Computational Statistics

Lab 2 report

Jonathan Dorairaj, Yi Hung Chen

2022-11-19

Question 1: Optimizing parameters

1.1

Function `my_optim()` is used with inputs `param` and `X` which is then used by `optim()` to optimize parameters.We return these parameters.
`piecewise()` is a function that calculates evaluates the function for given values of `x` and parameters (a_0,a_1,a_2).
Function `MSE()` calculates the Mean square error between functions `fx()` and `piecewise()`.

```r
myoptim <- function(param,X){
  X1 <- X
  optimizer <- optim(par = param, fn = MSE,X = X1)
  A <- optimizer$par
  return(A)

}

# piecewise function
piecewise <- function(A,x){

  A <-as.matrix(A)
  X <- matrix(c(1,x,x^2), nrow = 1)
  fx <- X %*% A
  return(fx)
}


#function to calculate Mean Square Error

MSE <- function(param,X){
  x0 <- X[1]
  x1 <- X[2]
  x2 <- X[3]

  mse_calc <- sum( (fx(x0) - piecewise(param,x0))^2,
                   (fx(x1) - piecewise(param,x1))^2,
                   (fx(x2) - piecewise(param,x2))^2 )
  return(mse_calc)
}
```

1.2

Function `fapprox()` takes arguments `obj` and `n` where `obj` is the function to be approximated and `n` is the number of intervals. It returns a data.frame of x values and the corresponding values of the parameters.

```r
# obj <-  function to approximate
# n <- number of intervals

#function returns data.frame with x value & approximated parameters at that value
fapprox <- function(obj,n){
  coeff_list <- list()
  x_df1 <- data.frame(x= numeric(),
                      par1 = numeric(),
                      par2 = numeric(),
                      par2 = numeric())

  k <- seq(0,1,1/n)
  i <- 1
  while(i < length(k)){
    x0 <- k[i]
    x2 <- k[i+1]
    x1 <- (x0+x2)/2

    X <- c(x0,x1,x2)

    coeff_list[[i]] <- optim(par = c(0,0,1),fn = MSE,X = X)

    temp1 <- c(x0,coeff_list[[i]]$par[1],coeff_list[[i]]$par[2],coeff_list[[i]]$par[3])
    temp2 <- c(x1,coeff_list[[i]]$par[1],coeff_list[[i]]$par[2],coeff_list[[i]]$par[3])
   temp3 <- c(x2,coeff_list[[i]]$par[1],coeff_list[[i]]$par[2],coeff_list[[i]]$par[3])
    # # store calculated parameters for each x value in variable x_df1
    x_df1 <- rbind(x_df1,temp1,temp2,temp3)
    # x_df1 <- rbind(x_df1,temp2)
    i <-i + 1

  }
   colnames(x_df1) <- c("x","par1","par2","par3")
   return(x_df1)
}
```

1.3

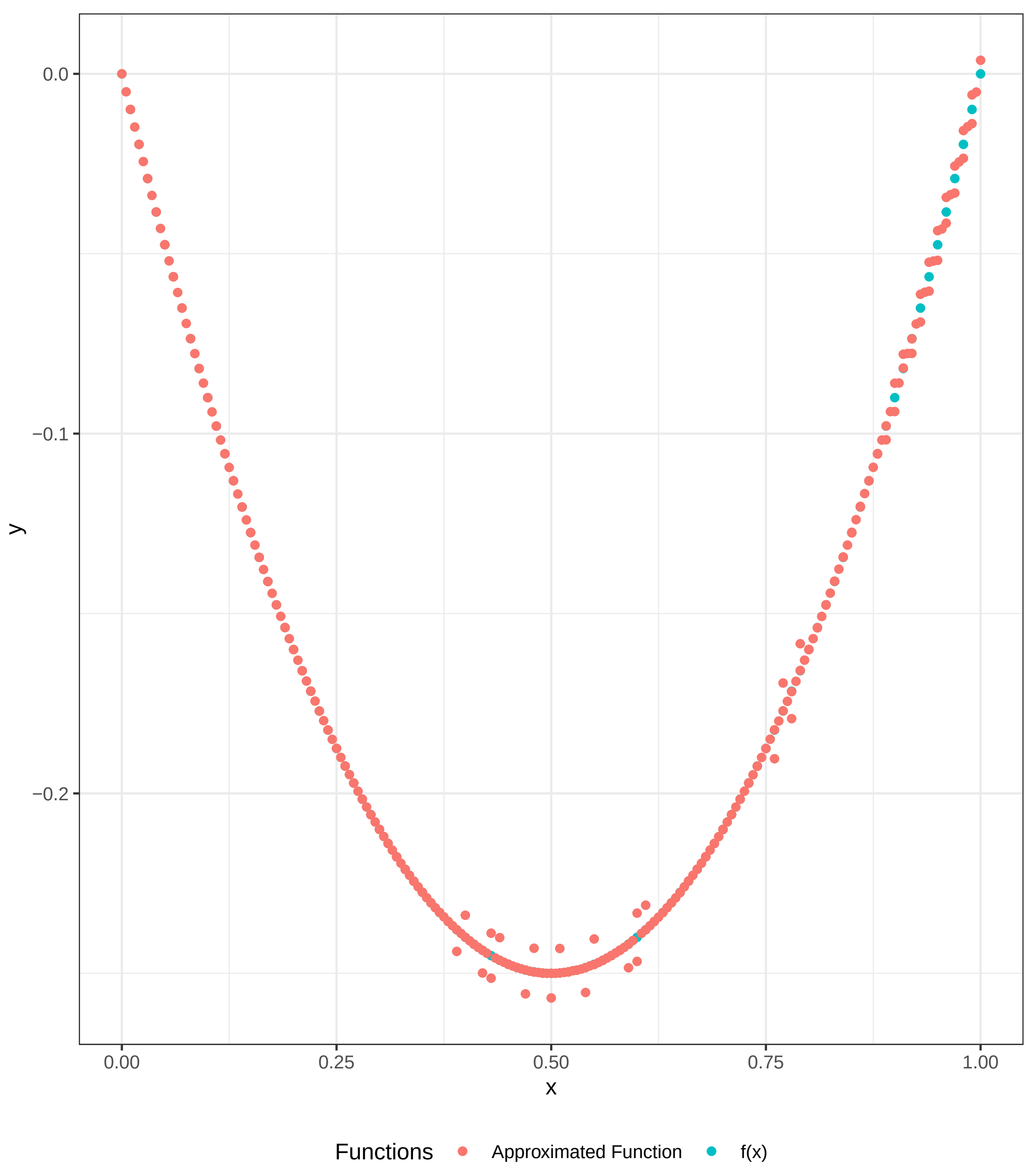
Functions `fx_calc()` and `pred_param()` are used to evaluate respective functions `fx()` and `fappprox()` at each interval and return the values in a data.frame for plotting.
`plot_func()` is used to plot the respective functions.

```r
#function to calculate fx and return df of x,y values for plotting
# n <- number of intervals
fx_calc <- function(n){
  k <- seq(0,1,1/n)
  fx_df = data.frame(x = numeric(),y = numeric())
for (i in 1:length(k)) {
  fx_df[i,] <- c(k[i],fx(k[i]))
}
  return(fx_df)
}

#function to calculate fapprox values and return df of x,y values for plotting
# n <- number of intervals
pred_param <- function(n){
  result_df <-data.frame(X = numeric(), A = numeric())
  x_df1 <- fapprox(fx,n)

   for (i in 1:dim(x_df1)[1]) {
       result_df[i,] <- c(x_df1[i,1],piecewise(t(as.matrix(x_df1[i,2:4])),as.vector(x_df1[i,1])))
  }
   return(result_df)
}

# function to plot fx, fapprox
# n <-  number of intervals
plot_func <- function(n)
  {
    fx_df <- fx_calc(n)
    result_df <- pred_param(n)
    p <- ggplot() + geom_point(data = fx_df,aes(x = x,y= y,colour = "f(x)")) +
    geom_point(data = result_df,aes(x= X,y = A,colour = "approximated function")) +
      theme_bw()   + labs(x = "x",
      y = "y") +
      scale_color_discrete(name = "Functions",
                           labels = c("Approximated Function", "f(x)")) +
      theme(legend.position = 'bottom')
  return(p)
}
```

1.3.1 Approximating $f_1(x) = -(x)(1-x)$

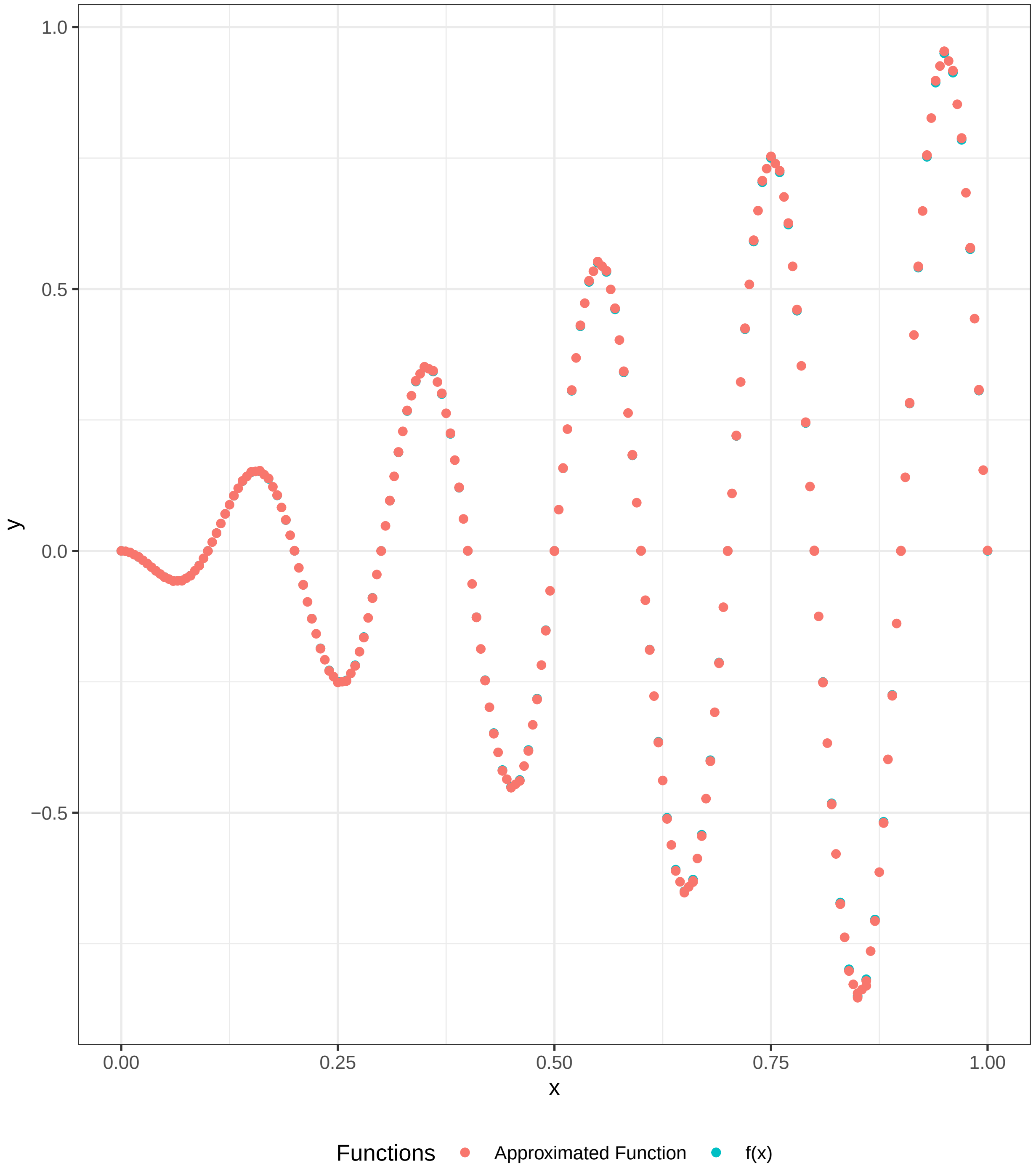
Function `fx()` defined below and plot function `plot_func()` is called for `n=100`.

```
fx <- function(x){
  fx =  -1*x *(1-x)

  return(fx)
}
plot_func(100)
```


0.0
−0.1
−0.2
y
0.00
0.25
0.50
0.75
1.00
x
Functions
Approximated Function
f(x)

1.3.2 Approximating $f_1(x) = -(x)sin(10\pi x)$

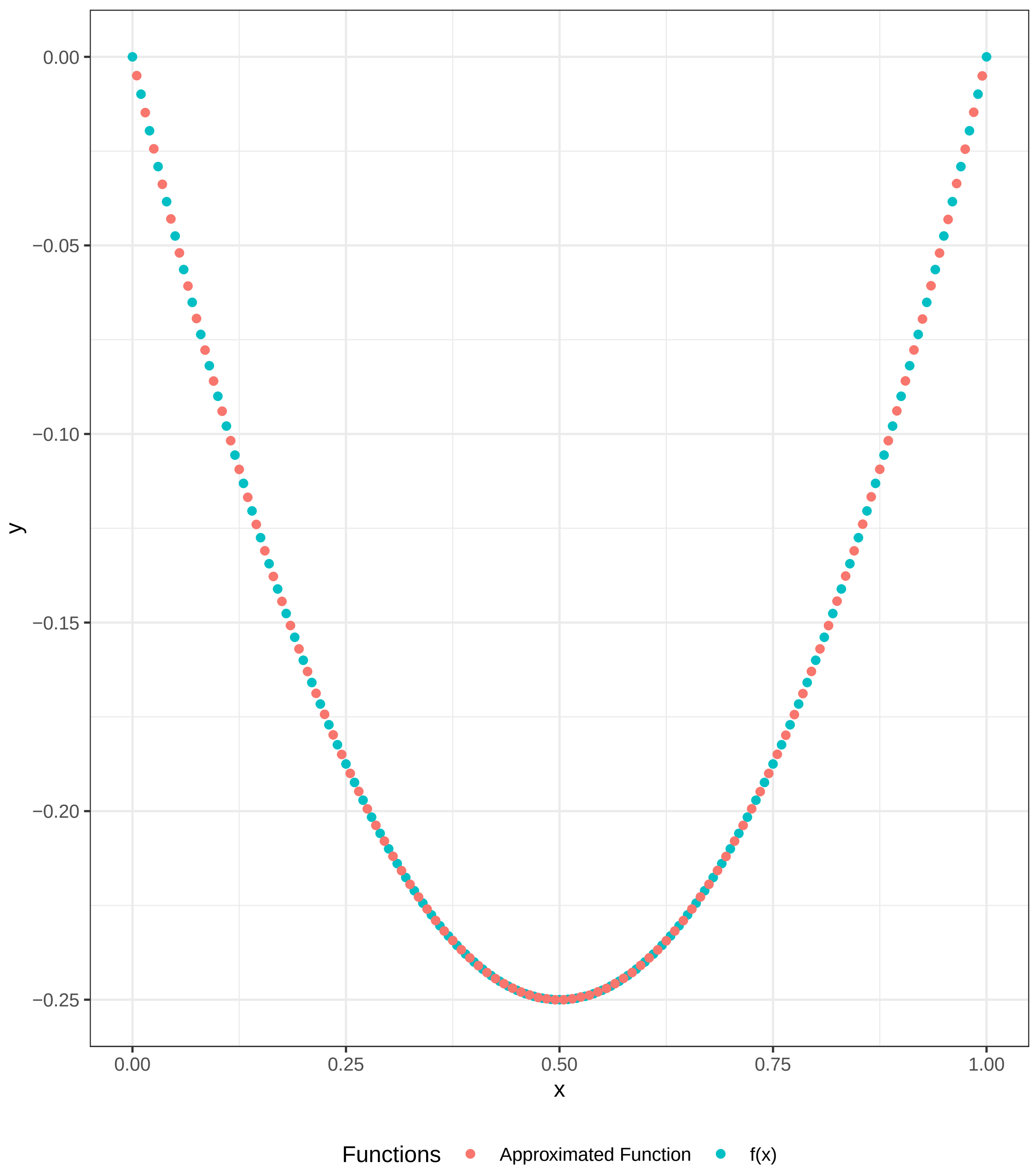
```
fx <- function(x){
  fx <- -1 * x * sin(10*pi*x)
  return(fx)
}
plot_func(100)
```



From the plots approximating the two functions, we see that the piecewise parabolic interpolater generally does a good job of approximating the function `fx()`. However, we do observe points that are not fully aligned with the curve of `fx()`. This is because of the calculation of the parameters (a_0,a_1,a_2) at very small intervals.For example, on the right side of the approximated function, we can see a pattern of 3 dots in a group very close to each other at approximately the same y value. If we plot the function `fapprox()` only for the midpoints of the evaluated interval, we can observe that it fits the function `fx()` better.

```
plot_func2 <- function(n)
  {
    fx_df <- fx_calc(n)
    result_df <- pred_param(n)
    k <- rep(c(FALSE,TRUE,FALSE),100)
    result_df <- result_df[k,]
    p <- ggplot() + geom_point(data = fx_df,aes(x = x,y= y,colour = "f(x)")) +
    geom_point(data = result_df,aes(x= X,y = A,colour = "approximated function")) +
      theme_bw()   + labs(x = "x",
      y = "y") +
      scale_color_discrete(name = "Functions",
                          labels = c("Approximated Function", "f(x)")) +
      theme(legend.position = 'bottom')

  return(p)
}
```

```
fx <- function(x){
  fx =  -1*x *(1-x)
}
plot_func2(100)
```



0.00
−0.05
−0.10
−0.15
−0.20
−0.25
y
0.00
0.25
0.50
0.75
1.00
x
Functions
● Approximated Function
● f(x)

Question 2: Maximizing likelihood

1. Load the data to R environment.

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

2. Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use the derived formula to obtain parameter estimates for the loaded data.

The pdf of Normal Distribution is

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

The likelihood function is

$$L(\mu, \sigma^2) = (2\pi\sigma^2)^{\frac{-n}{2}} exp(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2)$$

The log likelihood function is

$$lnL(\mu, \sigma^2) = \frac{-n}{2} ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

Calculate μ estimater

Set the partial derivative to zero

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

so,

$$\sum_{i=1}^{n} x_i - n\mu = 0$$

we get

$$\hat{\mu} = \frac{1}{n}\sum x_i$$

```r
my_mean <- function(x) {
  mean_hat <- 1/length(x)*sum(x)
  return(mean_hat)
}
my_mean(data)
```

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

Calculate σ estimater

For σ estimater we do similar calculation. (Note: we first calculate the partial derivative of σ^2 then square root it)

$$\frac{\partial lnL(\mu,\sigma^2)}{\partial\sigma^2} = \frac{-n}{2}\frac{2\pi}{2\pi\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i-\mu)^2 = 0$$

so,

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

we get.

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

```
my_sigma<- function(x){
  sigma_hat <- sqrt(sum((x-my_mean(x))^2)/length(x))
  return(sigma_hat)
}
my_sigma(data)
```

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

3. Optimize the minus log–likelihood function with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method and BFGS algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log–likelihood? The minus log likelihood function is

$$-lnL(\mu,\sigma^2) = \frac{n}{2}ln(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

The gradient for μ

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

The gradient for σ

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

	CG_no_grad	BFGS_no_grad	CG_with_grad	BFGS_with_grad
mu	1.275528	1.275527	1.275528	1.275527
sigma	2.005977	2.005977	2.005977	2.005977
value	211.506949	211.506949	211.506949	211.506949
counts.function	182.000000	37.000000	53.000000	38.000000
counts.gradient	29.000000	15.000000	17.000000	15.000000

	CG_no_grad	BFGS_no_grad	CG_with_grad	BFGS_with_grad
convergence	0.000000	0.000000	0.000000	0.000000

Q3

Why it is a bad idea to maximize likelihood rather than maximizing log–likelihood? In a numeric calculation stand point, maximizing likelihood is more complicated than maximize log-likelihood, this is especially doing calculation by hand. In a computer stand point, log-likelihood is computationally faster than likelihood. Additionally,likelihood results in small values and when they are multiplied,it could result in underflow. Applying log-likelihood converts the multiplication operations into a summation process, which reduces the risk of underflow.

Q4

Did the algorithms converge in all cases? What were the optimal values of parameters? All algorithms converge, with optimal values $\mu = 1.2755277$ and $\sigma = 2.005977$
(Note: The last digit difference is due to display rounding).

How many function and gradient evaluations were required for algorithms to converge? CG with no gradient has 297 function evaluations and 45 gradient evaluations
BFGS with no gradient has 37 function evaluations and 15 gradient evaluations
CG with gradient has 53 function evaluations and 17 gradient evaluations
BFGS with gradient has 39 function evaluations and 15 gradient evaluations

Which settings would you recommend? We would recommend BFGS without gradient. Since it takes the least evaluations(function evaluation = 37 and gradient evaluations = 15) to get the optimal value.

Appendix

```r
#=====Question 1=====
myoptim <- function(param,X){
  X1 <- X

  optimizer <- optim(par = param, fn = MSE,X = X1)
  A <- optimizer$par
  return(A)

}

# piecewise function
piecewise <- function(A,x){

  A <-as.matrix(A)
  X <- matrix(c(1,x,x^2), nrow = 1)
  fx <- X %*% A
  return(fx)
}


#function to calculate Mean Square Error

MSE <- function(param,X){
  x0 <- X[1]
  x1 <- X[2]
  x2 <- X[3]

  mse_calc <- sum( (fx(x0) - piecewise(param,x0))^2,
                   (fx(x1) - piecewise(param,x1))^2,
                   (fx(x2) - piecewise(param,x2))^2 )
  return(mse_calc)
}

# obj <-  function to approximate
# n <- number of intervals

#function returns data.frame with x value & approximated parameters at that value
fapprox <- function(obj,n){
  coeff_list <- list()
  x_df1 <- data.frame(x= numeric(),
                      par1 = numeric(),
                      par2 = numeric(),
                      par2 = numeric())

  k <- seq(0,1,1/n)
  i <- 1
  while(i < length(k)){
    x0 <- k[i]
    x2 <- k[i+1]
    x1 <- (x0+x2)/2
    X <- c(x0,x1,x2)

    coeff_list[[i]] <- optim(par = c(0,0,1),fn = MSE,X = X)

    temp1 <- c(x0,coeff_list[[i]]$par[1],coeff_list[[i]]$par[2],coeff_list[[i]]$par[3])
    temp2 <- c(x1,coeff_list[[i]]$par[1],coeff_list[[i]]$par[2],coeff_list[[i]]$par[3])
   temp3 <- c(x2,coeff_list[[i]]$par[1],coeff_list[[i]]$par[2],coeff_list[[i]]$par[3])
    # # store calculated parameters for each x value in variable x_df1
    x_df1 <- rbind(x_df1,temp1,temp2,temp3)
    # x_df1 <- rbind(x_df1,temp2)
    i <-i + 1

  }
   colnames(x_df1) <- c("x","par1","par2","par3")
   return(x_df1)
}


#function to calculate fx and return df of x,y values for plotting
# n <- number of intervals
fx_calc <- function(n){
  k <- seq(0,1,1/n)
  fx_df = data.frame(x = numeric(),y = numeric())
for (i in 1:length(k)) {
  fx_df[i,] <- c(k[i],fx(k[i]))
}
  return(fx_df)
}

#function to calculate fapprox values and return df of x,y values for plotting
# n <- number of intervals
pred_param <- function(n){
  result_df <-data.frame(X = numeric(), A = numeric())
  x_df1 <- fapprox(fx,n)

   for (i in 1:dim(x_df1)[1]) {
       result_df[i,] <- c(x_df1[i,1],piecewise(t(as.matrix(x_df1[i,2:4])),as.vector(x_df1[i,1])))
  }
   return(result_df)
}

# function to plot fx, fapprox
# n <-  number of intervals
plot_func <- function(n)
  {
    fx_df <- fx_calc(n)
    result_df <- pred_param(n)
    p <- ggplot() + geom_point(data = fx_df,aes(x = x,y= y,colour = "f(x)")) +
    geom_point(data = result_df,aes(x= X,y = A,colour = "approximated function")) +
      theme_bw()   + labs(x = "x",
      y = "y") +
      scale_color_discrete(name = "Functions",
                           labels = c("Approximated Function", "f(x)")) +
      theme(legend.position = 'bottom')

  return(p)
}


fx <- function(x){
  fx =  -1*x *(1-x)

  return(fx)
}
plot_func(100)


fx <- function(x){
  fx <- -1 * x * sin(10*pi*x)
  return(fx)
}
plot_func(100)

plot_func2 <- function(n)
  {
    fx_df <- fx_calc(n)
    result_df <- pred_param(n)
    k <- rep(c(FALSE,TRUE,FALSE),100)
    result_df <- result_df[k,]
    p <- ggplot() + geom_point(data = fx_df,aes(x = x,y= y,colour = "f(x)")) +
    geom_point(data = result_df,aes(x= X,y = A,colour = "approximated function")) +
      theme_bw()   + labs(x = "x",
      y = "y") +
      scale_color_discrete(name = "Functions",
                           labels = c("Approximated Function", "f(x)")) +
      theme(legend.position = 'bottom')

  return(p)
}

plot_func2(100)


#=====Question 2=====
load("data.RData")

#====MLE estimaters
my_mean <- function(x) {
  mean_hat <- 1/length(x)*sum(x)
  return(mean_hat)
}
my_mean(data)

my_sigma<- function(x){
  sigma_hat <- sqrt(sum((x-my_mean(x))^2)/length(x))
  return(sigma_hat)
```

```
}
my_sigma(data)


#=====Miuns log likelihood and optim()
minus_logliklihood <-  function(par){
  x<-data
  formula <- length(x)/2*log(2*pi*par[2]^2) + (1/2/par[2]^2)* sum((x-par[1])^2)
  #par[1] is mean
  #par[2] is sigma
  return(formula)

}
my_grad <- function(par){
  x<-data
  grad_mean <-  - (1/par[2]^2)* sum(x-par[1])
  grad_sigma <- (length(x)/par[2]) - (1/par[2]^3) * sum((x-par[1])^2)

  return(c(grad_mean,grad_sigma))
}
CG_no_grad <- optim(c(0,1), fn = minus_logliklihood, gr=NULL, method = c("CG"))
BFGS_no_grad <- optim(c(0,1), fn = minus_logliklihood, gr=NULL, method = c("BFGS"))
CG_with_grad <- optim(c(0,1), fn = minus_logliklihood, gr=my_grad, method = c("CG"))
BFGS_with_grad <- optim(c(0,1), fn = minus_logliklihood, gr=my_grad, method = c("BFGS"))
CG_no_grad<- c(unlist(CG_no_grad))
BFGS_no_grad<- c(unlist(BFGS_no_grad))
CG_with_grad<- c(unlist(CG_with_grad))
BFGS_with_grad<- c(unlist(BFGS_with_grad))
compare <- data.frame(cbind(CG_no_grad,BFGS_no_grad,CG_with_grad,BFGS_with_grad))

row.names(compare) <- c("mu","sigma","value","counts.function","counts.gradient","convergence")
knitr::kable(compare)
```