STATISTICAL METHODS FOR DATA SCIENCE - MINI PROJECT 3

Names of group members: 1. Manneyaa Jayasanker Net ID: MXJ180040
2. Venkatesh Sankar Net ID: VXS200014

Contribution of each group member:

Manneyaa Jayasanker:
• Worked on R code for 1(a) , 1(b), 1(c) and 1(d).
• Worked on conclusion for 1(a) , 1(b), 1(c) and 1(d).
• Wrote documentation for 1(a) , 1(b) , 1(c) and 1(d).

Venkatesh Sankar:
• Worked on R code for 2(a), 2(b), 2(c) and 2(d).
• Worked on conclusion for 2(a), 2(b), 2(c) and 2(d).
• Wrote documentation for 2(a), 2(b), 2(c) and 2(d).

==

1a)

The Monte Carlo simulations can be generated using the uniform random distribution function runif with parameters (n,0, θ). Here, n is the number of observations. The range [0, θ] represents the minimum and maximum values respectively. We use this variable to find the maximum likelihood estimator(θ_1) and moment of method(θ_2). We then generate the estimate by creating 1000 draws and calculate square of the difference between the estimated value and the actual value.In order to calculate the Mean Squared Error we have to take the following steps:

i. Select a random sample of size 'n' of uniform distribution with parameters 0 and θ using the runif function.

   data = runif(n,0,θ)

ii. Calculate the estimated values of $\theta_1$and θ_2 as per the given estimators.

   θ_1= max(data) and θ_2 = 2 * mean(data)

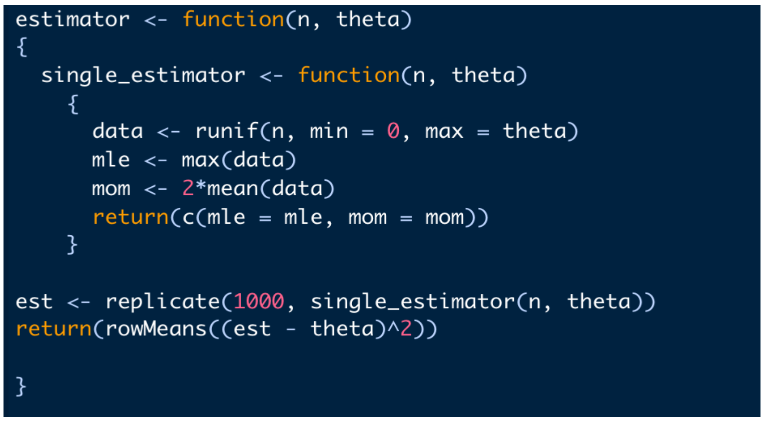
iii. Calculate the squared error for each estimator from θ, θ1, θ2 as follows:

   mse.mle = (θ_1-θ)^2

   mse.mom = (θ_2-θ)^2

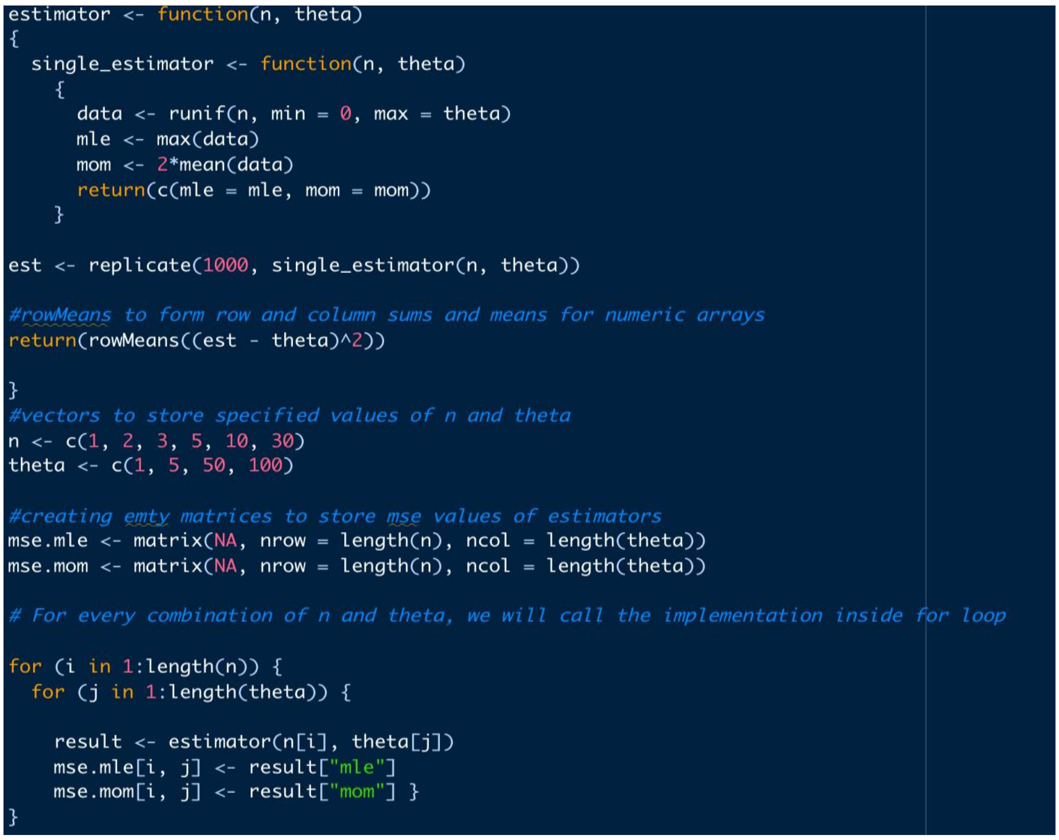
iv. Repeat the steps i-iii for N number of times to calculate the mean squared error.
v. The mean of errors for each of the estimators will give us the estimate of the MSE for both the estimators.

1b) For a given combination of (n, θ), to compute the mean squared errors of both mle and mome using Monte Carlo simulation with N = 1000 replications:

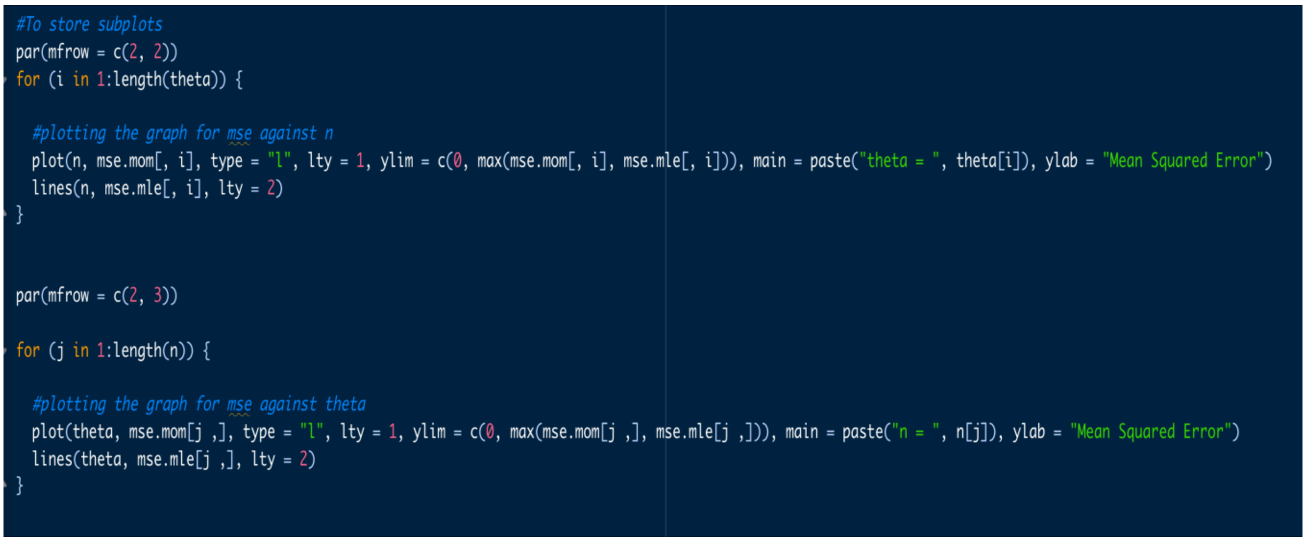


```
estimator <- function(n, theta)
{
  single_estimator <- function(n, theta)
    {
      data <- runif(n, min = 0, max = theta)
      mle <- max(data)
      mom <- 2*mean(data)
      return(c(mle = mle, mom = mom))
    }

est <- replicate(1000, single_estimator(n, theta))
return(rowMeans((est - theta)^2))

}
```

1C) We execute the following code for calculating MSE of the other combinations of n and θ. The results can be viewed in their graphical representation. To calculate the errors, we will call the function for 1000 times by passing different values for n and θ

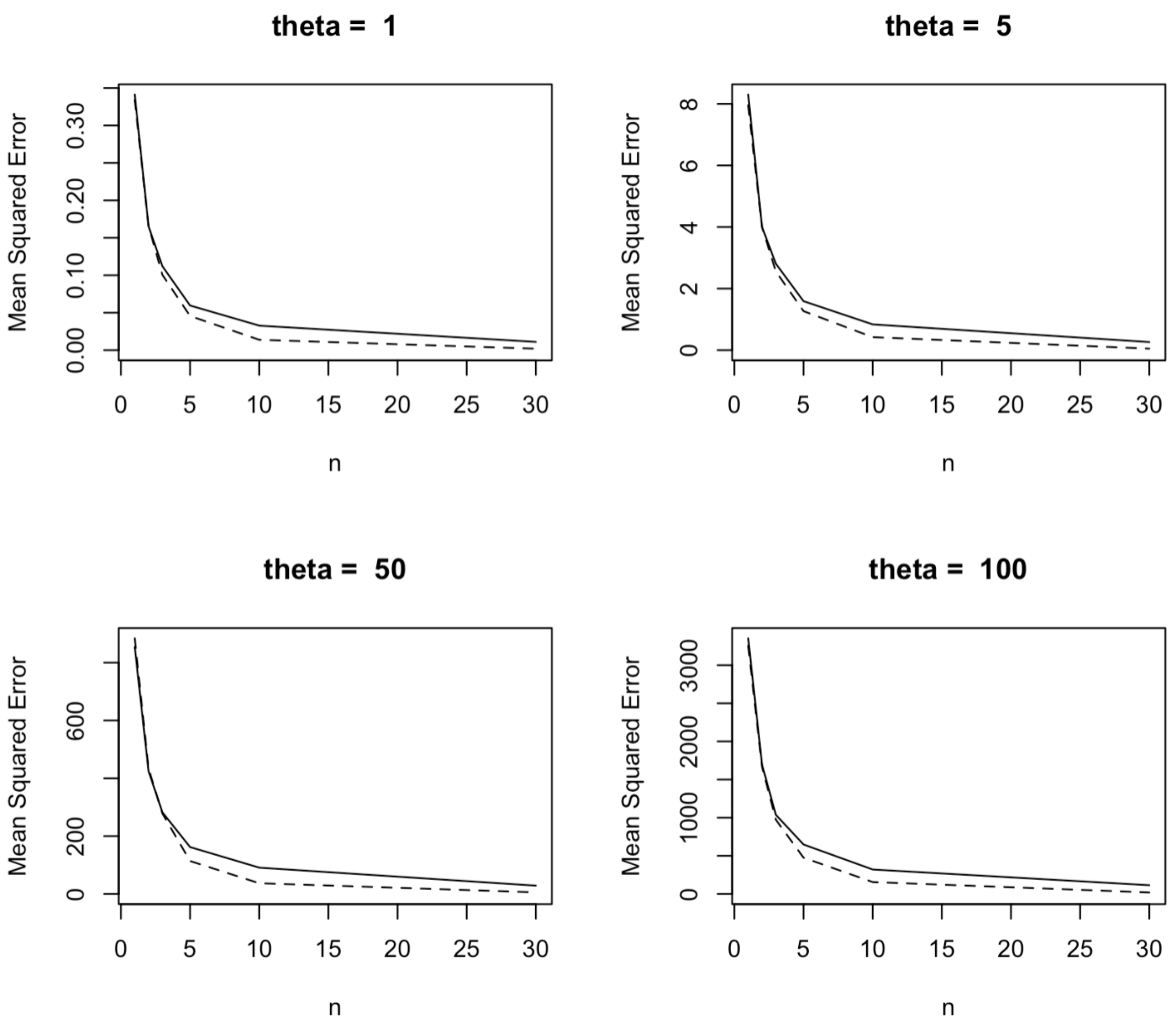


```
estimator <- function(n, theta)
{
  single_estimator <- function(n, theta)
    {
      data <- runif(n, min = 0, max = theta)
      mle <- max(data)
      mom <- 2*mean(data)
      return(c(mle = mle, mom = mom))
    }

est <- replicate(1000, single_estimator(n, theta))

#rowMeans to form row and column sums and means for numeric arrays
return(rowMeans((est - theta)^2))

}
#vectors to store specified values of n and theta
n <- c(1, 2, 3, 5, 10, 30)
theta <- c(1, 5, 50, 100)

#creating emty matrices to store mse values of estimators
mse.mle <- matrix(NA, nrow = length(n), ncol = length(theta))
mse.mom <- matrix(NA, nrow = length(n), ncol = length(theta))

# For every combination of n and theta, we will call the implementation inside for loop

for (i in 1:length(n)) {
  for (j in 1:length(theta)) {

    result <- estimator(n[i], theta[j])
    mse.mle[i, j] <- result["mle"]
    mse.mom[i, j] <- result["mom"] }
}
```

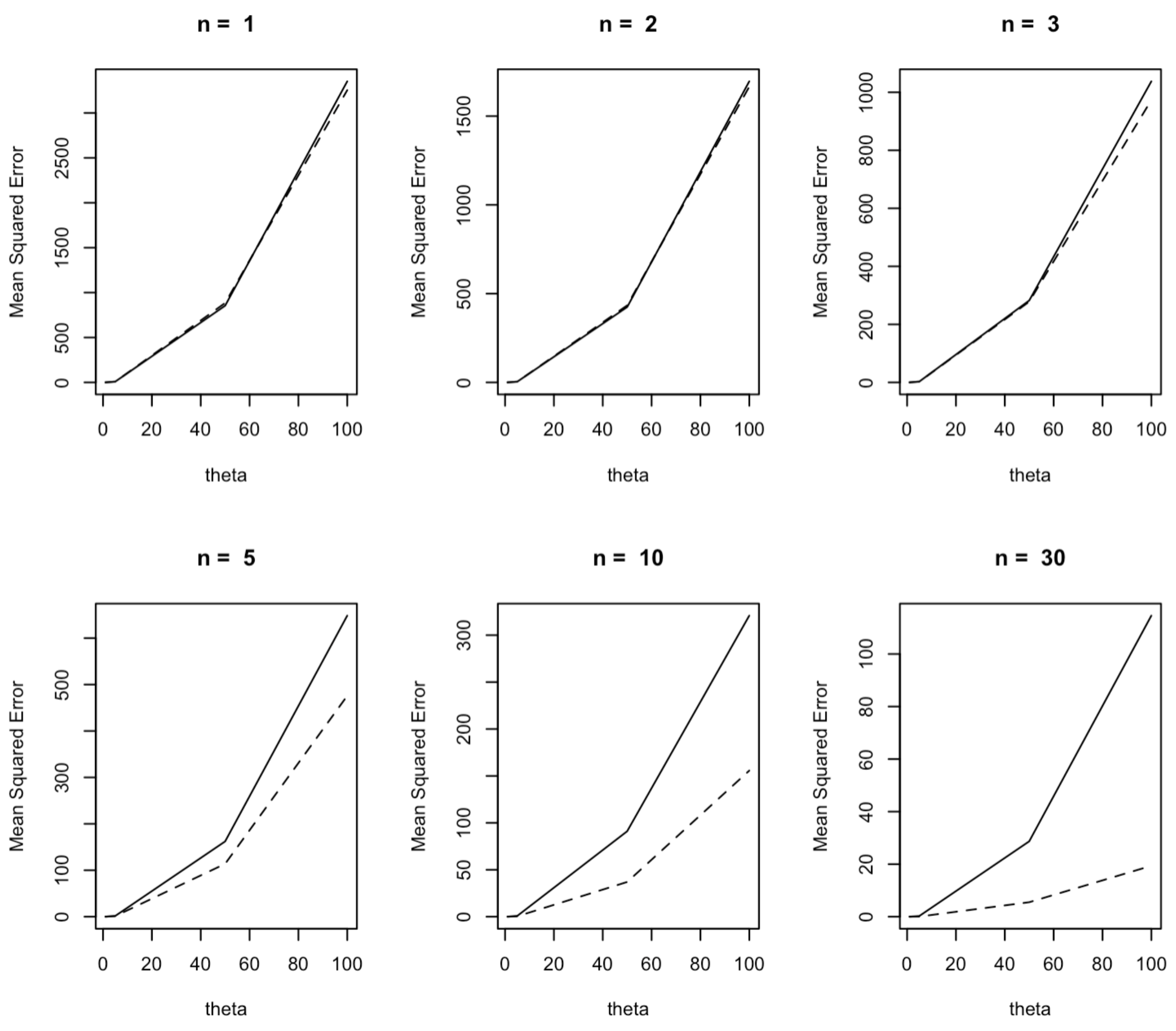


```r
#To store subplots
par(mfrow = c(2, 2))
for (i in 1:length(theta)) {

  #plotting the graph for mse against n
  plot(n, mse.mom[, i], type = "l", lty = 1, ylim = c(0, max(mse.mom[, i], mse.mle[, i])), main = paste("theta = ", theta[i]), ylab = "Mean Squared Error")
  lines(n, mse.mle[, i], lty = 2)
}


par(mfrow = c(2, 3))

for (j in 1:length(n)) {

  #plotting the graph for mse against theta
  plot(theta, mse.mom[j ,], type = "l", lty = 1, ylim = c(0, max(mse.mom[j ,], mse.mle[j ,])), main = paste("n = ", n[j]), ylab = "Mean Squared Error")
  lines(theta, mse.mle[j ,], lty = 2)
}
```

Graph 1 : MSEs of the MLE estimator and MOM estimator of θ as functions of n for various values of θ.



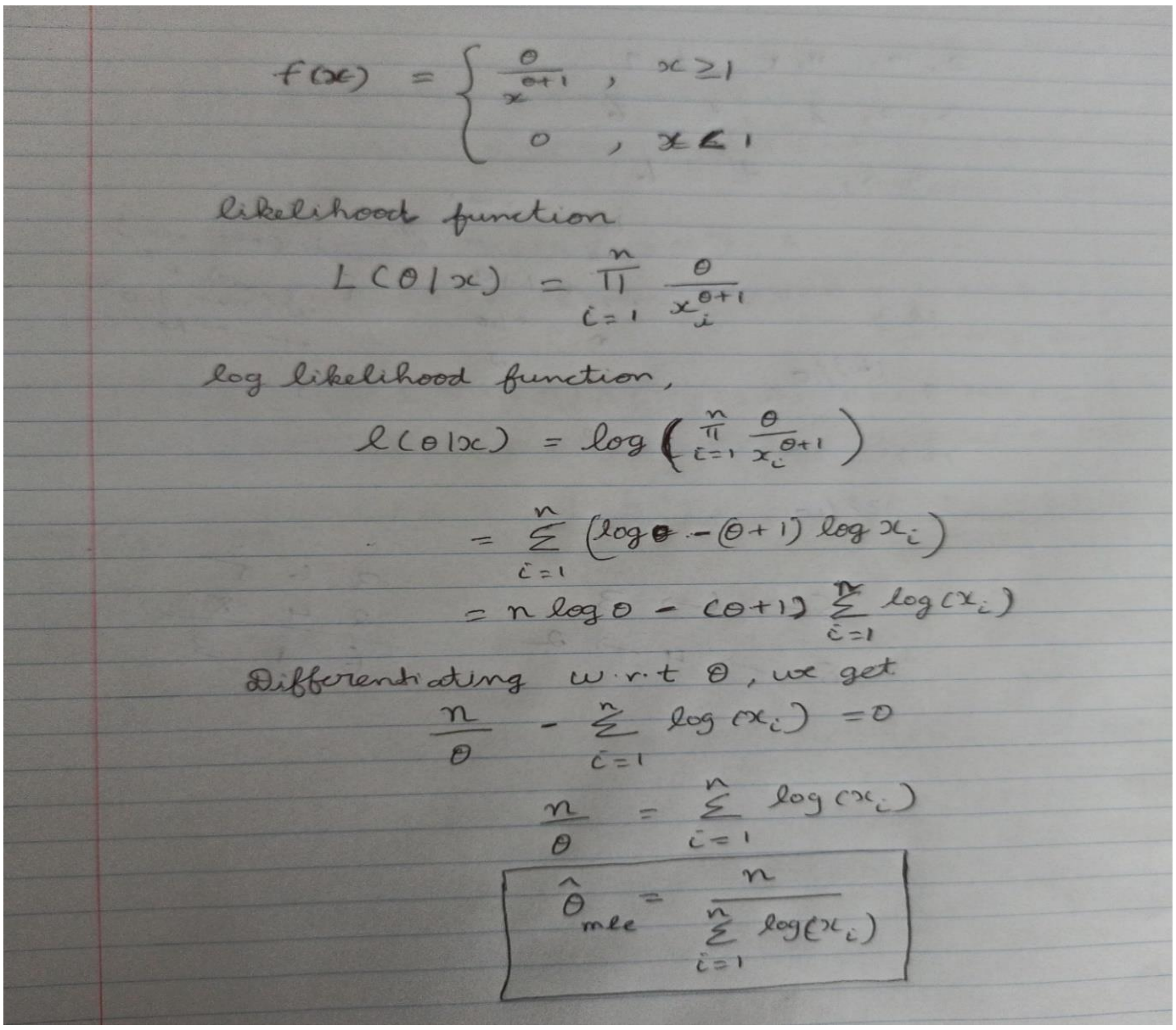
Graph 2 : MSEs of the MLE estimator and MOM estimator of θ as functions of θ for various values of n.



1d) From the graphs shown above(Graph 2), we can easily observe that for small values of n like 1 and 3, the performance of both the estimators MLE and Method of moment is almost the same. As soon as the value of n increases, we can notice a difference in the performance of these estimators. The Mean Squared Error of method of moment is more as compared to that of MLE, so we can conclude that θ1 is the better estimator from the two estimators. From Graph 1, we can say that as value of n increases, the MSE value decreases, i.e converges to true value. Thus, the estimation is closer to the actual values for higher values of n.

2.

a) Derivation of expression for maximum likelihood estimator of Θ based on given probability density function.



$$f(x) = \begin{cases} \frac{\theta}{x^{\theta+1}} , & x \geq 1 \\ 0 , & x \leq 1 \end{cases}$$

likelihood function

$$L(\theta | x) = \prod_{i=1}^{n} \frac{\theta}{x_i^{\theta+1}}$$

log likelihood function,

$$l(\theta|x) = \log\left(\prod_{i=1}^{n} \frac{\theta}{x_i^{\theta+1}}\right)$$

$$= \sum_{i=1}^{n} \left(\log \theta - (\theta+1) \log x_i\right)$$

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

Differentiating w.r.t θ, we get

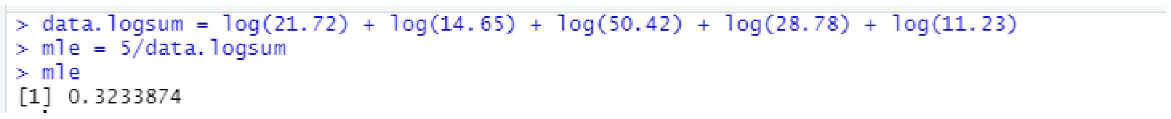
$$\frac{n}{\theta} - \sum_{i=1}^{n} \log (x_i) = 0$$

$$\frac{n}{\theta} = \sum_{i=1}^{n} \log (x_i)$$

$$\boxed{\hat{\theta}_{mle} = \frac{n}{\sum_{i=1}^{n} \log(x_i)}}$$

b) Maximum likelihood estimator of Θ based on values x1 = 21.72, x2 = 14.65, x3 = 50.42, x4 =28.78, x5 = 11.23.

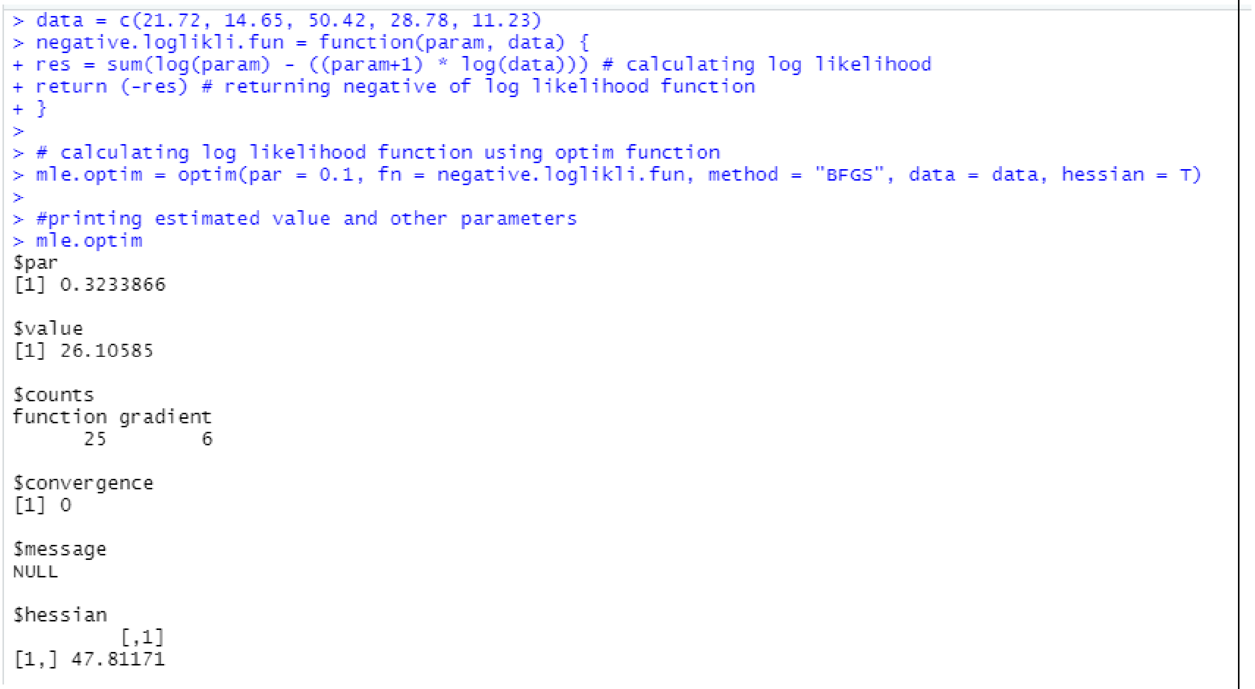
Rcode:



```
> data.logsum = log(21.72) + log(14.65) + log(50.42) + log(28.78) + log(11.23)
> mle = 5/data.logsum
> mle
[1] 0.3233874
```

c) Estimating log-likelihood function using optim function in R.

Rcode :

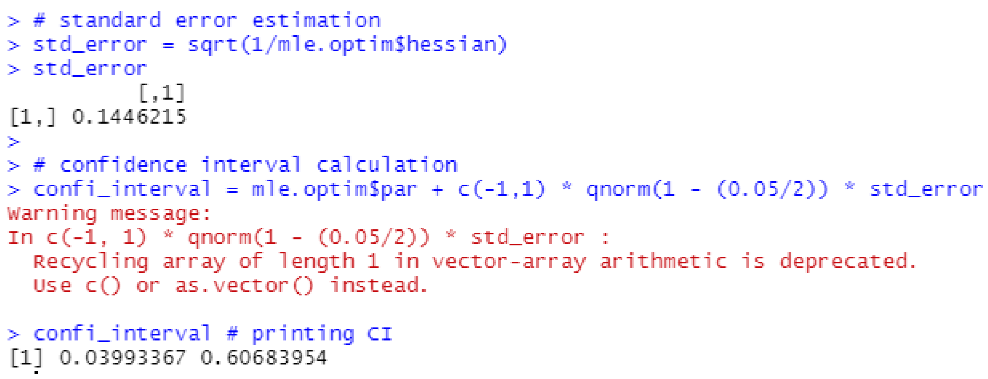


```
> data = c(21.72, 14.65, 50.42, 28.78, 11.23)
> negative.loglikli.fun = function(param, data) {
+ res = sum(log(param) - ((param+1) * log(data))) # calculating log likelihood
+ return (-res) # returning negative of log likelihood function
+ }
>
> # calculating log likelihood function using optim function
> mle.optim = optim(par = 0.1, fn = negative.loglikli.fun, method = "BFGS", data = data, hessian = T)
>
> #printing estimated value and other parameters
> mle.optim
$par
[1] 0.3233866

$value
[1] 26.10585

$counts
function gradient
      25        6

$convergence
[1] 0

$message
NULL

$hessian
         [,1]
[1,] 47.81171
```

The estimated value of the parameter $par = 0.3233866 matches with the previously calculated value in b) .

d) Estimation of standard error and confidence interval

Rcode :



```
> # standard error estimation
> std_error = sqrt(1/mle.optim$hessian)
> std_error
          [,1]
[1,] 0.1446215
>
> # confidence interval calculation
> confi_interval = mle.optim$par + c(-1,1) * qnorm(1 - (0.05/2)) * std_error
Warning message:
In c(-1, 1) * qnorm(1 - (0.05/2)) * std_error :
  Recycling array of length 1 in vector-array arithmetic is deprecated.
  Use c() or as.vector() instead.

> confi_interval # printing CI
[1] 0.03993367 0.60683954
```

The standard error is 0.1446 and 95% confidence internval for ϴ is [0.0399, 0.6068]. Though the calculated value of ϴ in part b) and c)(0.3233) lies within this range, for larger values of n , MLE follows normal distribution and may not fall within this range. So these approximations cannot be considered as good in those cases.