Computer Lab 1 (Group 7)

Qinyuan Qi(qinqi464) Satya Sai Naga Jaya Koushik Pilla (satpi345)

2023-11-05

Question 1: Maximization of a likelihood function in one variable (Solved by Satya Sai Naga Jaya Koushik Pilla)

We have independent data $x_1, ..., x_n$ from a Cauchy-distribution with unknown location parameter θ and known scale parameter 1. The log likelihood function is

$$-nlog(\pi) - \sum_{i=1}^{n} log(1 + (x_i - \theta)^2)$$

and it's derivative with respect to θ is

$$\sum_{i=1}^{n} \frac{2(x_i - \theta)}{1 + (x_i - \theta)^2}$$

Data of size $n = 5$ is given: $x = (-2.8, 3.4, 1.2, -0.3, -2.6)$

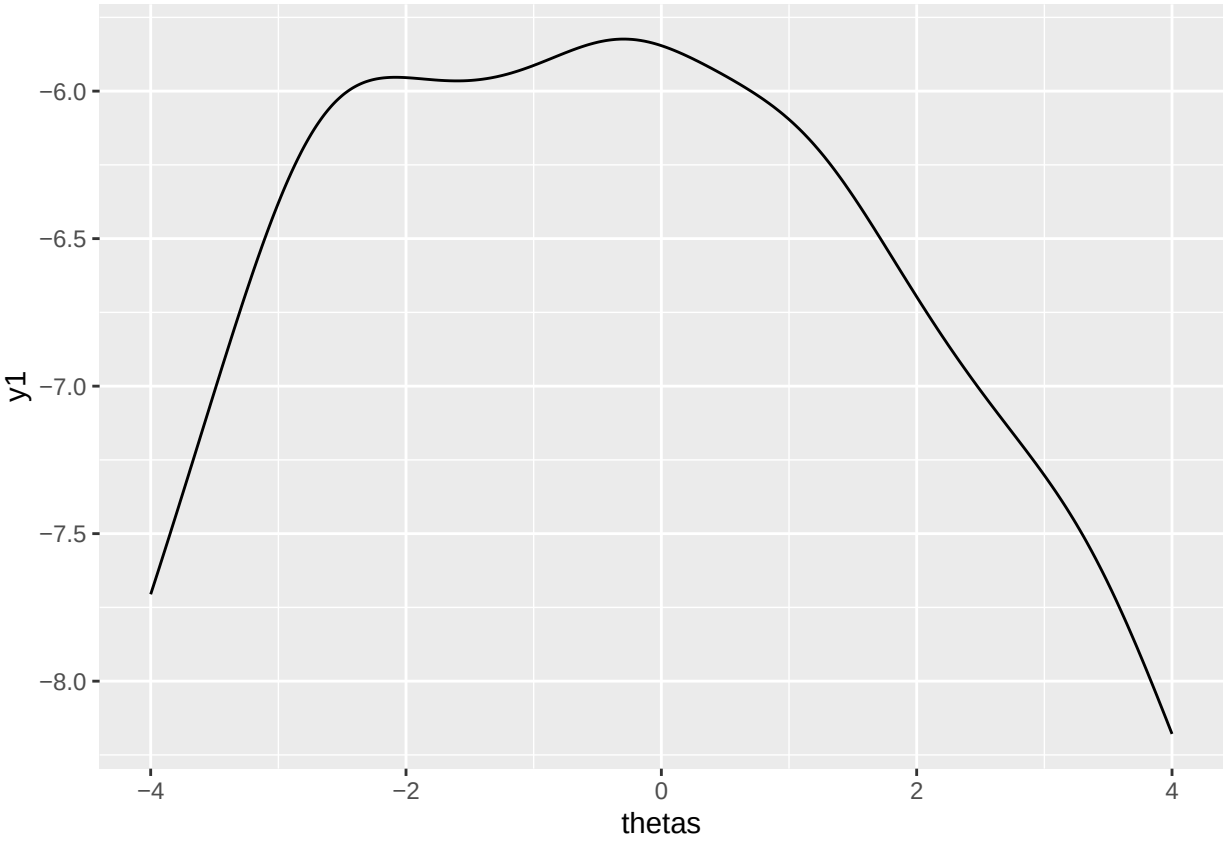
a. Plot the log likelihood function for the given data in the range from -4 to 4. Plot the derivative in the same range and check visually how often the derivative is equal to 0.

Answer:

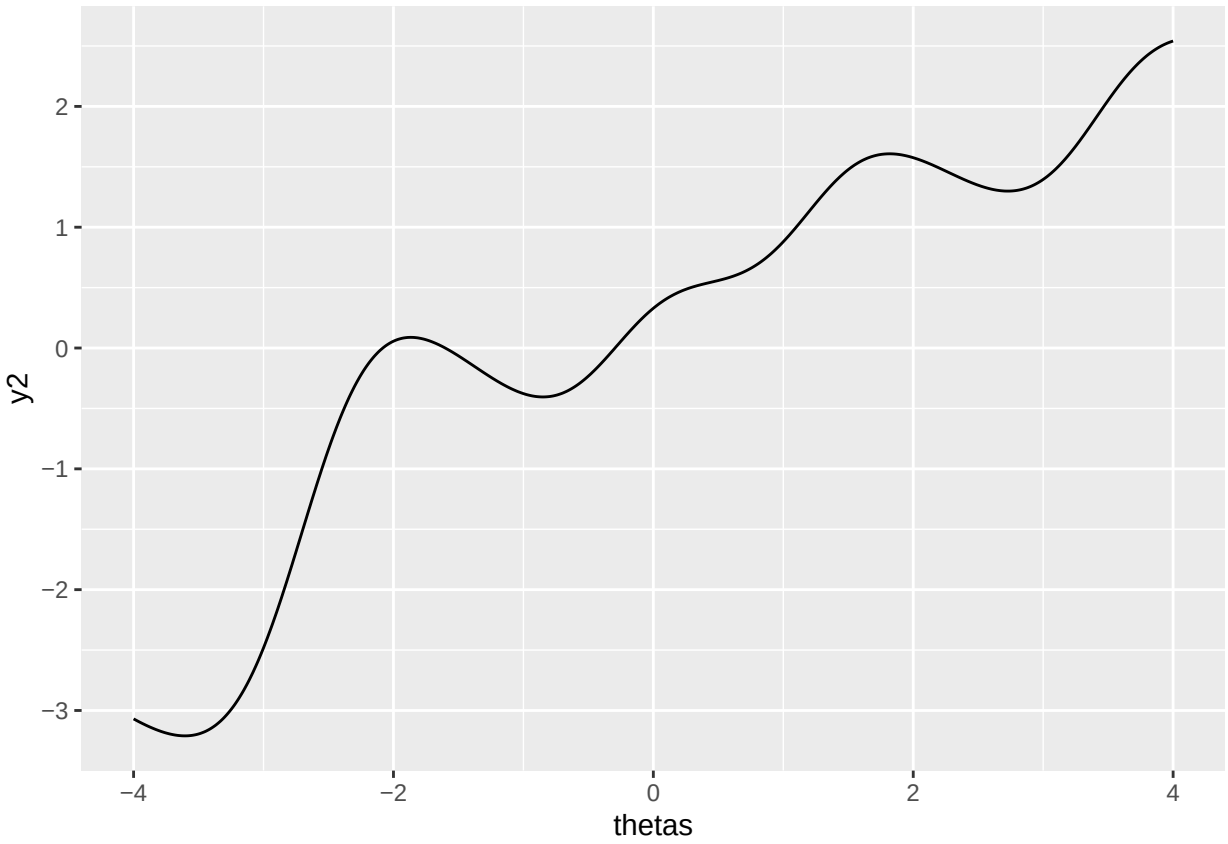
The data range is -4 to 4, to make it smooth curve , we using step 0.01 to generate hundreds of data points.The 2 plots are given below. We can found that there have three points that have derivative is equal to 0.

```
#----------------------------------------
# Code for question 1a
#----------------------------------------
# log_likelihood function
log_likelihood <- function(x, theta){
  n <- length(x)
  ret <- -n * log10(pi) - sum(log10(1+ (x-theta)^2))
  return(ret)
}

# derivative of log_likelihood function
log_likelihood_derivative <- function(x, theta){
  n <- length(x)
  ret <- -sum(2 * (x - theta) / (1 + (x - theta)^2))
  return(ret)
}

# init x ,theta
x <- c(-2.8, 3.4, 1.2, -0.3, -2.6)
thetas <- seq(-4, 4, by = 0.01)
```

```r
# get length of theta
n <- length(thetas)

# init vectors of value of log_likelihood() and log_likelihood_derivative()
y1 <- numeric(n)
y2 <- numeric(n)

# loop to call log_likelihood and log_likelihood_derivative functions
for(i in 1:n){
  y1[i] <- log_likelihood(x,thetas[i])
  y2[i] <- log_likelihood_derivative(x,thetas[i])
}

# plot data
data <- data.frame(thetas,y1,y2)
ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=thetas,y=y1)) +
  ggplot2::geom_line()
```




```r
ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=thetas,y=y2)) +
  ggplot2::geom_line()
```



b. Choose one of the following methods: bisection, secant, or Newton-Raphson. Write your own code to optimize with the chosen method. If you have chosen Newton-Raphson, describe how you derived the second derivative.

Answer:

In the following codes, bisection method was implemented.

For bisection, we set $epsilon = 1e-10, max_k = 1000$.

if $abs(log_likelihood_derivative(mid)) < epsilon$, then loop will stop and we will get our answer mid.

if $k > max_k$,then it will return $null$ which means it does not find the answer.

```
#----------------------------------------
# Code for question 1b
#
# bisection
#----------------------------------------

# init values to stop recursion
epsilon <- 1e-10
max_k <- 1000

# implementation of bisection
bisection_method <- function(xl,xu,k){

  # check k is not greater than max_k
  # if it's too big then stop recursion
  # this is mostly caused if the selected interval is a increasing interval or decreasing interval
```

```
  if (k > max_k) return(NULL)

  # get the mid point
  mid <- (xl + xu) / 2

  # calculate values of log_likelihood_derivative
  f_mid <- log_likelihood_derivative(x,mid)
  f_xl <- log_likelihood_derivative(x,xl)

  # check signs
  if (sign(f_mid) == sign(f_xl)){
    xl <- mid
  }else{
    xu <- mid
  }

  # if the distance between f_mid and 0 is small enough, then we got the answer.
  # else we need to split the interval and call the function again to find new answer.
  if (abs(f_mid) < epsilon){
    return(mid)
  }else(
    bisection_method(xl,xu,k+1)
  )
}

#----------------------------------------
# Test code
#----------------------------------------
#k <- 0
#xl <- -4
#xu <- 4
#ret <- bisection_method(xl=xl,xu=xu,k=k)
```

c. Choose suitable starting values based on your plots to identify all local maxima of the likelihood function. Note that you need pairs of interval boundaries for the bisection or pairs of starting values for secant. Are there any (pairs of) starting values which do not lead to a local maximum? Decide which is the global maximum based on programming results.

Answer:

As mentioned in the 1*b* code, when the selected interval is an increasing interval or decreasing interval that will cause it's derivative will never equal to Zero, in this case we will not get a local maximum.

According to the plot, to get maximization of likelihood function, the range should between [-1,0] so we set $xl = -1, xu = 0$, and we know that $\theta = -0.295245$ (according to result of the following function call).

```
#----------------------------------------
# Code for question 1c
#----------------------------------------
# init values
k <- 0
xl <- -1
xu <- 0

# function call to bisection_method()
ret <- bisection_method(xl=xl,xu=xu,k=k)
```

```r
# check ret to make sure bisection_method will not return null
# which will caused by a increasing interval or decreasing interval
if (is.null(ret)){
  print("Not found")
}else{
  print(ret)
}
```

```
## [1] -0.2952455
```

d. Assume now that you are in a situation where you cannot choose starting values based on a plot since the program should run automatised. How could you modify your program to identify the global maximum even in the presence of other local optima?

Answer:

we split -4:4 to 80 equal sub-intervals,and try to find all the possible θ, then choose the max value of log_likelihood,and this one will be the global maximum. But this method still have a disadvantage. If the width of the sub-intervals is not small enough and it contains a bell curve, this method can not guarantee the return value is a global maximum.

```r
#----------------------------------------
# Code for question 1d
#----------------------------------------

# init the sub-intervals
xls <- seq(-4, 3.9, by = 0.1)
xus <- seq(-3.9, 4, by = 0.1)

# get the number of sub-intervals
n <- length(xls)

# init max_val and val to minimal value
max_val <- .Machine$double.xmin
val <- .Machine$double.xmin

# use loop to calculate all the possible theta(local maximum)
for(i in 1:n){
  k <- 0
  tmp_theta <- bisection_method(xl=xls[i],xu=xus[i],k=k)
  if (!(is.null(val))){
    val <- log_likelihood(x,tmp_theta)
    # get the max of local maximum
    if (val > max_val){
      max_val <- val
      ret <- tmp_theta
    }
  }
}
return(ret)
```

```
## [1] -0.2952455
```

Question 2: Computer arithmetics (variance) (Solved by Qinyuan Qi)

A known formula for estimating the variance based on a vector of n observations is

$$Var(\vec{x}) = \frac{1}{n-1}(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2)$$

a. Write your own R function, myvar, to estimate the variance in this way.

Answer:

The following code implemented $myvar()$ function to estimate the variance

```
#----------------------------------------
# Code for question 2a
#----------------------------------------
myvar <- function(X) {
  # get the length of X
  n <- length(X)
  # check n to make sure n > 2
  if (n < 2) return(0)
  # calculate ret based on the formula given
  ret <-  (1 / (n-1) ) * (sum(X^2) - ( 1 / n ) * sum(X)^2)
  return(ret)
}
```

b. Generate a vector $x = (x_1, ..., x_{10000})$ with 10000 random numbers with mean 10^8 and variance 1.

Answer:

We create a function $generate_vector()$ as follows which using built-in function $rnorm$.

```
#----------------------------------------
# Code for question 2b
#----------------------------------------
generate_vector <- function(vec_mean,vec_val,n){
  set.seed(24)
  ret <- rnorm(n, mean = vec_mean, sd = sqrt(vec_val))
  return(ret)
}

random_vector <- generate_vector(vec_mean = 10^8, vec_val = 1, n = 10000)
```

c. For each subset $Xi = x_1, ..., x_i, i = 1, ..., 10000$ compute the difference $Y_i = myvar(Xi) - var(X_i)$, where $var(X_i)$ is the standard variance estimation function in R. Plot the dependence Y_i on i. Draw conclusions from this plot. How well does your function work? Can you explain the behavior?

Answer: The function and the plot are give below.

According to plot, we can get the conclusion:

According to the 1st graph, when n get bigger,difference between $myvar()$ and $var()$ is more likely slower converge to zero or -2 when n increase. The reason is when n increase, the change of the variance is will get slower.

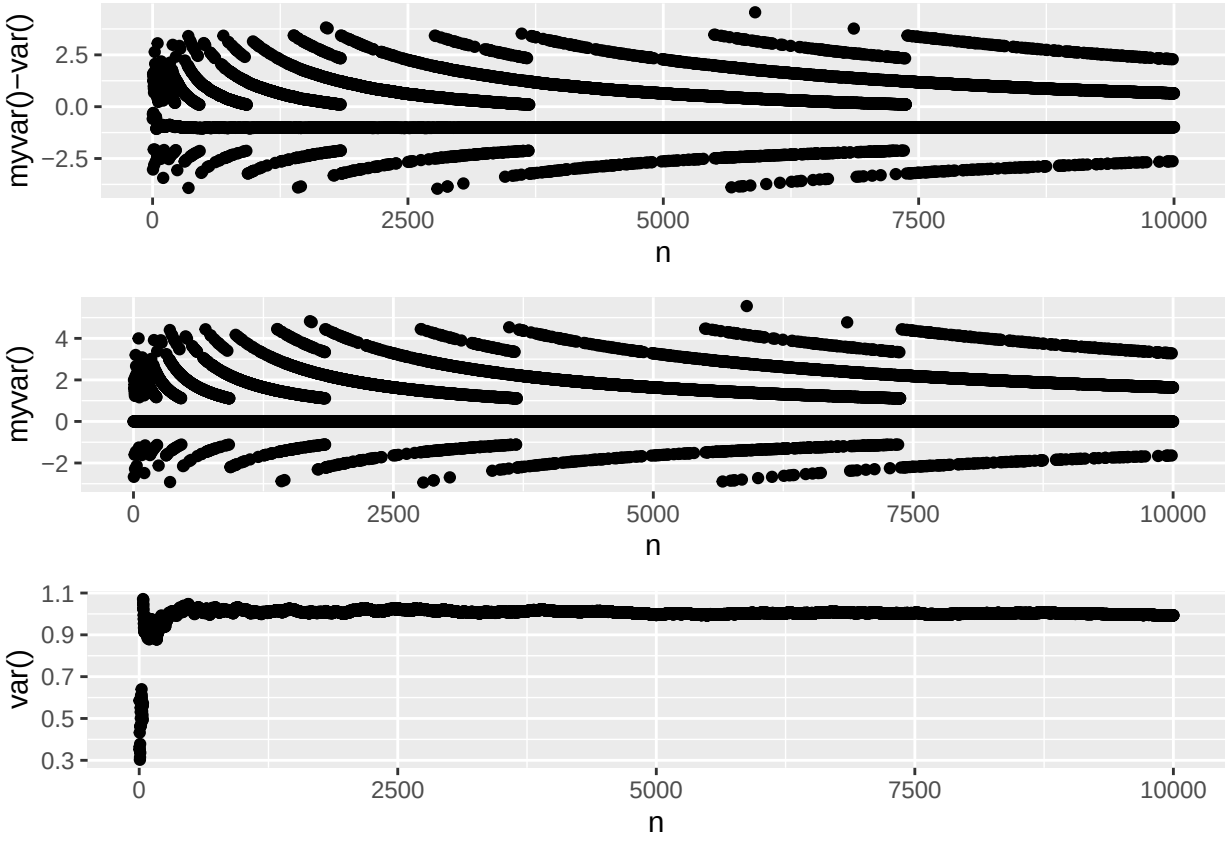
And we also notice that there is a line at around $y \approx -1$, means the difference is around -1.

According to the 2nd($myvar()$) and 3rd($var()$) graphs, we know that the original $var()$ has a mean around 1 and \$myvar() \$ has a mean around 0, which is the reason why there have a line around -1 in graph 1.

```r
#----------------------------------------
# Code for question 2c1
#----------------------------------------
plot_graph <- function(x,random_vector){
  n <- length(x)
  y1 = numeric(n)
  # loop to get the difference of value of myvar() and var()
  for(i in 1:n){
      y1[i] <- myvar(random_vector[1:i]) - var(random_vector[1:i])
  }

  #plot the data
  data <- data.frame(x,y1)
  return(ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=x,y=y1)) +
         ggplot2::geom_point()+
         ggplot2::labs(

          y = "myvar()-var()",
          x = "n"
          )
         )
}


#----------------------------------------
# Code for question 2c2
#----------------------------------------
plot_graph_original <- function(x,random_vector,original=FALSE){
  n <- length(x)
  y2=numeric(n)

  # loop to get the difference of value of myvar() and var()
  for(i in 1:n){
    if (original){
      y2[i] <- var(random_vector[1:i])
    }else{
      y2[i] <- myvar(random_vector[1:i])
    }
  }
  if (original){
    ylabel <- "var()"
  }else{
    ylabel <- "myvar()"
  }
  #plot the data
  data <- data.frame(x,y2)
  return(ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=x,y=y2)) +
         ggplot2::geom_point()+
         ggplot2::labs(

          y = ylabel,
          x = "n"
          )
```

```
        )
}

#----------------------------------------
# Function call to draw graph
#----------------------------------------
n = 10000
x=1:n
# difference between myvar and var
p1 <- plot_graph(x=x,random_vector = random_vector)
# myvar
p2 <- plot_graph_original(x=x,random_vector = random_vector,original=FALSE)
# var
p3 <- plot_graph_original(x=x,random_vector = random_vector,original=TRUE)

#plot them
gridExtra::grid.arrange(p1, p2,p3, heights = c(.34,.34,.32))
```

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
## Removed 1 rows containing missing values (`geom_point()`).
```



d. How can you better implement a variance estimator? Find and implement a formula that will give the same results as $var()$**?**

Answer:

According to the wiki page $Algorithms for calculating variance$ (https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance), we will adopt Welford's online algorithm to calculate new $newvar()$. According

to the plot, it looks same as the build in function $var()$.

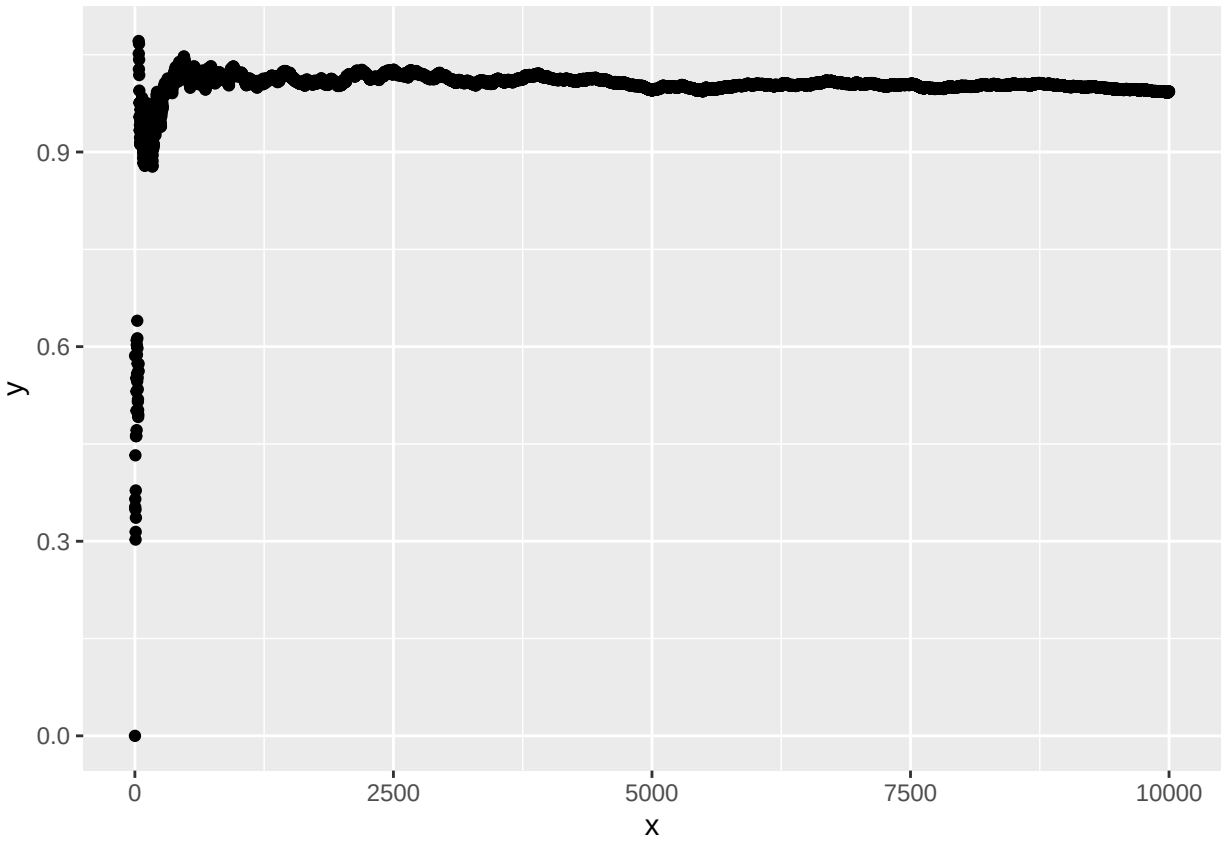
The formula used in this function is as follows, $\sum_{i=1}^{n}(x_i - \bar{x}_n)^2$ denoted by $M_{2,n}$.

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

$$\sigma_n^2 = \frac{M_{2,n}}{n}$$

$$s_n^2 = \frac{M_{2,n}}{n-1}$$

```
#----------------------------------------
# Code for question 2d
#----------------------------------------
newvar <- function(X) {
  n <- length(random_vector)
  y <- numeric(n)

  m <- 0
  x_mean <- random_vector[1]
  sigma2 <- 0
  s2 <- 0
  y[1] <- s2
  for(i in 2:n){
    x_mean_new <- mean(random_vector[1:i])
    m <- m + (random_vector[i] - x_mean) * (random_vector[i] - x_mean_new)
    sigma2 <- m / i
    s2 <- m / (i-1)
    y[i] <- s2

    x_mean <- x_mean_new
  }
  return(y)
}

y <- newvar(X=random_vector)
data <- data.frame(random_vector,y)
ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=x,y=y)) +
  ggplot2::geom_point()
```



Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
#----------------------------------------
# Code for question 1a
#----------------------------------------
# log_likelihood function
log_likelihood <- function(x, theta){
  n <- length(x)
  ret <- -n * log10(pi) - sum(log10(1+ (x-theta)^2))
  return(ret)
}

# derivative of log_likelihood function
log_likelihood_derivative <- function(x, theta){
  n <- length(x)
  ret <- -sum(2 * (x - theta) / (1 + (x - theta)^2))
  return(ret)
}

# init x ,theta
x <- c(-2.8, 3.4, 1.2, -0.3, -2.6)
thetas <- seq(-4, 4, by = 0.01)

# get length of theta
n <- length(thetas)
```

```r
# init vectors of value of log_likelihood() and log_likelihood_derivative()
y1 <- numeric(n)
y2 <- numeric(n)

# loop to call log_likelihood and log_likelihood_derivative functions
for(i in 1:n){
  y1[i] <- log_likelihood(x,thetas[i])
  y2[i] <- log_likelihood_derivative(x,thetas[i])
}

# plot data
data <- data.frame(thetas,y1,y2)
ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=thetas,y=y1)) +
  ggplot2::geom_line()

ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=thetas,y=y2)) +
  ggplot2::geom_line()
#----------------------------------------
# Code for question 1b
#
# bisection
#----------------------------------------

# init values to stop recursion
epsilon <- 1e-10
max_k <- 1000

# implementation of bisection
bisection_method <- function(xl,xu,k){

  # check k is not greater than max_k
  # if it's too big then stop recursion
  # this is mostly caused if the selected interval is a increasing interval or decreasing interval
  if (k > max_k) return(NULL)

  # get the mid point
  mid <- (xl + xu) / 2

  # calculate values of log_likelihood_derivative
  f_mid <- log_likelihood_derivative(x,mid)
  f_xl <- log_likelihood_derivative(x,xl)

  # check signs
  if (sign(f_mid) == sign(f_xl)){
    xl <- mid
  }else{
    xu <- mid
  }

  # if the distance between f_mid and 0 is small enough, then we got the answer.
  # else we need to split the interval and call the function again to find new answer.
  if (abs(f_mid) < epsilon){
    return(mid)
```

```
  }else(
    bisection_method(xl,xu,k+1)
  )
}

#----------------------------------------
# Test code
#----------------------------------------
#k <- 0
#xl <- -4
#xu <- 4
#ret <- bisection_method(xl=xl,xu=xu,k=k)

#----------------------------------------
# Code for question 1c
#----------------------------------------
# init values
k <- 0
xl <- -1
xu <- 0

# function call to bisection_method()
ret <- bisection_method(xl=xl,xu=xu,k=k)

# check ret to make sure bisection_method will not return null
# which will caused by a increasing interval or decreasing interval
if (is.null(ret)){
  print("Not found")
}else{
  print(ret)
}
#----------------------------------------
# Code for question 1d
#----------------------------------------

# init the sub-intervals
xls <- seq(-4, 3.9, by = 0.1)
xus <- seq(-3.9, 4, by = 0.1)

# get the number of sub-intervals
n <- length(xls)

# init max_val and val to minimal value
max_val <- .Machine$double.xmin
val <- .Machine$double.xmin

# use loop to calculate all the possible theta(local maximum)
for(i in 1:n){
  k <- 0
  tmp_theta <- bisection_method(xl=xls[i],xu=xus[i],k=k)
  if (!(is.null(val))){
    val <- log_likelihood(x,tmp_theta)
    # get the max of local maximum
```

```
    if (val > max_val){
      max_val <- val
      ret <- tmp_theta
    }
  }
}
return(ret)
#----------------------------------------
# Code for question 2a
#----------------------------------------
myvar <- function(X) {
  # get the length of X
  n <- length(X)
  # check n to make sure n > 2
  if (n < 2) return(0)
  # calculate ret based on the formula given
  ret <-  (1 / (n-1) ) * (sum(X^2) - ( 1 / n ) * sum(X)^2)
  return(ret)
}
#----------------------------------------
# Code for question 2b
#----------------------------------------
generate_vector <- function(vec_mean,vec_val,n){
  set.seed(24)
  ret <- rnorm(n, mean = vec_mean, sd = sqrt(vec_val))
  return(ret)
}

random_vector <- generate_vector(vec_mean = 10^8, vec_val = 1, n = 10000)

#----------------------------------------
# Code for question 2c1
#----------------------------------------
plot_graph <- function(x,random_vector){
  n <- length(x)
  y1 = numeric(n)
  # loop to get the difference of value of myvar() and var()
  for(i in 1:n){
      y1[i] <- myvar(random_vector[1:i]) - var(random_vector[1:i])
  }

  #plot the data
  data <- data.frame(x,y1)
  return(ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=x,y=y1)) +
         ggplot2::geom_point()+
         ggplot2::labs(

          y = "myvar()-var()",
          x = "n"
          )
         )
}
```

```r
#----------------------------------------
# Code for question 2c2
#----------------------------------------
plot_graph_original <- function(x,random_vector,original=FALSE){
  n <- length(x)
  y2=numeric(n)

  # loop to get the difference of value of myvar() and var()
  for(i in 1:n){
    if (original){
      y2[i] <- var(random_vector[1:i])
    }else{
      y2[i] <- myvar(random_vector[1:i])
    }
  }
  if (original){
    ylabel <- "var()"
  }else{
    ylabel <- "myvar()"
  }
  #plot the data
  data <- data.frame(x,y2)
  return(ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=x,y=y2)) +
         ggplot2::geom_point()+
         ggplot2::labs(

          y = ylabel,
          x = "n"
          )
         )
}

#----------------------------------------
# Function call to draw graph
#----------------------------------------
n = 10000
x=1:n
# difference between myvar and var
p1 <- plot_graph(x=x,random_vector = random_vector)
# myvar
p2 <- plot_graph_original(x=x,random_vector = random_vector,original=FALSE)
# var
p3 <- plot_graph_original(x=x,random_vector = random_vector,original=TRUE)

#plot them
gridExtra::grid.arrange(p1, p2,p3, heights = c(.34,.34,.32))
#----------------------------------------
# Code for question 2d
#----------------------------------------
newvar <- function(X) {
  n <- length(random_vector)
  y <- numeric(n)
```

```
  m <- 0
  x_mean <- random_vector[1]
  sigma2 <- 0
  s2 <- 0
  y[1] <- s2
  for(i in 2:n){
    x_mean_new <- mean(random_vector[1:i])
    m <- m + (random_vector[i] - x_mean) * (random_vector[i] - x_mean_new)
    sigma2 <- m / i
    s2 <- m / (i-1)
    y[i] <- s2

    x_mean <- x_mean_new
  }
  return(y)
}

y <- newvar(X=random_vector)
data <- data.frame(random_vector,y)
ggplot2::ggplot(data = data,mapping = ggplot2::aes(x=x,y=y)) +
  ggplot2::geom_point()
```