computational statistic lab2

Group 12

11/13/2021

This font is code

This font is comment

Red color is print message

Example :

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

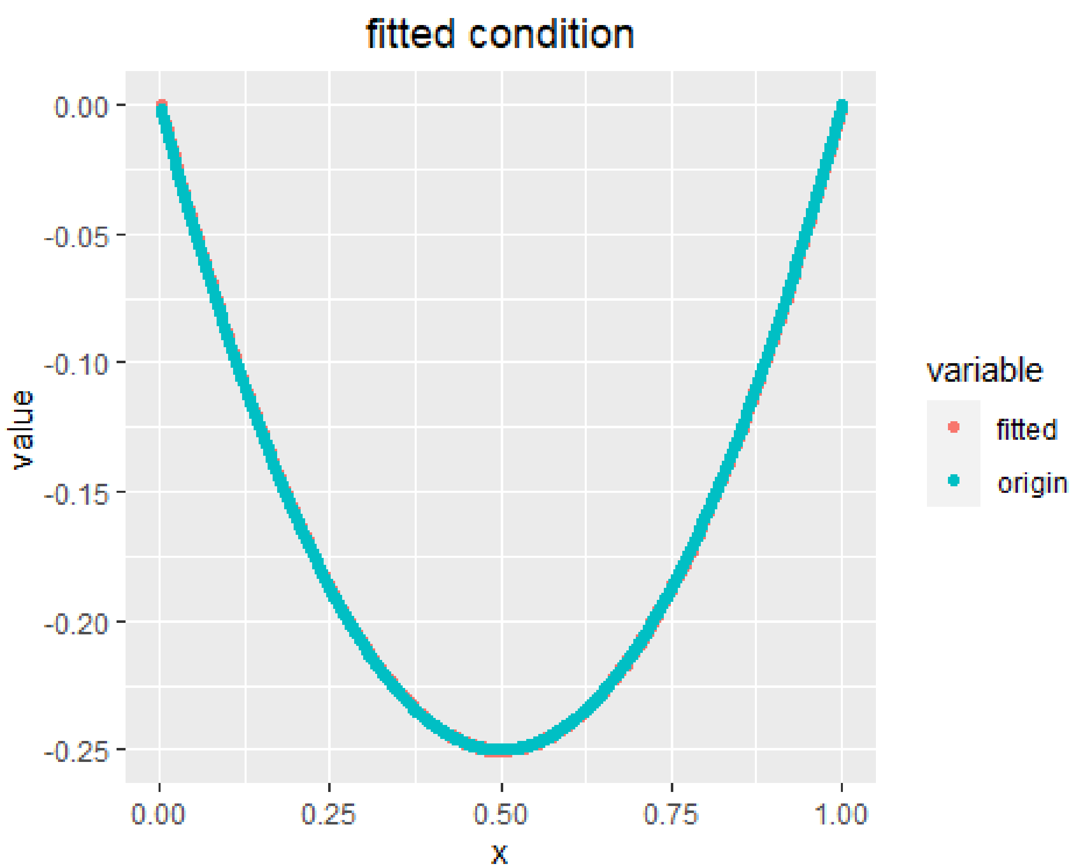
‘number_of_interval’: number of intervals for each interval.

```
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)
}
```

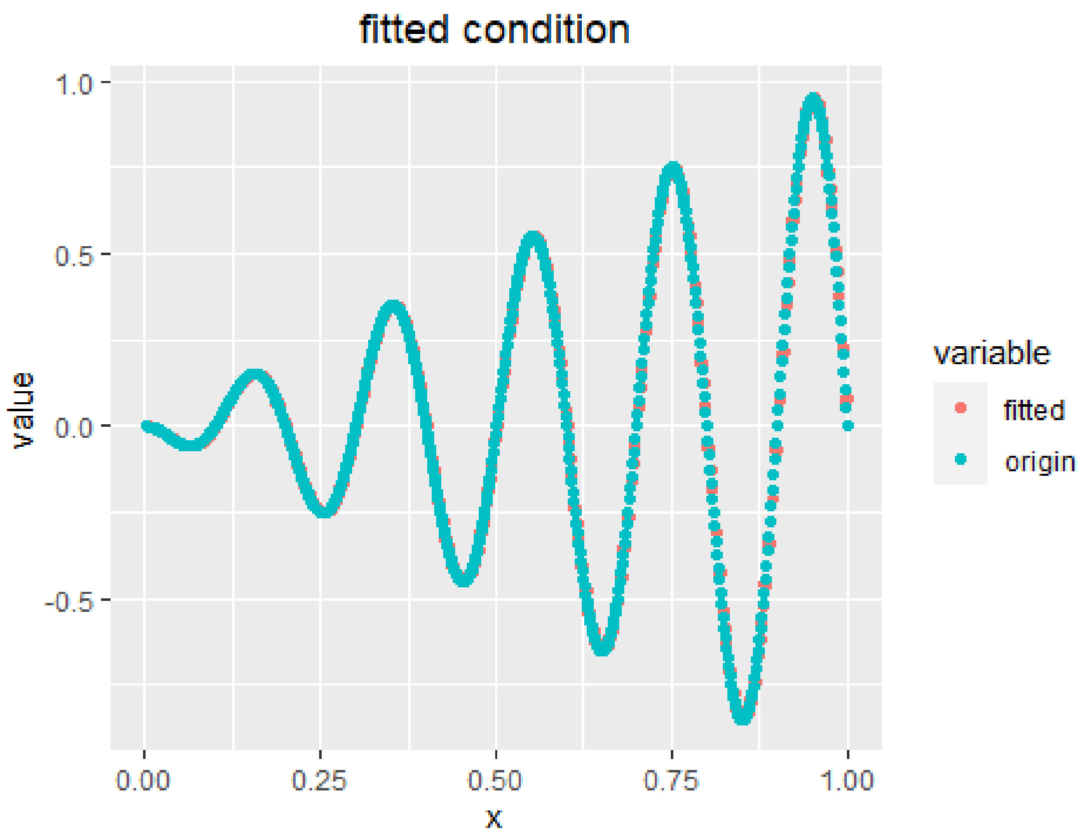
question 3:

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

The dirivative result is:

question 1 and 2:

By using the result, we have log_likelihood function as below.

```
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, since this process is related to exponential computing, which is very time consuming and will use much memory.

```
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)
{
```

```r
 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.

```r
print(bf[['par']])
```

```
## [1] 1.275528 2.005977
```

For CG method:

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

```r
print(cg_gr[['counts']])
```

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

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

```r
print(cg[['counts']])
```

```
## function gradient
##      111       23
```

For BFGS method:

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

```r
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
```

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