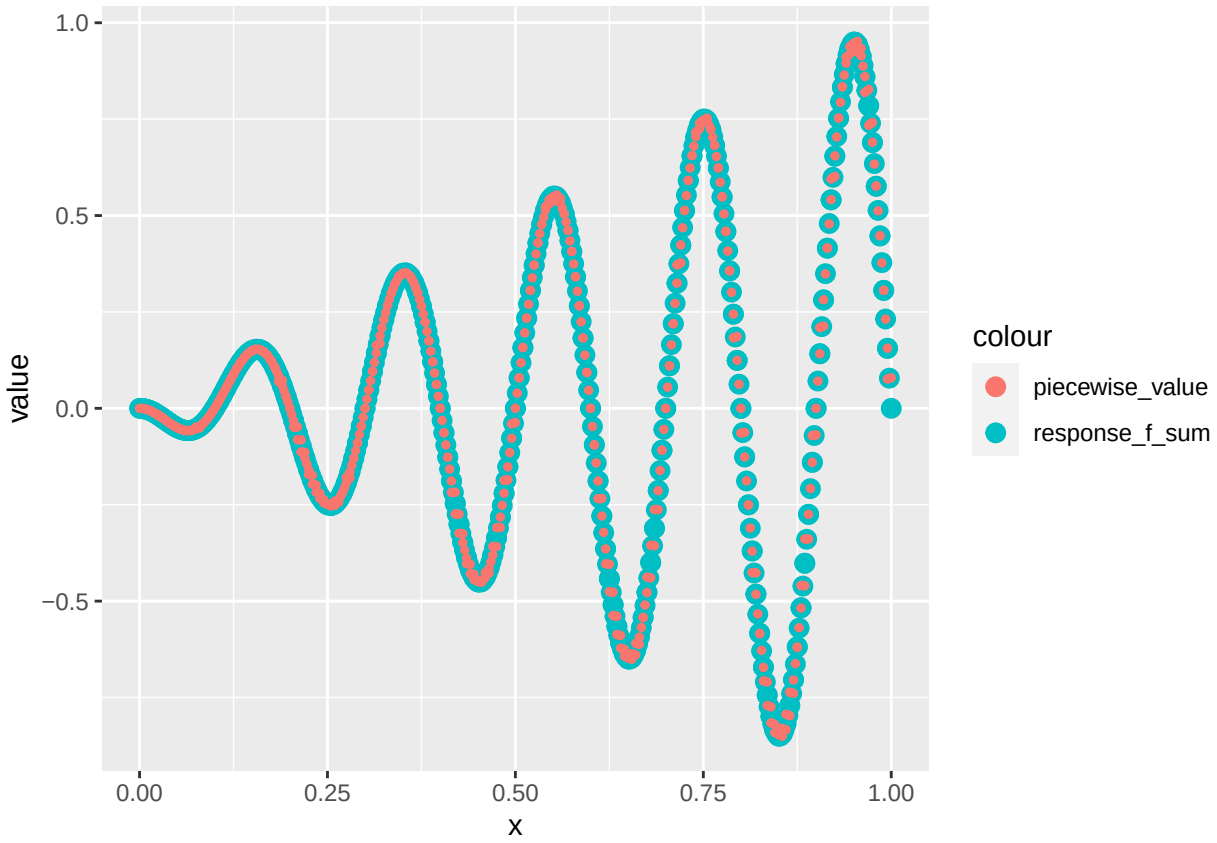
```
    #x is a vector contains the start,middle and end points of the ith interval
    x<-c(x_interval[i],mean(c(x_interval[i],x_interval[i+1])),x_interval[i+1])
    response_f<-f(x)
    op_value<-optim_squared_error(x,response_f,c(1,1,1))
    a <- op_value[['par']]
    #sum_a contains a0,a1,a2 in all subintervals
    sum_a<-rbind(sum_a,a)
    piecewise_value<-c(piecewise_value,piecewise_function(x,a))
    sum_x<-c(sum_x,x)
    response_f_sum<-c(response_f_sum,response_f)
  }
  plot_data1<-data.frame(x=sum_x, y=response_f_sum)
  plot_data2<-data.frame(x=sum_x, y=piecewise_value)
  p = ggplot2::ggplot() +
    ggplot2::geom_point(data = plot_data1, ggplot2::aes(x = x, y = y,
                                        colour = "response_f_sum"),size = 3) +
    ggplot2::geom_point(data = plot_data2, ggplot2::aes(x = x, y = y,
                                        colour = "piecewise_value"),size = 1) +
    ggplot2::xlab('x') +
    ggplot2::ylab('value')

  print(p)
}
```

3) Apply function from the previous item to f1 and f2 and plot f1,f2 and the piecewise quadratic function

```
f1<-function(x){
  response_f1<- -x*(1-x)
  return(response_f1)
}

f2<-function(x){
  response_f2<- -x*sin(10*pi*x)
  return(response_f2)
}

subinterval(f1,150)
```




```
subinterval(f2,200)
```



Q: How did your piecewise parabolic interpolater fare? Explain what you observe.

We use the piecewise quadratic function to approximate the function f1 and f2. The results fitted well, very close to the Original function. The more subinterval we use, the better the results will be.

Question 2: Maximizing likelihood

task1 and task2

The prerequisite for using MLM is to get a expression that shows the probability of getting these observation.

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

In order to do MLM and make the calculation easier, we need to get the logarithm of the expression above.

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

The partial derivative with respect to μ

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

When we set the partial derivative to zero, we know that

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

Similarly, the partial derivative with respect to σ

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

After setting the derivative to zero, we know that

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

```
load("data.RData")
log_likelihood <- function(data){

  n <- length(data)
  miu <- sum(data) / n
  sigma <- sqrt((1 / n) * sum((data-miu)^2))
  print(paste("miu =", miu, "signma =", sigma, sep = " "))
}

log_likelihood(data)
```

```
## [1] "miu = 1.27552760103638 signma = 2.00597647210603"
```

task3

In order to get expression of minus loglikelihood, we should add a minus sign in front to the expression below

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

After doing it, we successfully transform the original question in to another one. Namely we should find the appropriate values of σ and μ to get the minimum of the new expression and it is

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

```
minus_log_likelihood <- function(par){
  n <- length(data)
  miu <- par[1]
  sigma <- par[2]
  value <- n / 2  * log(2 * pi * sigma^2) + (1 / (2 * sigma^2)) * sum((data - miu)^2)
  return(value)
}
```

The gradient vector is calculated like this

```
de_gradient <- function(par){
  n <- length(data)
  miu <- par[1]
  sigma <- par[2]
  partial_derivative_miu <- (1 / sigma^2) * (sum(data) - n * miu)
  partial_derivative_sigma <- -n * (1 / sigma) + sigma^-3 * sum((data - miu)^2)
  return(-c(partial_derivative_miu, partial_derivative_sigma))
}
```

We can use two different methods with and without the argument "gradient" to get the optimums of miu and sigma

```
initial_value <- c(0,1)
optimum_cgm <- optim(initial_value,
                     fn = minus_log_likelihood,
                     gr = NULL,
                     method = "CG")
optimum_cgm_gr <- optim(initial_value,
                        fn = minus_log_likelihood,
                        gr = de_gradient,
                        method = "CG")
optimum_bfgs <- optim(initial_value,
                      fn = minus_log_likelihood,
                      gr = NULL,
                      method = "BFGS")
optimum_bfgs_gr <- optim(initial_value,
                         fn = minus_log_likelihood,
                         gr = de_gradient,
                         method = "BFGS")
```

Why it is a bad idea to maximize likelihood rather than maximizing log–likelihood?

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

From the expression, we can see that the part$(\frac{1}{\sigma\sqrt{2\pi}})$ is less than 1, which means if it multiplies itself many times, the value would be pretty small.In this way, underflow will be more likely to occur.

task4

Did the algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

We can use a table to show the answers.

```
table <- data.frame(
  methods = c("optimum_cgm",
              "optimum_cgm_gr",
              "optimum_bfgs",
              "optimum_bfgs_gr"),
  miu = c(optimum_cgm$par[1],
          optimum_cgm_gr$par[1],
          optimum_bfgs$par[1],
          optimum_bfgs_gr$par[1]),
  sigma = c(optimum_cgm$par[2],
            optimum_cgm_gr$par[2],
            optimum_bfgs$par[2],
            optimum_bfgs_gr$par[2]),
  converge = c(optimum_cgm$convergence,
               optimum_cgm_gr$convergence,
               optimum_bfgs$convergence,
               optimum_bfgs_gr$convergence),
  value = c(optimum_cgm$value,
            optimum_cgm_gr$value,
            optimum_bfgs$value,
```

```
            optimum_bfgs_gr$value),
  fun = c(optimum_cgm$counts[1],
          optimum_cgm_gr$counts[1],
          optimum_bfgs$counts[1],
          optimum_bfgs_gr$counts[1]),
  gradient = c(optimum_cgm$counts[2],
               optimum_cgm_gr$counts[2],
               optimum_bfgs$counts[2],
               optimum_bfgs_gr$counts[2])
)

table
```

```
##           methods      miu    sigma converge    value fun gradient
## 1     optimum_cgm 1.275528 2.005977        0 211.5069 297       45
## 2  optimum_cgm_gr 1.275528 2.005976        0 211.5069  53       17
## 3    optimum_bfgs 1.275528 2.005977        0 211.5069  37       15
## 4 optimum_bfgs_gr 1.275528 2.005977        0 211.5069  41       15
```

From the table, we can clearly see that all of the algorithms converge since all of the converge value for each algorithm is zero. All of the optimal values for "miu" and "sigma" are all the same and equals to 1.275528 and 2.005977 respectively. The information about function and gradient valuations are shown above. Based on the data in the table, we can draw a conclusion that when we use the method "BFGS" with default gradient, the result will be the best.

Appendix

Code in Question 1

```
library(ggplot2)
#1)

#a is a vector that contains three elements a0,a1,a2.
#They are the parameters of the piecewise function.
#The purpose of the function below is to find the value of a.
#So that the value of the squared error is the minimum.
piecewise_function<-function(x,a){
  n<-length(x)
  reslut<-c()
  for(i in 1:n){
    x_new<-c(0,x[i],x[i]^2)
    reslut<-append(reslut,t(x_new)%*%a)
  }
  return(reslut)
}

# The original function is f.When the input is a vector x,the response of the
#function f is f(x).We assume the response is response_f
the_squared_error<-function(response_f,x,a){
  squared_error<-sum((response_f-piecewise_function(x,a))^2)
  return(squared_error)
```

```
}

optim_squared_error<-function(x,response_f,initial_a){
  op_value<-optim(par=initial_a,fn=the_squared_error,x=x,response_f=response_f)
  return(op_value)
}

#2)

subinterval <- function(f,number_interval)
{
 #x_interval is a vector contains the start and end points of all the intervals
  x_interval<-seq(from=0,to=1,length.out=number_interval+1)
  sum_a<-c()
  piecewise_value<-c()
  response_f_sum<-c()
  sum_x<-c()
  for(i in 1:number_interval){
    #x is a vector contains the start,middle and end points of the ith interval
    x<-c(x_interval[i],mean(c(x_interval[i],x_interval[i+1])),x_interval[i+1])
    response_f<-f(x)
    op_value<-optim_squared_error(x,response_f,c(1,1,1))
    a <- op_value[['par']]
    #sum_a contains a0,a1,a2 in all subintervals
    sum_a<-rbind(sum_a,a)
    piecewise_value<-c(piecewise_value,piecewise_function(x,a))
    sum_x<-c(sum_x,x)
    response_f_sum<-c(response_f_sum,response_f)
  }
  plot_data1<-data.frame(x=sum_x, y=response_f_sum)
  plot_data2<-data.frame(x=sum_x, y=piecewise_value)
  p = ggplot() +
    geom_point(data = plot_data1, aes(x = x, y = y,
                                      colour = "response_f_sum"),size = 3) +
    geom_point(data = plot_data2, aes(x = x, y = y,
                                      colour = "piecewise_value"),size = 1) +
    xlab('x') +
    ylab('value')

  print(p)
}

#3)

f1<-function(x){
  response_f1<- -x*(1-x)
  return(response_f1)
}

f2<-function(x){
  response_f2<- -x*sin(10*pi*x)
  return(response_f2)
}
```

```
subinterval(f1,150)
subinterval(f2,200)
```

Code in Question 2

```
# log_likelihood function
load("data.RData")
log_likelihood <- function(data){

  n <- length(data)
  miu <- sum(data) / n
  sigma <- sqrt((1 / n) * sum((data-miu)^2))
  print(paste("miu =", miu, "signma =", sigma, sep = " "))
}

# minus_loglikelihood function
minus_log_likelihood <- function(par){
  n <- length(data)
  miu <- par[1]
  sigma <- par[2]
  value <- n / 2  * log(2 * pi * sigma^2) + (1 / (2 * sigma^2)) * sum((data - miu)^2)
  return(value)
}

# de_gradient function

de_gradient <- function(par){
  n <- length(data)
  miu <- par[1]
  sigma <- par[2]
  partial_derivative_miu <- (1 / sigma^2) * (sum(data) - n * miu)
  partial_derivative_sigma <- -n * (1 / sigma) + sigma^-3 * sum((data - miu)^2)
  return(-c(partial_derivative_miu, partial_derivative_sigma))
}


# functions used for optimization

initial_value <- c(0,1)
optimum_cgm <- optim(initial_value,
                     fn = minus_log_likelihood,
                     gr = NULL,
                     method = "CG")
optimum_cgm_gr <- optim(initial_value,
                        fn = minus_log_likelihood,
                        gr = de_gradient,
                        method = "CG")
optimum_bfgs <- optim(initial_value,
                      fn = minus_log_likelihood,
                      gr = NULL,
                      method = "BFGS")
```

```r
optimum_bfgs_gr <- optim(initial_value,
                         fn = minus_log_likelihood,
                         gr = de_gradient,
                         method = "BFGS")


# a dataframe used for showing information

table <- data.frame(
  methods = c("optimum_cgm",
              "optimum_cgm_gr",
              "optimum_bfgs",
              "optimum_bfgs_gr"),
  miu = c(optimum_cgm$par[1],
          optimum_cgm_gr$par[1],
          optimum_bfgs$par[1],
          optimum_bfgs_gr$par[1]),
  sigma = c(optimum_cgm$par[2],
            optimum_cgm_gr$par[2],
            optimum_bfgs$par[2],
            optimum_bfgs_gr$par[2]),
  converge = c(optimum_cgm$convergence,
               optimum_cgm_gr$convergence,
               optimum_bfgs$convergence,
               optimum_bfgs_gr$convergence),
  value = c(optimum_cgm$value,
            optimum_cgm_gr$value,
            optimum_bfgs$value,
            optimum_bfgs_gr$value),
  fun = c(optimum_cgm$counts[1],
          optimum_cgm_gr$counts[1],
          optimum_bfgs$counts[1],
          optimum_bfgs_gr$counts[1]),
  gradient = c(optimum_cgm$counts[2],
               optimum_cgm_gr$counts[2],
               optimum_bfgs$counts[2],
               optimum_bfgs_gr$counts[2])
)
```