732A90 Computational Statistics

Lab 1

Johannes Hedström & Mikael Montén

STIMA
Department of Computer and Information Science
Linköpings universitet

2023-11-06

Contents

1 Task 1

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:

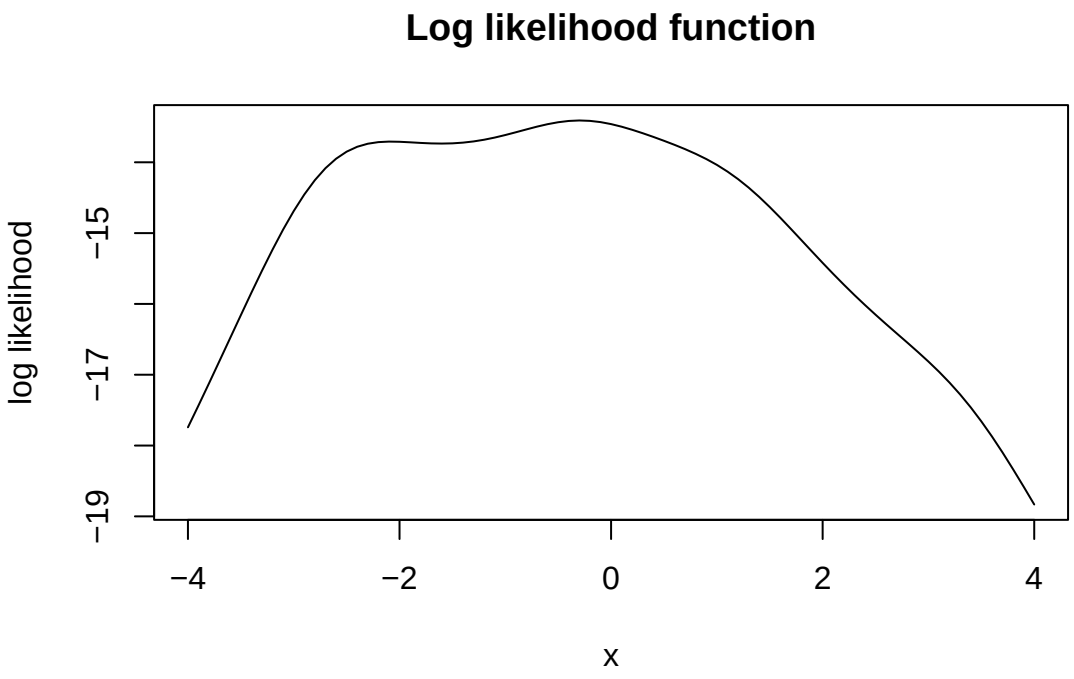
$$\sum_{i=1}^{n} \frac{2(x_i - \theta)}{1 + (x_i - \theta)^2}$$

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

1.1 a

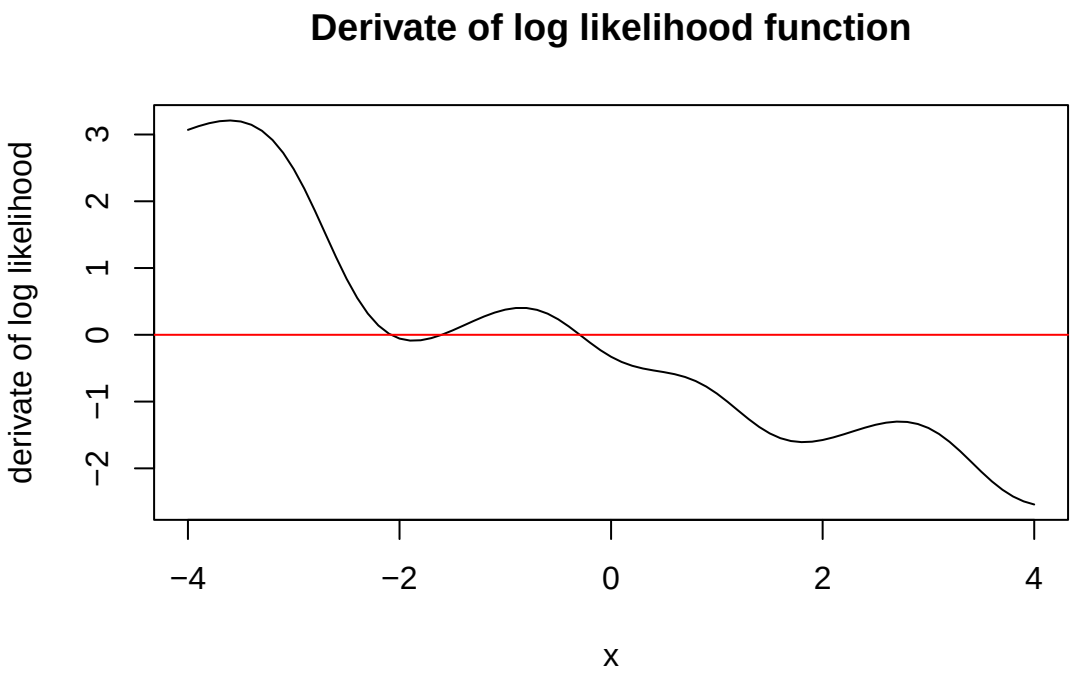
Plot the log likelihood function for the given data in the range from -4 to 4. Plot the derivative in the same range and check visually how often the derivative is equal to 0.

```
# size
n <- 5
# given x vector
x <- c(-2.8,3.4,1.2,-0.3,-2.6)
loc <- seq(-4,4,0.1)


## a
res <- c()

# log likelihood calculations
for (i in loc) {


    logli <- -n * log(pi) - sum(log(1 + (x - i)^2))

    res <- c(res,logli)
}




plot(y=res, x=loc, type='l',ylab='log likelihood',xlab='x',
     main='Log likelihood function')
```



There looks to be a local max and min and a the maximum of the log likelihood function for the given data and range.

```
# log likelihood derivative calculations

res_der <- c()
for (i in loc) {


  der <- sum((2 * (x - i))/(1 + (x - i)^2))

  res_der <- c(res_der,der)
}

# plotting the derivative
plot(y=res_der, x=loc, type='l',ylab='derivate of log likelihood',xlab='x',
     main='Derivate of log likelihood function')

# adding a line for 0
abline(h=0, col='red')
```



The derivative of the log likelihood functions for the given data is zero three times, which can be seen when the line crosses the horizontal red line in the plot.

1.2 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.

```
bisection <- function(x,a0,b0, criterion){
      ga0 <- sum((2 * (x - a0))/(1 + (x - a0)^2))

      gb0 <- sum((2 * (x - b0))/(1 + (x - b0)^2))

      # checking if there is a derivate of 0 in the intervall
      if(ga0 * gb0 >= 0)stop('Start with interval [a0,b0] such that g`(a0) * g(b0) < 0')

      # which is the biggest one if the user made an input error
      if(!(ga0 >= gb0))stop('Check that the intervall have atleast 1 maximum')

      a <- a0
      b <- b0

      # objects an calculations for the loop
      int_mean <- (a0+b0)/2
```

```
      t <- 1
      g_x <- sum((2 * (x - int_mean))/(1 + (x - int_mean)^2))
      absolute_c_c <- 1
      int_mean_c <- c(int_mean)

      # iterate until criterion is met
      while (absolute_c_c > criterion) {
        if(g_x==0)break # if we found the max
        if(t >= 100)break # Max amount if iterations so the loop stops if the criterion is too low

        if (g_x > 0) { # checking if its positive or negative
          a <- int_mean
        }else{
          b <- int_mean
        }
        # new location
        int_mean <- (a+b)/2
        # adding it to the vector
        int_mean_c <- c(int_mean_c,int_mean)
        # computing the derivative of the new location
        g_x <- sum((2 * (x - int_mean))/(1 + (x - int_mean)^2))

        # new absolute difference for the new loc and the previous
        absolute_c_c <- abs(int_mean_c[t+1] - int_mean_c[t])

        t <- t+1 # iteration
      }
      return(list('pos'=int_mean,'iterate'=t)) # returning a list with the position of the maxima and nu
      # iterates to find it
}

# example
a0 <- -3

b0 <- 4

bisection(x,a0,b0, 0.01)
```

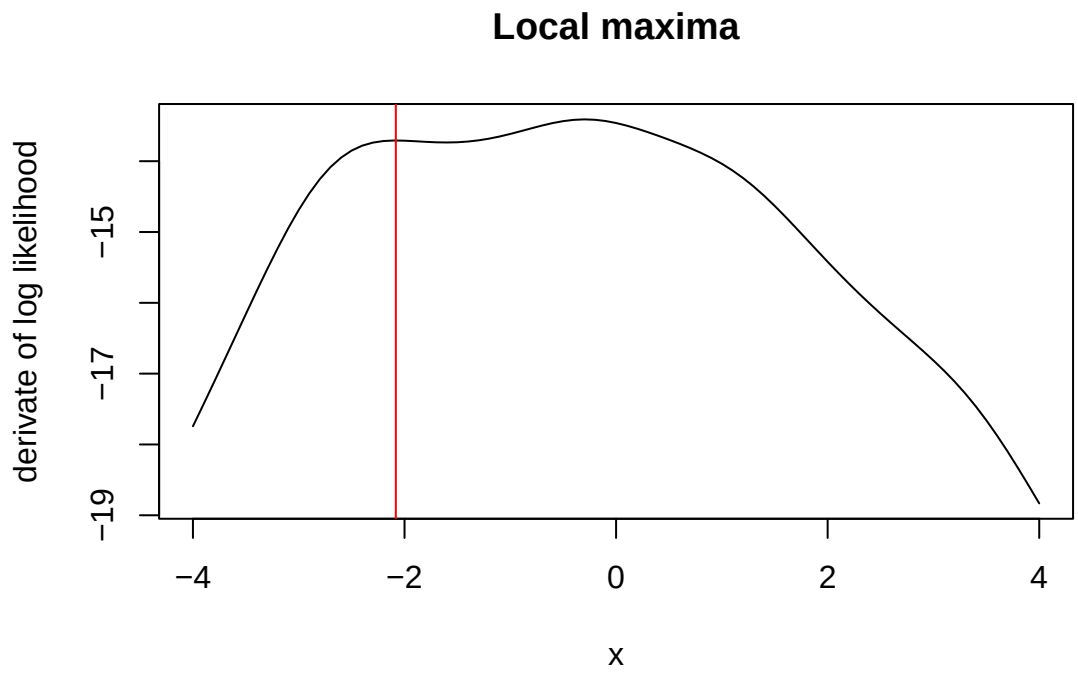
```
## $pos
## [1] -0.2998047
##
## $iterate
## [1] 10
```

1.3 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.

```
### lokal maximum interval a0 = -4 , b0 = -2
start1<- bisection(x,-4,-2, 0.000001)


plot(y=res, x=loc, type='l',ylab='derivate of log likelihood',xlab='x',
     main='Local maxima')

# adding a line for 0
abline(v=start1$pos, col='red')
```

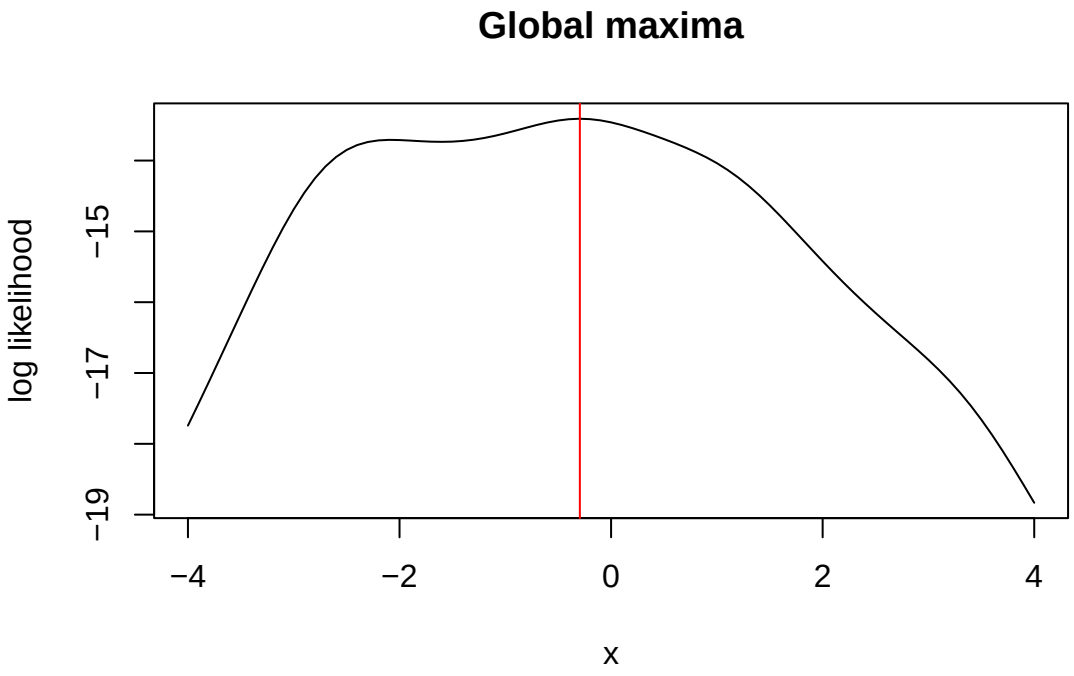



```
-n * log(pi) - sum(log(1 + (x - start1$pos)^2))
```

```
## [1] -13.7077
```

```
### maximum interval a0 = -1 , b0 = 2
start2 <-bisection(x,-1,2, 0.00000001)
# Some pairs doesn't fulfill the criterion of the bisection method
# "Start with interval [a0,b0] such that g`(a0) * g(b0) < 0'" like -4 and -3
#bisection(x,-4,-3, 0.000001) # == ERROR
```

```
plot(y=res, x=loc, type='l',ylab='log likelihood',xlab='x',
     main='Global maxima')

# adding a line for 0
abline(v=start2$pos, col='red')
```




```
-n * log(pi) - sum(log(1 + (x - start2$pos)^2))
```

```
## [1] -13.40942
```

We see that the global maxima when computing the log likelihood of x for that position and the last output has the highest value.

1.4 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?

One way to identify the global maximum would be to have the algorithm start at values min(x) and max(x), and use the bisection to find the first maximum and save the log likelihood value for that x. Subsequentially set that mean value x0 as your a0 and max(x) as b0, then iteratively find maximums, saving the log likelihoods

until the minimum value has iterated close enough according to some criterion to the maximum value. After you have found your unknown maximums for the right half of the value space, you do the same for the left half by setting the first maximum x0 as your b0 and min(x) as a0 then repeating the process until both sides of the last maximum returns values g‘(a0) * g(b0) >= 0. This would result in you having a vector with different log likelihood values, from which you choose the highest one which then is your global maximum.

2 Task 2

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

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

2.1 a

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

```
# creating my function
myvar <- function(x){
      # creating n
      n <- length(x)
      # the first part of the calculations
      div <- 1 / (n - 1)

      # second part of the calculations
      summ <- sum(x^2) - ((1 / n) * (sum(x)^2))

      # product of them
      variance <- div * summ

      # returning my variance
      variance

}

myvar(x)
```

```
## [1] 6.862
```

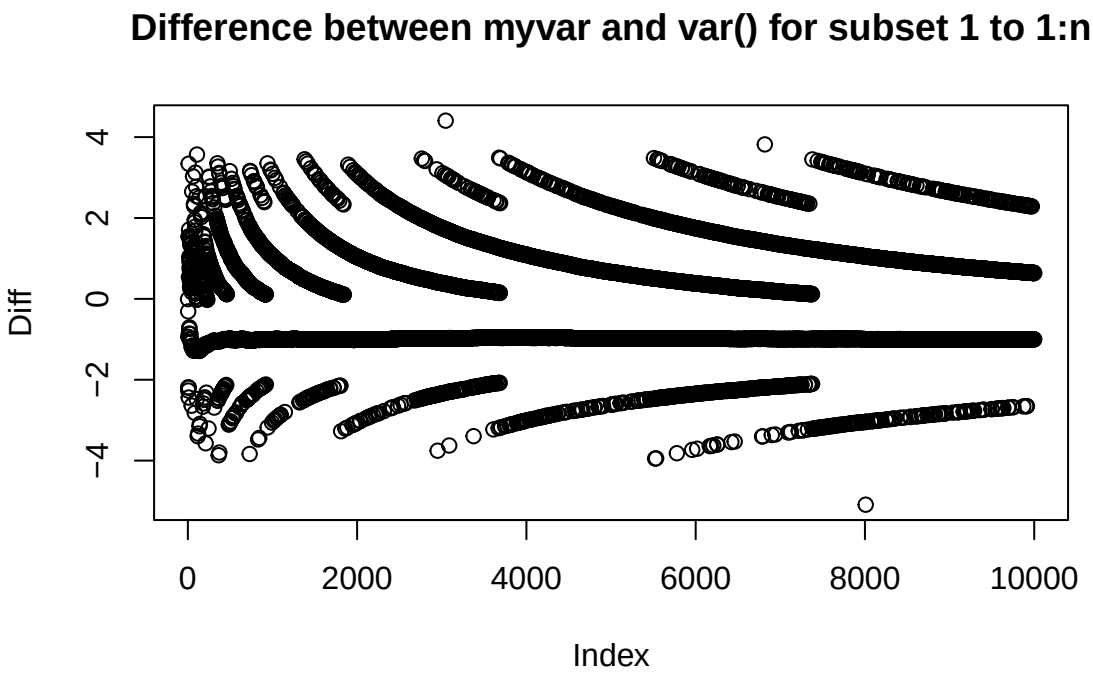
2.2 b

Generate a vector $x = (x_1, ..., x_1 0000)$ *with 10000 random numbers with mean* 10^8 *and variance 1.*

```r
set.seed(12345) # seed for the same results every knit
x_vec <- rnorm(10000,mean=10^8,sd=1) # the vector
```

2.3 c

```r
yi <- c()

for(i in 1:10000){
  # calculating the difference (i=1 will return NA)
yi[i] <- myvar(x_vec[1:i]) - var(x_vec[1:i])

}

plot(yi, main='Difference between myvar and var() for subset 1 to 1:n',
     ylab = 'Diff')
```



The difference gets closer to 0 the more observations we measure from, the 'line' is under 0 which might indicate that our function are undershooting the variance a bit, but at the same time the 'curves' looks to be more dense on the positive side which might lead the average close to 0.

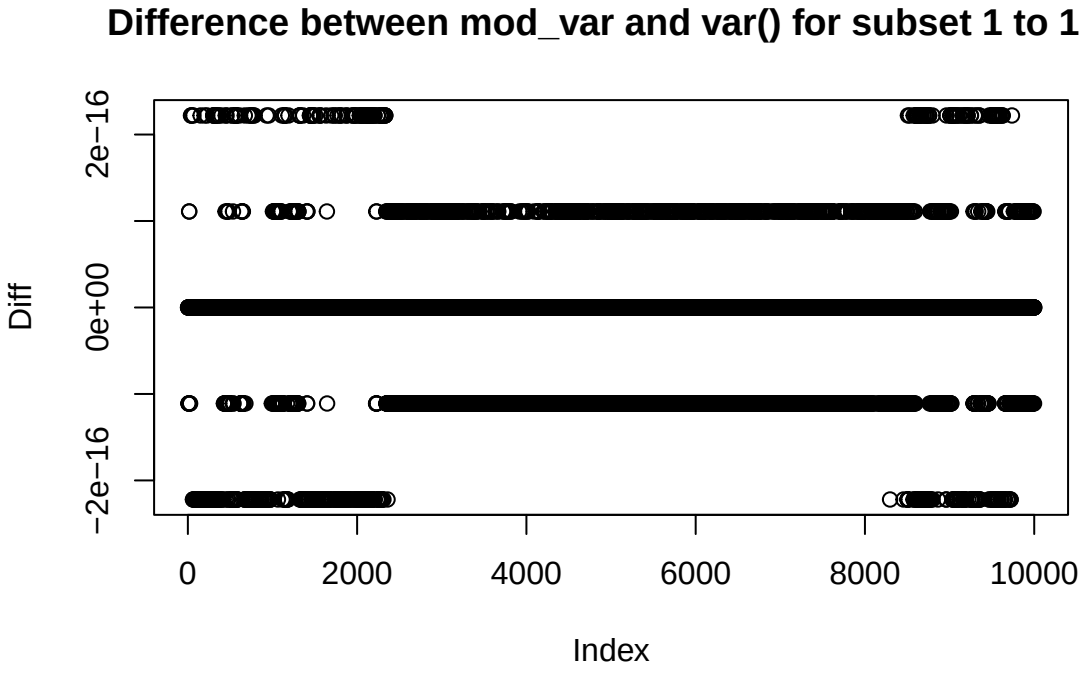
So our function might be performing good and the difference between the functions could be accumulating rounding error of the calculations when working with numbers of this size. And the summarizing of several

numbers(especially really large or small ones) can affect the precision(making it worse), and as our function has two summations this might be causing the differences between our function and var()(Gentle 2009). Kahan summation algorithm might help to make this closer to the var()-function.

2.4 d

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

```
mod_var <- function(x){
    # creating n
    n <- length(x)

    variance <- sum((x - mean(x))^2) /(n-1)

    variance
}

yi2 <- c()

for(i in 1:10000){
  # calculating the difference (i=1 will return NA)
  yi2[i] <- mod_var(x_vec[1:i]) - var(x_vec[1:i])

}

plot(yi2, main='Difference between mod_var and var() for subset 1 to 1:n',
     ylab = 'Diff')
```



If we calculate the mean of x we can calculate the variance as this("Variance" 2023):

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

And now we have fewer steps to calculate the variance which seem to reduce the accumulating rounding error of the calculations close to zero but we still seem to have some catastrophic cancellation error, and the digits left are probably rounding differences (Gentle 2009).

3 References

Gentle, James E. 2009. "Computational Statistics." https://doi.org/https://doi.org/10.1007/978-0-387-98144-4.

"Variance." 2023. Wikipedia. https://en.wikipedia.org/wiki/Variance.

4 Appendix

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning=FALSE, fig.width = 6, fig.height = 4)
# size
n <- 5
```

```r
# given x vector
x <- c(-2.8,3.4,1.2,-0.3,-2.6)
loc <- seq(-4,4,0.1)


## a
res <- c()

# log likelihood calculations
for (i in loc) {


    logli <- -n * log(pi) - sum(log(1 + (x - i)^2))

    res <- c(res,logli)
}




plot(y=res, x=loc, type='l',ylab='log likelihood',xlab='x',
     main='Log likelihood function')



# log likelihood derivative calculations

res_der <- c()
for (i in loc) {


  der <- sum((2 * (x - i))/(1 + (x - i)^2))

  res_der <- c(res_der,der)
}

# plotting the derivative
plot(y=res_der, x=loc, type='l',ylab='derivate of log likelihood',xlab='x',
     main='Derivate of log likelihood function')

# adding a line for 0
abline(h=0, col='red')




bisection <- function(x,a0,b0, criterion){
      ga0 <- sum((2 * (x - a0))/(1 + (x - a0)^2))
```

```
      gb0 <- sum((2 * (x - b0))/(1 + (x - b0)^2))

      # checking if there is a derivate of 0 in the intervall
      if(ga0 * gb0 >= 0)stop('Start with interval [a0,b0] such that g`(a0) * g(b0) < 0')

      # which is the biggest one if the user made an input error
      if(!(ga0 >= gb0))stop('Check that the intervall have atleast 1 maximum')

      a <- a0
      b <- b0

      # objects an calculations for the loop
      int_mean <- (a0+b0)/2
      t <- 1
      g_x <- sum((2 * (x - int_mean))/(1 + (x - int_mean)^2))
      absolute_c_c <- 1
      int_mean_c <- c(int_mean)

      # iterate until criterion is met
      while (absolute_c_c > criterion) {
        if(g_x==0)break # if we found the max
        if(t >= 100)break # Max amount if iterations so the loop stops if the criterion is too low

        if (g_x > 0) { # checking if its positive or negative
          a <- int_mean
        }else{
          b <- int_mean
        }
        # new location
        int_mean <- (a+b)/2
        # adding it to the vector
        int_mean_c <- c(int_mean_c,int_mean)
        # computing the derivative of the new location
        g_x <- sum((2 * (x - int_mean))/(1 + (x - int_mean)^2))

        # new absolute difference for the new loc and the previous
        absolute_c_c <- abs(int_mean_c[t+1] - int_mean_c[t])

        t <- t+1 # iteration
      }
      return(list('pos'=int_mean,'iterate'=t)) # returning a list with the position of the maxima and nu
      # iterates to find it
}

# example
a0 <- -3

b0 <- 4
```

```r
bisection(x,a0,b0, 0.01)



### lokal maximum interval a0 = -4 , b0 = -2
start1<- bisection(x,-4,-2, 0.000001)


plot(y=res, x=loc, type='l',ylab='derivate of log likelihood',xlab='x',
     main='Local maxima')

# adding a line for 0
abline(v=start1$pos, col='red')

-n * log(pi) - sum(log(1 + (x - start1$pos)^2))


### maximum interval a0 = -1 , b0 = 2
start2 <-bisection(x,-1,2, 0.00000001)
# Some pairs doesn't fulfill the criterion of the bisection method
# "Start with interval [a0,b0] such that g`(a0) * g(b0) < 0'" like -4 and -3
#bisection(x,-4,-3, 0.000001) # == ERROR

plot(y=res, x=loc, type='l',ylab='log likelihood',xlab='x',
     main='Global maxima')

# adding a line for 0
abline(v=start2$pos, col='red')

-n * log(pi) - sum(log(1 + (x - start2$pos)^2))



# creating my function
myvar <- function(x){
      # creating n
      n <- length(x)
      # the first part of the calculations
      div <- 1 / (n - 1)

      # second part of the calculations
      summ <- sum(x^2) - ((1 / n) * (sum(x)^2))

      # product of them
      variance <- div * summ

      # returning my variance
      variance
```

```
}

myvar(x)


set.seed(12345) # seed for the same results every knit
x_vec <- rnorm(10000,mean=10^8,sd=1) # the vector

yi <- c()

for(i in 1:10000){
  # calculating the difference (i=1 will return NA)
yi[i] <- myvar(x_vec[1:i]) - var(x_vec[1:i])

}

plot(yi, main='Difference between myvar and var() for subset 1 to 1:n',
     ylab = 'Diff')


mod_var <- function(x){
    # creating n
    n <- length(x)

    variance <- sum((x - mean(x))^2) /(n-1)

    variance
}

yi2 <- c()

for(i in 1:10000){
  # calculating the difference (i=1 will return NA)
  yi2[i] <- mod_var(x_vec[1:i]) - var(x_vec[1:i])

}

plot(yi2, main='Difference between mod_var and var() for subset 1 to 1:n',
     ylab = 'Diff')
```