1. **a)** We can calculate the mean squared error of an estimator using the Monte Carlo approach by following the below steps:

**Step (i):** First we will select a random sample of size ‘n’ from the Uniform (0,ϴ) population. For example, this can be done in R using the ‘runif’ function like below:

sample = runif(n, 0, ϴ)

**Step (ii):** Now we will find the estimators of the parameter(ϴ) of the population from the sample as defined in the question, i.e.

1 = max(sample) and 2 = 2\*mean(sample)

(Here the sample\_mean is calculated by using the ‘mean’ function in R).

**Step (iii):** Now we can calculate the squared error of each estimator for a particular sample, since we know the value of ϴ from step i) and the estimator (1 or 2) from step ii).

For 1st estimator, squared error = 1 - ϴ)2

For 2nd estimator, squared error = 2 - ϴ)2

**Step (iv):** Now repeat the steps i) to iii) above ‘N’ number of times for N replications. This can be done using replicate function in R.

For 1st or 2nd estimator, N replications of squared error = replicate (N, method to find squared error)

**Step (v):** Now the mean of the ‘N’ replications of squared error for each estimator will give us the Monte Carlo estimate of mean squared error. This can be calculated by using the ‘mean’ function in R:

mean(abs(N replications of squared error))

**R code:**

# Mean square error of 1st estimator =

>mean(abs(replicate(N,(max(runif(n, 0, ϴ)) - ϴ)^2)))

# Mean square error of 2nd estimator =

>mean(abs(replicate(N,(2\*mean(runif(n, 0, ϴ)) - ϴ)^2)))

**b)** Here, we are asked that estimates should be from same data, so the sample has to be the same each time we calculate squared error for an estimator. We can do this by using ‘set.seed()’ in R.

Let us take n = 1, ϴ = 1. Now the mean squared error can be calculated by the steps shown in 1 a). It is computed using R as below:

**R code:**  (N = 1000)

>set.seed(1)

# Mean squared error of 1st estimator, MSE(1 ) =

> mean(abs(replicate(1000,(max(runif(1, 0, 1)) - 1)^2)))

[1] 0.3333923

> set.seed(1)

# Mean squared error of 2nd estimator, MSE(2 ) =

> mean(abs(replicate(1000,(2\*mean(runif(1, 0, 1)) - 1)^2)))

[1] 0.332336

**c)** We will repeat part b) for remaining combinations of (n, ϴ):

**R code:** (N =1000)

Case ii) n = 1, ϴ = 5

>set.seed(1)

# Mean squared error of 1st estimator, MSE(1 ) =

> mean(abs(replicate(1000,(max(runif(1, 0, 5)) - 5)^2)))

[1] 8.334808

> set.seed(1)

# Mean squared error of 2nd estimator, MSE(2 ) =

> mean(abs(replicate(1000,(2\*mean(runif(1, 0, 5)) - 5)^2)))

[1] 8.308401

MSE(1 )

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | n=1 | n=2 | n=3 | n=5 | n=10 | n=30 |
| ϴ = 1 | 0.3333923 | 0.166913 | 0.1038308 | 0.04538 | 0.01371 | 0.00194 |
| ϴ = 5 | 8.334808 | 4.172824 | 2.595771 | 1.13460406 | 0.342672 | 0.04849 |
| ϴ = 50 | 833.4808 | 417.2824 | 259.5771 | 113.4604 | 34.26724 | 4.8498 |
| ϴ = 100 | 3333.923 | 1669.13 | 1038.308 | 453.8416 | 137.069 | 19.39959 |

MSE(2 )

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | n=1 | n=2 | n=3 | n=5 | n=10 | n=30 |
| ϴ = 1 | 0.332336 | 0.1634447 | 0.1116509 | 0.0707 | 0.0362 | 0.0118 |
| ϴ = 5 | 8.308401 | 4.086117 | 2.791272 | 1.766503 | 0.90419 | 0.29465 |
| ϴ = 50 | 830.8401 | 408.6117 | 279.1272 | 176.6503 | 90.4199 | 29.4658 |
| ϴ = 100 | 3323.36 | 1634.447 | 1116.509 | 706.6014 | 361.6798 | 117.8635 |

**d)** If ‘n’ increases, MSE decreases while ϴ is constant for each estimator. If ‘ϴ’ increases, MSE increases while ‘n’ is constant for each estimator.

It can be seen from the collected data that 2 (MOME) is better in the cases when n <=2 because it gives less MSE for all sets of ‘n’ and ‘ϴ’. For other cases, 1 (MLE) is better.

**Q2.** **a)** Here ,likelihood function can be written as below:

L(ϴ) = = ϴn/

log(L(ϴ)) = n.log(ϴ) – log() = n.log(ϴ) – (ϴ + 1).log() --- (x)

Now putting derivative of (x) w.r.t. ‘ϴ’ equals to zero, we get :

(n/ϴ) = log() 🡪 ϴMLE = n/ log()

**b)** n = 5,

Using expression in part a), ϴMLE = 5/log(5185263.5232) = 5/15.461331 = 0.323387

**c)** Negative of log likelihood function in R can be written as below:

sample=c(21.72,14.65,50.42,28.78,11.23)

neg.loglik.fun <- function(par,dat){

result <- length(dat)\*log(par) - (par + 1)\*log(prod(dat))

return(-result)

}

Note that ‘par’ in the above function will be our unknown parameter ϴ and ‘dat’ is collection of data samples given in the question.

Now for using the ‘optim’ function, we will need start value for ϴ, which we can get by Method of Moment estimator as below:

1st population moment = μ1 = E(X) = = ϴ = ϴ/ (ϴ - 1) for ϴ > 1.

Note that expectation is infinity for ϴ < 1.

1st sample moment = (1/n) = (1/5) = 25.36 , (because n =5 for given sample)

Equating 1st population moment and 1st sample moment, we get ϴ = 1.041 which will be our start value in optim function.

Now ,the optim function in R can be written as below :

> ml.est <- optim(par= 1.041,fn = neg.loglik.fun, method = "BFGS", hessian=T, dat=sample)

> ml.est$par

[1] 0.3233876

> ml.est

$`par`

[1] 0.3233876

$value

[1] 26.10585

$counts

function gradient

22 9

$convergence

[1] 0

$message

NULL

$hessian

[,1]

[1,] 47.81142

Yes ,it matches with our answer in part b).

**d)** Standard error will be square root of inverse of hessian(I) obtained in part c), i.e.

se <- sqrt(1/ml.est$hessian)

[1,] 0.1446219

Now the 95% confidence interval can be calculated as below:

CI = +- Z-score(alpha/2)\*se(),

We have ‘se’ calculated above and we know from part c). Now Z-score(alpha/2) = qnorm(1- (alpha/2)) = qnorm(0.975) = 1.9599

So, CI = (0.3234 - 1.96\*0.1446, 0.3234 + 1.96\*0.1446) = (-28.0182, 28.665)

But we have seen above in part c) that ϴ should be greater than 1, so above CI will not hold good for values of ϴ < 1.