MS6051 Statistical Inference

Assignment

Dara Corr - 22275193

November 23, 2022

Contents

Introduction

This Assignment looks at carrying out simulation studies in `R`. This assignment is split into 5 different questions.

- Question 1 looks at Estimating the bias and efficiency of an estimator of Poisson Data by carrying out a simulation study in `R`.
- Question 2 looks at investigating the Central Limit Theorem for sample means of different distributions using Histograms and Q-Q plots.
- Question 3 is concerned with finding the maximum likelihood estimators for parameters of the Gamma function.
- Question 4 is concerned with evaluating the performance of two approximate confidence intervals for sample means of normal data.
- In Question 5, we produce Confidence Intervals for a distribution using Bootstrapping and then we compare the performance of the bootstrapped Confidence Intervals to Wald type Confidence Intervals.

1 Question 1

Question 1 asks us to carry out a simulation study for a Poisson samples.

(i) In Question 1 (i), we are asked to create a function with parameters λ and sample size n that generates a sample of Poisson Data using `rpois` and calculates

the mean of this sample. The mean of this sample is an Estimator for λ, since for Poisson Distributions $E(X) = \lambda$.

I created this function using the following code:

```
#Question 1 Estimation Bias and Efficiency
#i) Using the function rpois, create a function that generates a random sample of
#Poisson Data and calculates the mean of this sample. inputs are vals of lambda and n (sample size)

set.seed(22275193)

meanpois <- function(lam,n){
  x<- rpois(lambda=lam,n=n)
  output <- mean(x)
  output
}
```

(ii) Using the function `meanpois` we just defined, in part (ii) we generate a sample mean with $\lambda = 1$ and n $= 10$.

```
#(ii) generate one sample mean using lambda = 1 and n = 10
lam = 1

set.seed(22275193)
meanpois(lam,n=10)
```

This gives us one sample mean of `1.1`.

(iii) To find out more about using sample means generated from `meanpois` as an estimator for λ we need to make replicate samples to create more sample means.

I used `R`'s `replicate` function to execute `meanpois` 1000 times to give me a vector of 1000 sample Poisson means. Now I can calulcate the mean, variance and and mean squared error of these 1000 sample means.

```
#(iii) run (ii) 1000 times and find mean and variance of the resulting sample means
#lambda vec is our estimator, contains 1000 sample means

set.seed(22275193)
```

```
lambda_vec <- replicate(1000,meanpois(1,10))
mean(lambda_vec)
var(lambda_vec)

#MSE = E((theta_hat-theta)^2)
#MSE = Var(theta_hat) + (E(theta_hat)- theta)
#= var(lambda_vec) + (mean(lambda_vec) - lambda)^2
MSE <- var(lambda_vec) + (mean(lambda_vec) - lam)^2
MSE
```

Using the fact that

$$\begin{aligned} MSE(\hat{\theta}) &= \text{Var}(\hat{\theta}) + (\text{bias})^2 \\ &= \text{Var}(\hat{\theta}) + (E\hat{\theta} - \theta)^2 \end{aligned}$$

where $\theta = \lambda = 1$ and $\hat{\theta} = \hat{\lambda}$ which is the vector of sample means I just computed.

I was then able to obtain the following results from the code:

$$\text{Mean}(\hat{\lambda}) = 1.0066$$

$$\text{Var}(\hat{\lambda}) = 0.1132096$$

$$MSE(\hat{\lambda}) = 0.1132532$$

(iv) Now I repeat the previous step but for $n = 20, n = 50, n = 200, n = 400$ and $n = 1000$. Here I am interested in seeing how the Bias, MSE and Efficiency of the sample mean data change for a range of sample sizes.

To calculate the Efficencies, I first had to calculate the CRLB for the Poisson Distribution.

$$\text{I found that } l(\lambda) = \sum x_i log(\lambda) - n\lambda - \sum log(x!)$$

$$U(\lambda) = \frac{\sum x_i}{\lambda} - n$$

$$\mathcal{I}(\lambda) = E\left(-\frac{\partial}{\partial\lambda}U(\lambda)\right)$$

$$\mathcal{I}(\lambda) = E\left(\frac{\sum x_i}{\lambda^2}\right) = \left(\frac{\sum E(x_i)}{\lambda^2}\right)$$

$$= \frac{n\lambda}{\lambda^2} = \frac{n}{\lambda}$$

$$\text{CRLB} = \frac{1}{\mathcal{I}(\hat{\theta})} = \frac{1}{n/\lambda}$$

$$\text{CRLB} = \frac{\lambda}{n}$$

Here I constructed empty vectors to hold the values we are interested in, and I populated the vectors by running the code we had we had in part (iii) inside the loop with new code for calculating the Bias and the Efficiencies.

```
#(iv)#repeat last step for n=10,20,50,200,400,1000


lam = 1
MSEvec <- c()
Biasvec <- c()
meansvec <- c()
varsvec <- c()
efficiencyvec <- c()
n_nums <- c(10,20,50,200,400,1000)

set.seed(22275193)
for (i in 1:length(n_nums)) {
  n = n_nums[i]
  lambda_vec <- replicate(1000,meanpois(lam,n))
  meansvec[i] <- mean(lambda_vec)
  varsvec[i] <- var(lambda_vec)
  MSE <- var(lambda_vec) + (mean(lambda_vec) - lam)^2
  MSEvec[i] <- MSE
  Bias <- (mean(lambda_vec) - lam)^2
  Biasvec[i] <- Bias
  efficiencyvec[i] <- (lam/n) * 1/(varsvec[i])
  }
meansvec
Biasvec

```

```
plot(n_nums,Biasvec) #bias vs sample size
plot(n_nums,MSEvec) #MSE vs sample size
plot(n_nums,efficiencyvec) #efficiency vs sample size n
```

and then I obtained plots for Bias vs Sample Size, MSE vs Sample Size and Efficiency vs Sample Size $\lambda = 1$

(v) I repreated the method used in (iv) to obtain plots for $\lambda = 0.5$ and $\lambda = 4$

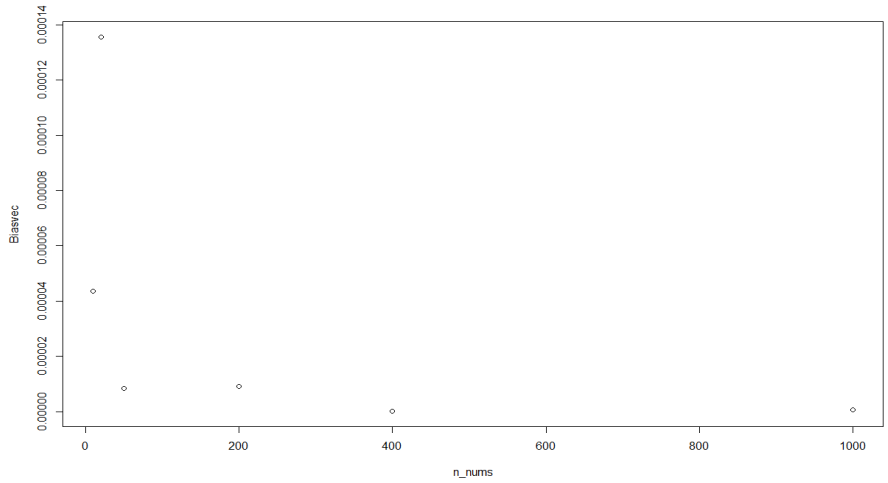


Figure 1: Bias vs Sample Size for $\lambda = 1$

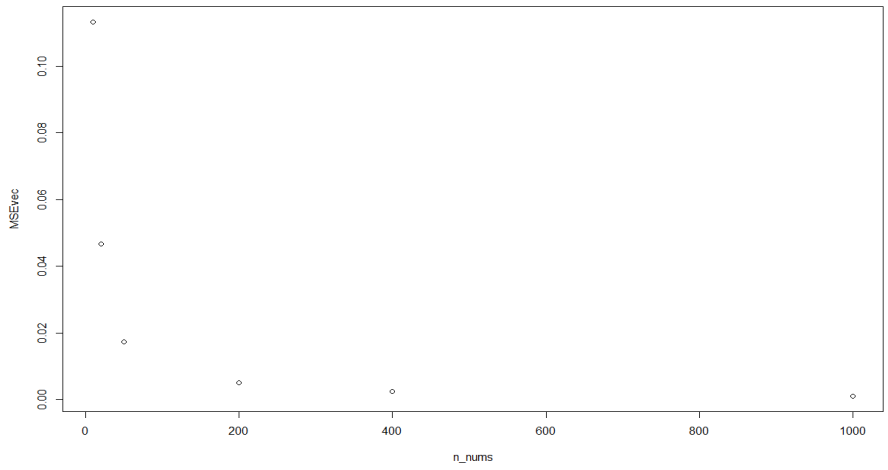


Figure 2: MSE vs Sample Size for $\lambda = 1$

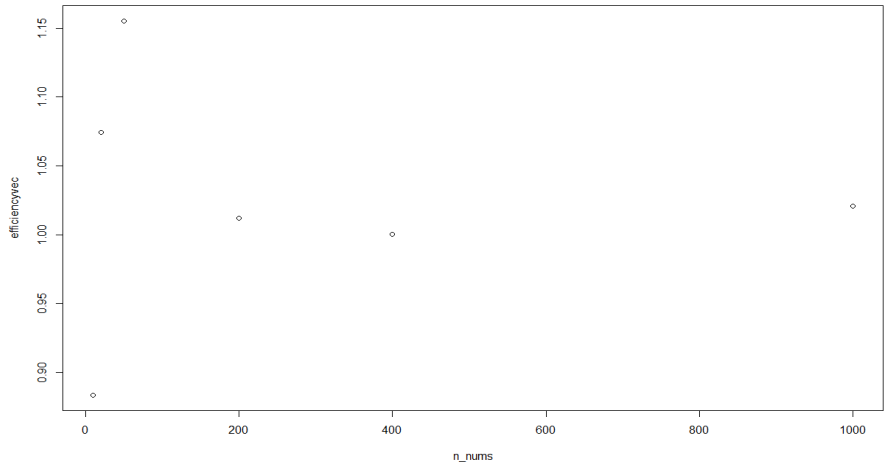


Figure 3: Efficiency vs Sample Size for $\lambda = 1$

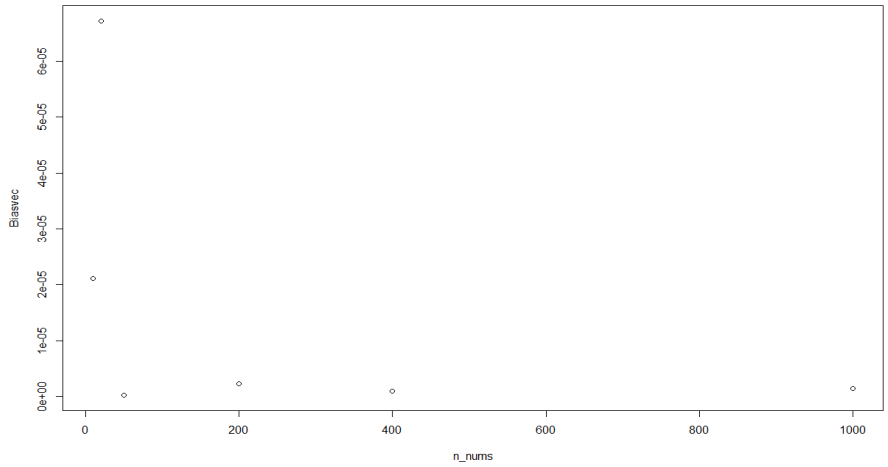


Figure 4: Bias vs Sample Size for $\lambda = 0.5$

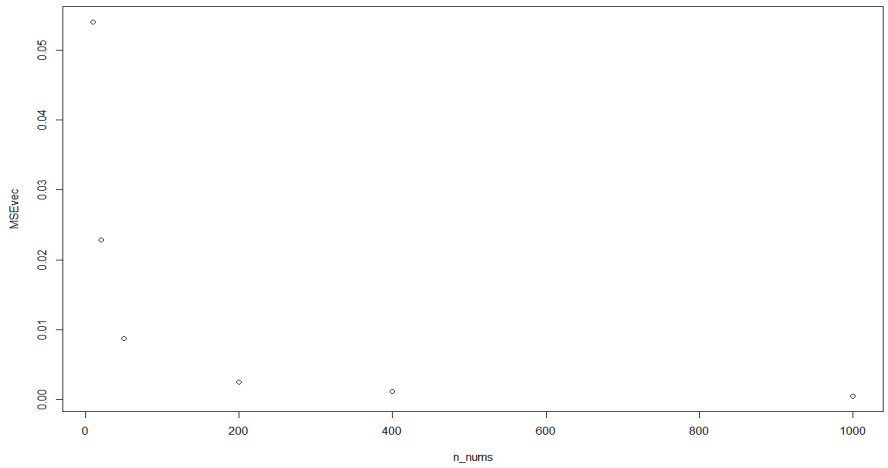


Figure 5: MSE vs Sample Size for $\lambda = 0.5$

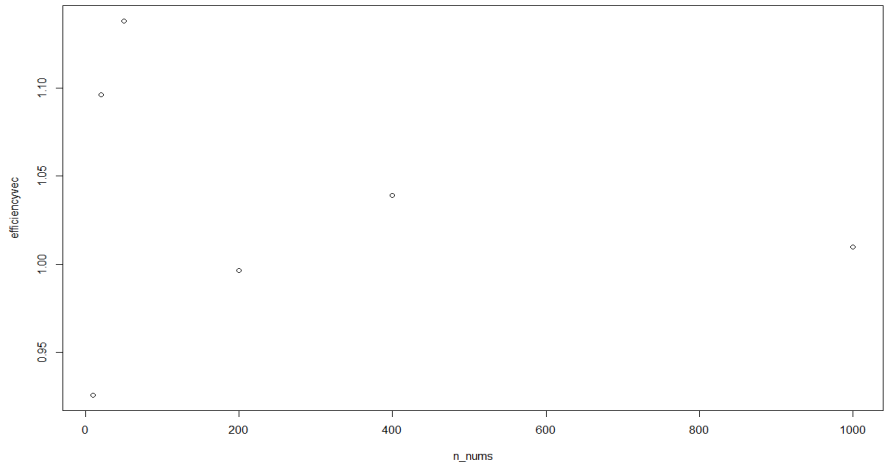


Figure 6: Efficiency vs Sample Size for $\lambda = 0.5$

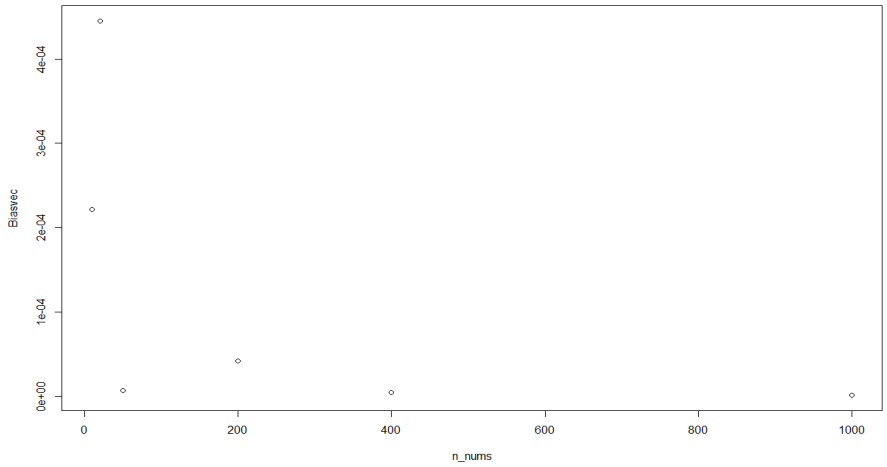


Figure 7: Bias vs Sample Size for $\lambda = 4$

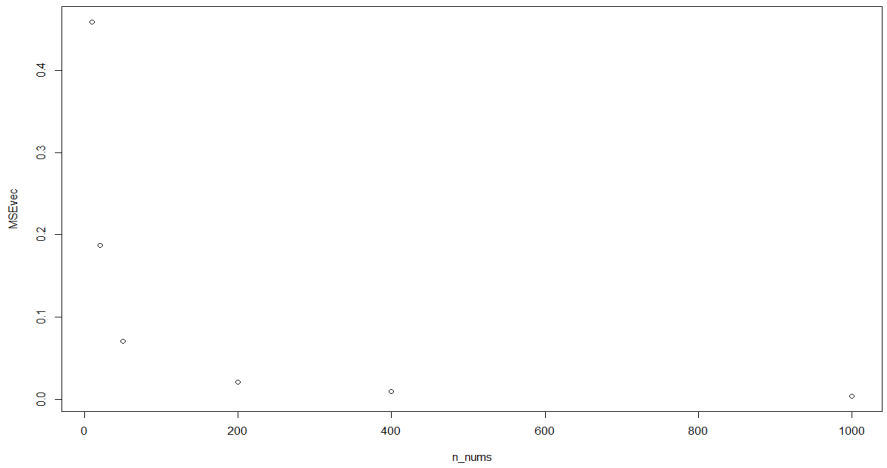


Figure 8: MSE vs Sample Size for $\lambda = 4$

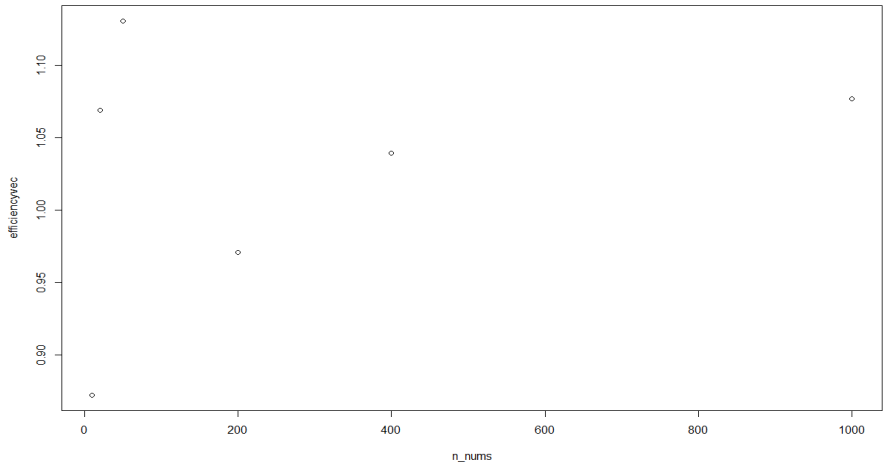


Figure 9: Efficiency vs Sample Size for $\lambda = 4$

Conclusions

- We see that the bias tends towards 0 as n is increased. The reason it does not equal exactly zero is because we are looking at the mean of 1000 sample means and there is going to be some variance in the sample means data and this variance decreases as n is increased. We can thus say that the sample mean of Poisson data is an unbiased (also asymptotically unbiased) estimator of the parameter λ for a Poisson Distribtion. This is also easy to prove analytically:

$$\begin{aligned} E(\overline{X}) &= E\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) \\ &= \frac{1}{n}E\left(\sum_{i=1}^{n} X_i\right) \\ &= \frac{1}{n}\left(\sum_{i=1}^{n} E(X_i)\right) \\ &= \frac{1}{n}\left(\sum_{i=1}^{n} \lambda_i\right) \\ &= \frac{1}{n}\cdot n\lambda \\ &= \lambda \end{aligned}$$

- We can also conclude that the sample mean of Poisson Data is a consistent estimator of λ since we can see from the plots of MSE vs n that, as $n \to +\infty, \text{MSE} \to 0$.

- We can see from the plots that the Efficiency of the sample means oscillates around the value of 1 and tends towards the value of 1 as n increases. Since we have established that the sample mean for Poisson Data is an unbiased estimator, this means its maximum variance is the CRLB, which means the efficiency of the estimator is some value between 0 and 1 given by $\text{eff}(\hat{\lambda}) = \frac{\text{CRLB}}{\text{Var}(\hat{\lambda})} \in (0, 1)$. Since this value cannot exceed 1, we deduce that this must be an estimator that achieves the CRLB, and is a Minimum Variance Unbiased Estimator, giving an Efficiency value of 1.

  The reason that the efficiency values oscillate about 1 in the plots is because the variance we compute is the sample variance of 1000 samples and there some uncertainty in this value especially for smaller values of n.

2 Question 2

The Central Limit Theorem States that for any iid sample $X_1, ...X_n$,

$$\overline{X} \to N\left(\mu, \frac{\sigma^2}{n}\right)$$

i.e. The sample means of any iid distribution are approximately normally distributed.

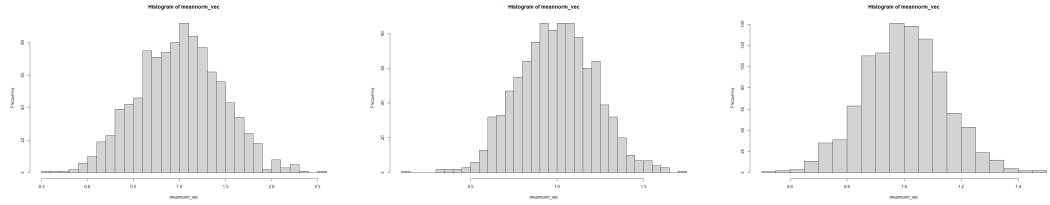
In this Question, we are tasked with demonstrating that the Central Limit Theorem holds using histograms and Q-Q plots for sample means of different distributions in R.

(i) I used R's `norm(n,mu,sigma)` function in my `meannorm(n,mu,sigma)` function to produce a sample mean of size n from a normal distrubution with mean $\mu = 1$ and standard deviation $\sigma = 1$.

```
##Question 2
#i)
#Normal distribution with mu = 1 and sigma = 1


#use rnorm() function
meannorm <- function(n,mu,sigma){
  x<- rnorm(n=n,mean = mu, sd = sigma)
  output <- mean(x)
  output
}
set.seed(22275193)
meannorm(5,1,1)
```

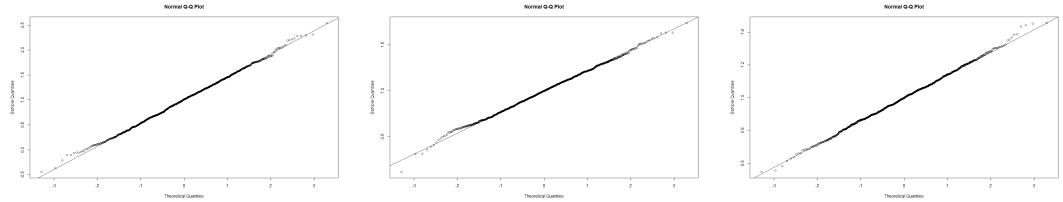
Like in Question 1, I replicate this 1000 times to produce 1000 sample means. Then I produced a Q-Q plot and a Histogram of the sample means to get an idea of how the sample means of the normal data are distributed.

```
set.seed(22275193)
meannorm_vec <- replicate(1000,meannorm(n=5,1,1)) #change n =5,
    20 and 50
mean(meannorm_vec)
var(meannorm_vec)

hist(meannorm_vec,breaks = 30) #showing sample of Normal data is
    normally distributed
qqnorm(meannorm_vec)#using hist and qqplots
qqline(meannorm_vec)
```



$n = 5$ $n = 20$ $n = 50$

Figure 10: Histograms of 1000 sample means of normal data for $n = 5, n = 20, n = 50$



$n = 5$ $n = 20$ $n = 50$

Figure 11: Q-Q plots of 1000 sample means of normal data for $n = 5, n = 20, n = 50$

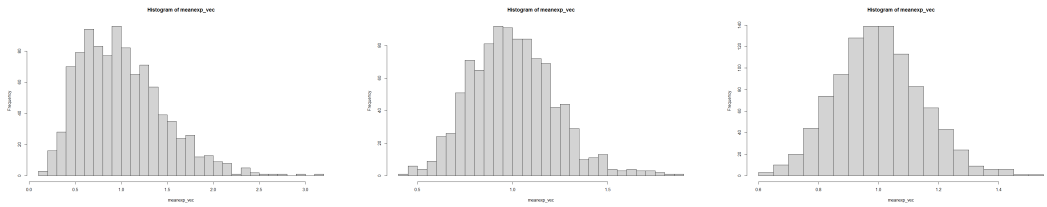
We can see from the above plots that the sample means of a normal distribution are always distributed normally for any n.

(ii) We will now produce histogram and Q-Q plots of the sample means of exponentially distributed data with $\lambda = 1$ and Bernouilli data with $p = 0.5$ and $p = 0.05$ for $n = 5, n = 20$ and $n = 50$

I created a function `meanexp(n,lambda)` in `R` which calculates the sample means of a randomly generated exponential sample with parameters n and λ. I replicated this process 1000 times to create 1000 sample means for exponential data.

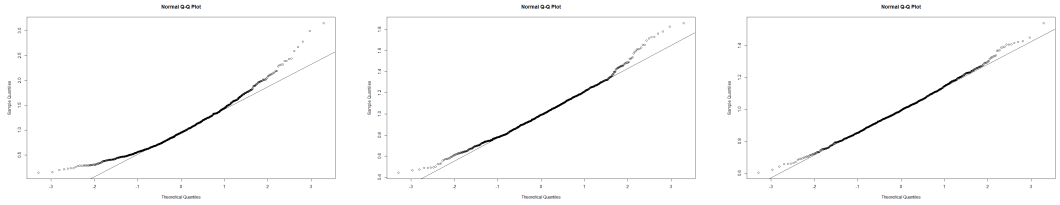
```
#rexp(# n observations, rate)
meanexp <- function(n_obvs,lambda){
  x <- rexp(n=n_obvs,rate = lambda)
  output <- mean(x)
  output
}

set.seed(22275193)
meanexp_vec <- replicate(1000,meanexp(n=50,lambda = 1)) #change n
    =5, 20 and 50

```

```
hist(meanexp_vec, breaks = 30)#using hist and qqplots to show
    sample mean of exp is normally distributed
qqnorm(meanexp_vec)
qqline(meanexp_vec)
```



$n = 5$ $n = 20$ $n = 50$

Figure 12: Histograms of 1000 sample means of exponential data with $\lambda = 1$ for $n = 5, n = 20, n = 50$



$n = 5$ $n = 20$ $n = 50$

Figure 13: Q-Q plots of 1000 sample means of exponential data with $\lambda = 1$ for $n = 5, n = 20, n = 50$

We see that for $n = 5$ and $n = 20$, the sample means of the Exponential Data is obviously right skewed. We notice that the data becomes less skewed as n increases and for $n = 50$, the distribution appears to be normally distributed.

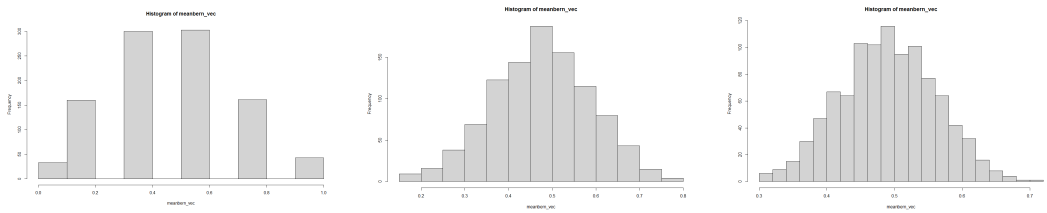
This is what we would expect to see. As a general rule of thumb, if $n > 30$, we should expect the distribution of sample means to be (approximately) normally distributed with distribution $\overline{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right)$. However with certain distributions, we may need greater values of n to achieve a normal distribution for $\overline{X}$.

Like before, we create a function `meanbernouilli(n,p)` with parameters n and p which creates randomly generated Bernouilli sample data with sample size n using R's `rbinom` function with `size = 1`. This function then finds the

sample mean for this sample. As before, I made 1000 replicates of this to create plots for $p = 0.5$ and $p = 0.05$ for sample sizes $n = 5$, $n = 20$ and $n = 50$.

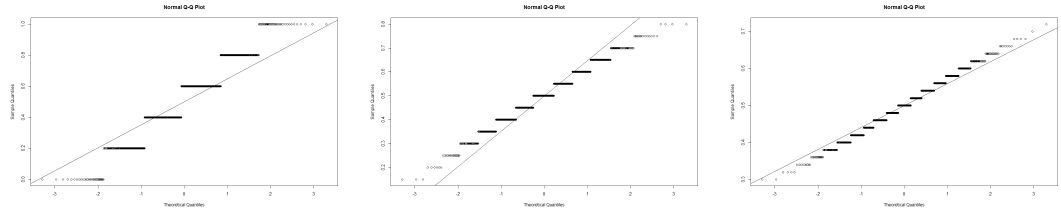
```
##repeat for Bernouilli dist
#p =0.5 and p = 0.05

meanbernouilli <- function(n_obvs,p){
  x <- rbinom(n=n_obvs,size = 1, prob = p)
  output <- mean(x)
  output
}

#n= 5, 20, 50
set.seed(22275193)
meanbern_vec <- replicate(1000,meanbernouilli(n = 20, p =0.05))
mean(meanbern_vec)
var(meanbern_vec)

meanbern_vec

hist(meanbern_vec, breaks = 20)#omitted breaks for n = 5, p =0.5
    & for p = 0.05
qqnorm(meanbern_vec)
qqline(meanbern_vec)
```

Plots for $\mathbf{p = 0.5}$:



$n = 5$ $n = 20$ $n = 50$

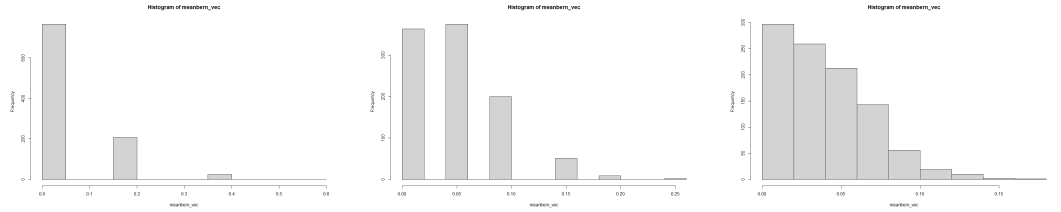
Figure 14: Histograms of 1000 sample means of Bernouilli data with $p = 0.5$ for $n = 5$, $n = 20$ and $n = 50$



$n = 5$ $n = 20$ $n = 50$

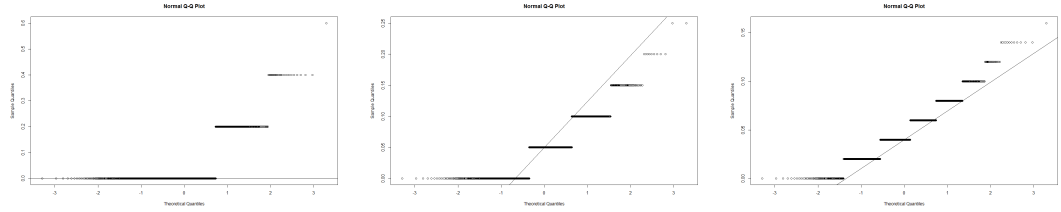
Figure 15: Q-Q plots of 1000 sample means of Bernouilli data with $p = 0.5$ for $n = 5$, $n = 20$ and $n = 50$

Plots for $\mathbf{p = 0.05}$:



$n = 5$ $n = 20$ $n = 50$

Figure 16: Histograms of 1000 sample means of Bernouilli data with $p = 0.05$ for $n = 5$, $n = 20$ and $n = 50$

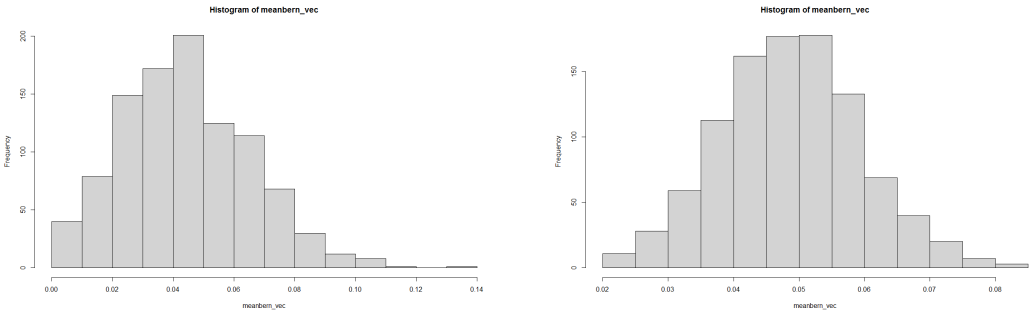


$n = 5$ $n = 20$ $n = 50$

Figure 17: Q-Q plots of 1000 sample means of Bernouilli data with $p = 0.05$ for $n = 5$, $n = 20$ and $n = 50$

We can see from figure 14 and 15 above, that for $p = 0.5$, the sample means appear to be distributed normally for $n = 5$, $n = 20$ and $n = 50$.

In figure 16 and 17, we see that the sample means do not appear to be distributed normally and are highly right skewed, which is unsurprising as $E(X) = p$ for Bernouilli Distributions, and here 5/100 events will be a success. The skewness of the distribution appears to be decreasing as n is increased and the distribution appears to be moving towards a Normal Distribution. To confirm this I created 2 more plots for $n = 100$ and $n = 400$.



$n = 100$ $n = 400$

Figure 18: Bernouilli Sample Mean plots with $p = 0.05$ for $n = 100$ and $n = 400$

These plots show that even for plots with high skewness, that for large enough values of n, the distribution of their sample means will approach a normal distribution.

From my analysis in this question, I have seen that in general, the more skewness in the original distribution, the larger the value of n is required to get the sample mean data distributed normally. If there is skewness in the original distribution then increasing the sample size n will make the distribution of sample means approach a (approximately) Normal Distribution and a larger sample size may be needed. For distributions with little to no skew, smaller sample sizes n will result in sample mean data which is Normally Distributed.

As a rule of thumb, $n \approx 30$ or greater is needed to achieve a Normal Distribution of Sample Means as we saw with the Exponential Distribution. We also observed that the sample mean of Normal Data is always Normally Distributed.

3 Question 3

In Question 3 of the assignment, we look at finding Maximum Likelihood Estimators for parameters of a randomly generated Gamma Distribution.

(i) First, I generated one sample of gamma data of size $n = 100$ with $\lambda = 1$ and $\alpha = 3$.

```
#Q3
#Generate one sample of Gamma data
#of size n = 100, lambda = 1 and alpha = 3
alpha = 3
lambda = 1
n = 100
set.seed(22275193)
gammasample <- rgamma(n,shape =alpha ,rate = lambda)
x <- gammasample #call the sample data x for simplicity and
    readability
```

(ii) Then by hand, I found the Maximum Likelihood Estimator for λ analytically assuming α is known.

We know the pdf for Gamma Distributions is $f(x|\lambda,\alpha) = \frac{\lambda^{\alpha}\cdot x^{\alpha-1}\cdot e^{-\lambda x}}{\Gamma(\alpha)}$

$$
\begin{aligned}
l(\lambda,\alpha) &= \sum_{i=1}^{n} log(f(x|\lambda,\alpha)) \qquad (1)\\
&= \sum_{i=1}^{n} \left[log(\lambda^{\alpha}) + log(x^{\alpha-1}) - \lambda x - log(\Gamma(\alpha))\right]\\
&= n\alpha \cdot log(\lambda) + (\alpha-1)\sum_{i=1}^{n} log(x_i) - \lambda x_i - n\cdot log(\Gamma(\alpha))
\end{aligned}
$$

To find MLE $\hat{\lambda}$, set $U(\lambda) = 0$

$$
\begin{aligned}
U(\lambda) &= \frac{\partial l(\lambda,\alpha)}{\partial \lambda}\\
&= \frac{n\alpha}{\lambda} - \sum_{i=1}^{n} x_i = 0
\end{aligned}
$$

$$\frac{n\alpha}{\lambda} = \sum_{i=1}^{n} x_i$$
$$\hat{\lambda} = \frac{n\alpha}{\sum_{i=1}^{n} x_i} = \frac{\alpha}{\overline{x}}$$

```
#ii) find MLE for gamma parameter lambda by hand
#lambda_hat = (n*alpha)/sum(x_i)
lambda_hat <- (n*alpha)/sum(x)
lambda_hat
```

This gives me a $\hat{\lambda}$ value of `0.98031`, by taking $\alpha = 3$.

(iii) Now I plug this value of $\hat{\lambda}$ into the Likelihood function for the Gamma function (1) in `R` so that it is only a function of α now.

By plotting the likelihood function for a range of α values in `R`, I should now be able to evaluate α to one decimal place and then I should be able to evaluate the corresponding λ value by plugging the value of α I found into the Likelihood function.

```
#iii) plug lambda_hat into likelihood function and estimate alpha
    _hat to
#one decimal place using plot() function

gammalikelihood <- function(alpha){
  #function computes log likelihood for our gamma sample data
  loglike <- n*alpha*log(lambda_hat) + (alpha - 1)*sum(log(x)) -
      lambda_hat*sum(x) - n*log(gamma(alpha))
  loglike
  }
alpha_vals = seq(from = 0.1, to = 10, by = 0.0005) #create finer
    mesh to pinpoint alpha

likelihoodplot <- gammalikelihood(alpha_vals)

max(likelihoodplot)
I <- match(max(likelihoodplot),likelihoodplot)

alpha_vals[I]

```

```
plot(alpha_vals,likelihoodplot)
library(zoom)
zm()
#alpha_hat = 2.8505 from plot
#asked to give alpha_hat to 1dp ->

alpha_hat <- 2.9
```

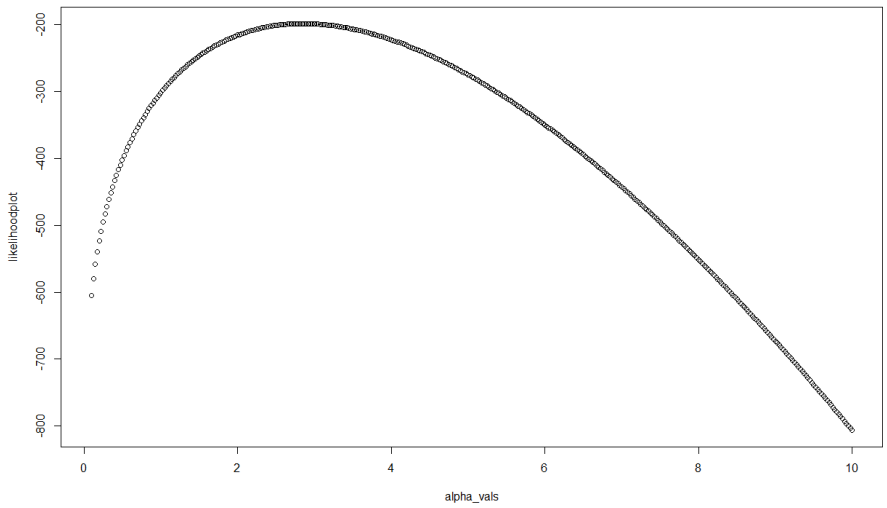


Figure 19: Plotting the likelihood function for a gamma distribution sample with λ = `0.98031` for a range of α values

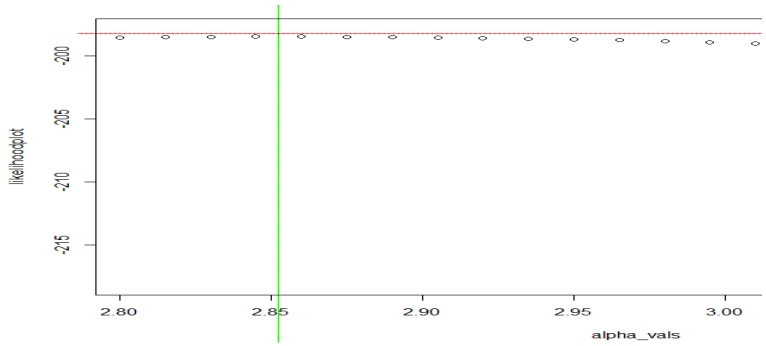


Figure 20: Zoom in of likelihood plot with Maximum Likelihood identified at around the point $\alpha = 2.85$

By looking at the plot, I deduced that using this method, $\hat{\alpha}$ has a value of approximately 2.85. But since we want to evaluate α to 1 decimal place, I made a finer grid of α values and evaluated the value of α needed to get

maximum value of the plot in R and obtained a value of 2.8505 for α which I rounded up to 2.9.

Using this value of $\hat{\alpha}$, I then evaluated its corresponding $\hat{\lambda}$ value.

```
#what is the associated lambda_hat value for this alpha_hat value
    :

#lambda_hat_eval = n*alpha/sum(x_i)

lambda_hat_2 <- (n*alpha_hat)/sum(x) #same as 2.85/X_bar =  2.85/
    mean(x)
lambda_hat_2
#lambda_hat = 0.9476307 for alpha_hat = 2.9
```

Which I found to be $0.9476307 \approx 0.948$

(iv) I used a Newton-Raphson type Estimation Procedure in R using its nlm function to find the parameters that minimize the minus likelihood i.e. maximise the likelihood of the likelihood function.

To do this I rewrote the previous Likelihood function I had defined but this time it has the parameters (λ,α) stored in one array called $\hat{\theta}$, which are the parameters we want the nlm function to estimate. The other change I made is that it now returns -loglike as its output instead of the regular loglikelihood value loglike that I had before.

So now our likelihood function has parameters $\hat{\theta}$ and x. Now we can use the nlm function to minimize the negative likelihood and estimate the parameters α and λ.

```
gammalikelihood2 <- function(theta_hat,x){
  #function computes log likelihood for our gamma sample data
  alpha <- theta_hat[1]
  lambda <- sqrt((theta_hat[2])^2) #define lambda > 0
  n <- length(x)
  loglike <- n*alpha*log(lambda) + (alpha - 1)*sum(log(x)) -
      lambda*sum(x) - n*log(gamma(alpha))
  return(-loglike)#minus log likelihood output
}

#calculating mle of gamma fn using nlm() function with data x and
    kicking off the
```

```
#newton method procedure at lambda_hat=1 and alpha_hat = 3
mle <- nlm(gammalikelihood2,p = c(3,1), x=x)
```

Which gave the following output:

```
> mle <- nlm(gammalikelihood2,p = c(3,1), x=x)
> mle
$minimum
[1] 196.461

$estimate
[1] 2.265237 0.740209

$gradient
[1]  1.342519e-06 -5.400125e-06

$code
[1] 1

$iterations
[1] 7
```

Figure 21: output from `nlm` function in `R`

The `nlm` procedure gave us values of $\alpha = 2.265237$ and $\lambda = 0.740209$. This method estimates these values to much more precision than the plotting method did.

(v) If we include the parameter `hessian = TRUE` when calling the `nlm` function, we can get `R` to produce the Hessian Matrix.

```
mle2 <- nlm(gammalikelihood2,p = c(3,1), x=x,hessian = TRUE)

mle2
```

```
> mle2 <- nlm(gammalikelihood2,p = c(3,1), x=x,hessian = TRUE)
> mle2
$minimum
[1] 196.461

$estimate
[1] 2.265237 0.740209

$gradient
[1]  1.342519e-06 -5.400125e-06

$hessian
          [,1]      [,2]
[1,]   55.26709 -135.0878
[2,] -135.08785  413.3210

$code
[1] 1

$iterations
[1] 7
```

Figure 22: output from `nlm` function in `R` with Hessian matrix in the output

The Hessian matrix **H** here, is of the following form:

$$\mathbf{H} = -\begin{bmatrix} \frac{\partial^2 l}{\partial {\theta_1}^2} & \frac{\partial^2 l}{\partial \theta_1 \partial \theta_2} \\ \frac{\partial^2 l}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 l}{\partial {\theta_2}^2} \end{bmatrix} = -\begin{bmatrix} \frac{\partial l^2}{\partial^2 \alpha} & \frac{\partial l^2}{\partial \alpha \partial \lambda} \\ \frac{\partial l^2}{\partial \lambda \partial \alpha} & \frac{\partial l^2}{\partial^2 \lambda} \end{bmatrix}$$

Where l is the loglikelihood. The minus sign appears because we used the `nlm` function on the negative of the likelihood function $-l(\alpha, \lambda)$.

Recalling the definition for Observed Information $I(\theta)$:

$$I(\theta) = -\begin{bmatrix} \frac{\partial^2 l(\theta)}{\partial {\theta_1}^2} & \frac{\partial^2 l(\theta)}{\partial \theta_1 \partial \theta_2} \\ \frac{\partial^2 l(\theta)}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 l(\theta)}{\partial {\theta_2}^2} \end{bmatrix}$$

We notice the Hessian **H** we just obtained is the same as the Observed Information $I(\hat{\theta})$. We can now calculate Confidence Intervals for $\hat{\alpha}$ and $\hat{\lambda}$ of the form

$$\hat{\alpha} \pm z_{\alpha/2}[I^{-1}(\hat{\theta})]_{11}$$

$$\hat{\lambda} \pm z_{\alpha/2}[I^{-1}(\hat{\theta})]_{22}$$

I calculated these confidence intervals in `R` as follows

```
obsv_info <- mle2$hessian
obsv_info

library(matlib)

var_alpha_hat <- Inverse(obsv_info)[1]
var_lambda_hat <- Inverse(obsv_info)[4]

alpha_hat_ci <- c(mle2$estimate[1] -1.96*var_alpha_hat , mle2$
    estimate[1] +1.96*var_alpha_hat )
alpha_hat_ci #95% CI for lambda_hat, alpha = 0.05

lambda_hat_ci <- c(mle2$estimate[2] -1.96*var_lambda_hat , mle2$
    estimate[2] +1.96*var_lambda_hat )
lambda_hat_ci #95% CI for lambda_hat, alpha = 0.05

```

```
alpha_hat_ci <- c(mle2$estimate[1] -2.576*var_alpha_hat , mle2$
    estimate[1] +2.576*var_alpha_hat )
alpha_hat_ci #99% CI for alpha_hat, alpha = 0.01

lambda_hat_ci <- c(mle2$estimate[2] -2.576*var_lambda_hat , mle2$
    estimate[2] +2.576*var_lambda_hat )
lambda_hat_ci #99% CI for lambda_hat, alpha = 0.01
```

I obtained the following Confidence Intervals:

95% Confidence Intervals for $\hat{\alpha}$ and $\hat{\lambda}$:

$$\hat{\alpha} = (2.088908, 2.441565)$$
$$\hat{\lambda} = (0.7166314, 0.7637867)$$

99% Confidence Intervals for $\hat{\alpha}$ and $\hat{\lambda}$:

$$\hat{\alpha} = (2.033491, 2.496982)$$
$$\hat{\lambda} = (0.7092212, 0.7711969)$$

Conclusions

Here we looked at different ways of estimating an estimator which cannot be easily solved analytically. The MLE for the gamma parameter α is hard to compute since $\frac{\partial l}{\partial \alpha}$ results in a non-linear equation with no closed form solution (due to the presence of the digamma function).

We then plotted the likelihood function over a range of α values to estimate $\hat{\alpha}$ and $\hat{\lambda}$. This method was not very precise, so to get a more precise estimate for $\hat{\alpha}$ and $\hat{\lambda}$, I used a Newton-Raphson procedure to minimize the -loglikelihood and from this find the MLE estimators for $\hat{\alpha}$ and $\hat{\lambda}$. Using the Hessian Matrix, I was then able to find confidence Intervals for $\hat{\alpha}$ and $\hat{\lambda}$.

Since the result from the `nlm` function is only an estimate for the Maximum Likelihood estimators $\hat{\alpha}$ and $\hat{\lambda}$ and not an exact value, by getting the Confidence Interval for these estimators we know there is a 95% chance that the true values of $\hat{\alpha}$ and $\hat{\lambda}$ lie within their associated 95% confidence intervals.

4 Question 4

In this part of the assignment, I will be assessing the performance of 2 different Confidence Intervals that calculate 95% Confidence Intervals for the mean μ from a Normal Distribution.

$$\bar{x} \pm z_{\alpha/2}\frac{s}{\sqrt{n}} \tag{2}$$

$$\bar{x} \pm t_{n-1,\alpha/2}\frac{s}{\sqrt{n}} \tag{3}$$

(i) I first created a function called `normCI_z` which generates data from a normal distribution and calculates a 95% confidence interval for μ from this sample using equation (2) above (note $z_{0.05/2} = 1.96$).

```
normCI_z <- function(n,mu,sigma){
  x<- rnorm(n=n,mean = mu, sd = sigma)
  xbar <- mean(x)
  s_squared <- var(x)
  output <- c(xbar - 1.96*sqrt(s_squared/n), xbar + 1.96*sqrt(s_
      squared/n))
}
```

(ii) Then I created another function called `normCI_t` which generates normal data and calculates a a 95% confidence interval using equation (3).

```
normCI_t<- function(n,mu,sigma){
  x<- rnorm(n=n,mean = mu, sd = sigma)
  xbar <- mean(x)
  s_squared <- var(x)
  t <- qt(p = 0.025, df = n-1) #gives negative t-vals
  output <- c(xbar + t*sqrt(s_squared/n), xbar - t*sqrt(s_squared
      /n))
}
```

(iii) I would expect the Confidence interval (3) to perform better for smaller sample sizes as the t-distribution is based off of the sample variance and sample size n which are known. This is an exact confidence interval when n and s are known.

The z-scores come from the standard distribution which assumes that the population standard deviation σ is known. Since we are working with the sample size s here and not the population standard deviation σ, I would expect

the confidence interval (2) not to perform as well as confidence interval (3) for small sample sizes n.

I would expect that for large n, the two CIs should be very similar as the t distribution becomes more similar to a normal distribution as n increases.

(iv) I performed a simulation to check the performance of these two confidence intervals by evaluating each function 1000 times and seeing how many times out of the 1000 did the confidence interval contain the true value of the mean.

```
n=10
TRUEmu1count = 0
TRUEmu2count = 0

set.seed(22275193)
for (i in 1:1000){

  CI1 <- normCI_z(n,1,1)
  if ((CI1[1] < 1) & (CI1[2] > 1)){
   TRUEmu1count = TRUEmu1count +1
  }

  CI2 <- normCI_t(n,1,1)
  if ((CI2[1] < 1) & (CI2[2] > 1)){
    TRUEmu2count = TRUEmu2count +1
  }

}

TRUEmu1count/10
TRUEmu2count/10
```

I found that in the 1000 replicates, the first 95% confidence interval contained the mean $\mu = 1$, in the interval 925 out of the 1000 iteratations corresponding to a performance of 92.5%. This is less than the 95% value which one would expect it to achieve.

The second confidence interval contained $\mu = 1$ in the interval 947 out of 1000 times, corresponding to 94.7%, which is very close to 95%, as expected.

Thus as predicted, the confidence interval for the normal sample means using t-values performs better than the confidence interval using z-values for small values of n.

(v) I conducted the performance test of these two confidence intervals again but this time I used $n = 500$, to investigate the performance of these two confidence intervals for large sample sizes.

```
n=500
TRUEmu1count = 0
TRUEmu2count = 0

set.seed(22275193)
for (i in 1:1000){

  CI1 <- normCI_z(n,1,1)
  if ((CI1[1] < 1) & (CI1[2] > 1)){
    TRUEmu1count = TRUEmu1count +1
  }

  CI2 <- normCI_t(n,1,1)
  if ((CI2[1] < 1) & (CI2[2] > 1)){
    TRUEmu2count = TRUEmu2count +1
  }

}

TRUEmu1count/10
TRUEmu2count/10
```

I found that the first confidence interval included $\mu = 1$, 954 times and the second confidence interval included $\mu = 1$, 952 times. This means they included $\mu = 1$, 95.4% of the time and 95.2% of the time respectively. Thus we can conclude that for large n, these two confidence intervals are equivalent and both perform the same as the t-distribution becomes distributed the same as a normal (z) distribution for large values of n. Both CIs perform well here and are both approximately equal to 95%.

The difference of 2 counts is insignificant here. Repeating the procedure will show that neither CI performs significantly better than the other for large n.

5 Question 5

In Question 5, I looked at Bootstrapping which is a procedure involving resampling a distribution with replacement. I used a bootstrapping procedure to create a confidence interval for an exponentially distributed sample and compared the performance of this confidence interval I computed using a bootstrapping technique to a Wald confidence interval.

(i) Bootstrapping is a resampling procedure that is used to estimate statistics on a population by sampling a dataset with replacement of the sampled data.

The Process for building a sample from using bootstrapping involves:

1. Choosing the sample size n of the bootstrapped sample
2. While the size of the bootstrapped sample data is less than n
   - Randomly select an observation from the dataset that we are sampling (any observation can be selected more than once)
   - Add the randomly selected observation to the bootstrapped sample data.

(ii) I generated a sample of exponential data with $n = 100$ and $\lambda = 1$. I then bootstrapped this data 1000 times and calculated the median in each bootstrapped sample. I then constructed a 95% confidence interval for the median of exponential data using the 0.025 and 0.9725 quartiles of the bootstrapped medians as the lower and upper bounds of the interval.

```
n = 100
lam = 1

set.seed(22275193)
expsample <- rexp(n=n, rate = lam)
expsample

#the statistic we are interested in, is the median:

##Getting 1000 Bootstrapped samples and finding the median of
    each bootstrapped
#sample and storing the medians in a vector (result_vec)

B = 1000
result_vec <- vector(length = B)

```

```
set.seed(22275193)
for (b in 1:B){

  boot_sample <- sample(expsample, size = length(expsample),
      replace = TRUE)

  med <- median(boot_sample)

  result_vec[b] <- med
}

hist(result_vec, breaks = 30)
median(expsample)

#getting 95% Confidence Interval
lower_bound <- quantile(result_vec, probs = 0.025)
upper_bound <- quantile(result_vec, probs = 0.975)
boot_CI <- c(lower_bound,upper_bound)
```

This gives me a 95% Confidence Interval for the Median of the Exponential Data:

$$CI_{\text{median boot}} = (0.6730023, 1.1003294)$$

(iii) Now I want to test the performance of this confidence interval using a similar method to what I did in Q4(ii). I ran this bootstrapping procedure in part 5(ii) 1000 times and counted the number of times out of 1000, that the true median of exponential data with $\lambda = 1$ is contained inside the interval.

To do this I first had to calculate the True Median for Exponential Data. This is the value of m such that $\Pr(X > m) = 0.5$.

$$\text{note: pdf} = \lambda e^{-\lambda x} \text{ for Exponential distribution}$$

$$
\begin{aligned}
0.5 &= \Pr(X > m) \\
0.5 &= \int_m^{+\infty} f(x)dx \\
0.5 &= \int_m^{+\infty} \lambda e^{-\lambda x} dx \\
0.5 &= [-e^{-\lambda x}]\Big|_{x=0}^{x=+\infty} \\
0.5 &= 0 + e^{-\lambda m} \\
0.5 &= e^{-\lambda m} \\
1 &= 2 \cdot e^{-\lambda m} \\
0 &= log(2) - \lambda m \\
m &= \frac{log(2)}{\lambda}
\end{aligned}
$$

And then I implemented this in R.

```
1
2  n = 100 #change to n = 10 and n = 50 for part (v)
3  lam = 1
4  B = 1000
5
6  set.seed(22275193)
7  expsample <- rexp(n=n, rate = lam)
8  expsample
9
10 #Need to calculate True median of original sample:
11 TRUEmed <- log(2)/lam
12 TRUEmed
13
14 CI_count <- 0
15
16 set.seed(22275193)
17 for (i in 1:1000){
18
19
20   ##Getting 1000 Bootstrapped samples and finding the median of
       each bootstrapped
21   #sample and storing the medians in a vector (result_vec)
22   result_vec <- vector(length = B)
23
24     for (b in 1:B){
25
```

```
      boot_sample <- sample(expsample, size = length(expsample),
          replace = TRUE)

      med <- median(boot_sample)

      result_vec[b] <- med
    }


  #getting 95% Confidence Interval
  lower_bound <- quantile(result_vec, probs = 0.025)
  upper_bound <- quantile(result_vec, probs = 0.975)
  boot_CI <- c(lower_bound,upper_bound)


  if ((boot_CI[1] < TRUEmed) & (boot_CI[2] > TRUEmed)){
    CI_count = CI_count + 1
  }
}
```

This resulted in an output of 981 times the true median was found in this confidence interval, corresponding to 98.1%.

(iv) I computed the Wald type confidence interval for the median m so I can compare its performance to the confidence interval produced by bootstrapping.

first I computed the Wald CI for $\hat{\lambda}$:

$$\hat{\lambda} \pm z_{\alpha/2}\sqrt{\frac{1}{I(\hat{\lambda})}}$$

To construct this Confidence Interval, we first need to find $I(\lambda)$ for the Exponential Distribution

We know that the pdf for exponential distribution is given by: $f(x|\lambda) = \lambda e^{-\lambda x}$

Then:

$$\begin{aligned} l(\lambda) &= \sum_{i=1}^{n} log(f) \\ &= n \cdot log(\lambda) - \lambda \sum_{i=1}^{n} x_i \end{aligned}$$

$$\begin{aligned} U(\lambda) &= \frac{\partial l}{\partial \lambda} \\ U(\lambda) &= \frac{n}{\lambda} - \sum_{i=1}^{n} x_i \\ \hat{\lambda} &= \frac{n}{\sum_{i=1}^{n} x_i} \end{aligned}$$

$$\begin{aligned} I(\lambda) &= -U'(\lambda) \\ I(\lambda) &= \frac{n}{\lambda^2} \\ I(\hat{\lambda}) &= \frac{n}{\left(\frac{n}{\sum x_i}\right)} = \sum_{i=1}^{n} x_i \end{aligned}$$

So then the 95% confidence interval for lambda is:

$$\hat{\lambda} \pm 1.96 \sqrt{\frac{1}{\sum_{i=1}^{n} x_i}} \tag{4}$$

and from this, we can evaluate the Wald Confidence interval for the median of exponential data by evaluating the true median at the lower and upper bounds of λ from the confidence interval (4) above.

$$CI_{\text{median wald}} = \left(\frac{log(2)}{\hat{\lambda_{\text{L}}}}, \frac{log(2)}{\hat{\lambda_{\text{U}}}} \right) \tag{5}$$

Implementing the Wald CI in R, I found for one bootstrap procedure with $n = 100$, the Wald CI for the median is:

$$CI_{\text{median wald}} = (0.6652383, 1.0217518)$$

```
n = 100 #change to n = 10, n = 50 and n = 100 for part (v)
lam = 1

set.seed(22275193)
expsample <- rexp(n=n, rate = lam)
x <- expsample
lambda_hat <- n/(sum(x))
ci_lambda_hat <- c(lambda_hat-1.96*sqrt(1/(sum(x))),lambda_hat
    +1.96*sqrt(1/(sum(x))) )
ci_lambda_hat

lambda_hat

#For EXP dist, true median = log(2)/lambda <- use lambda hat here
     to get
#CI for median
CI_median_WALD <- c((log(2))/ci_lambda_hat[2], (log(2))/ci_lambda
    _hat[1])
CI_median_WALD


#evaluate performance of this CI:

result_count <- 0
set.seed(22275193)
for (i in 1:1000){

  x <- rexp(n=n, rate = lam)
  lambda_hat <- n/(sum(x))
  ci_lambda_hat <- c(lambda_hat-1.96*sqrt(1/(sum(x))),lambda_hat
      +1.96*sqrt(1/(sum(x))) )
  CI_WALD <- c((log(2))/ci_lambda_hat[2], (log(2))/ci_lambda_hat
      [1])
  if ((CI_WALD[1] < TRUEmed) & (CI_WALD[2] > TRUEmed)){
    result_count = result_count +1
  }
}

result_count

result_count/10
```

When evaluating the performance of the Wald Type confidence interval for the median using equation (5), I found that for n = 100, the true median was found in the interval for 943 times out of 1000, corresponding to 94.3% confidence. The 95% Wald confidence interval performs very well for $n = 100$ here.

The Wald confidence interval performs better than the Bootstrapped confidence interval for the median of the exponential distribution for the case where $n = 100$. Next, we will investigate how these confidence intervals perform for different sample sizes.

(v) I checked the performance of the boostrap confidence interval and the Wald confidence interval by repeating the code I had in steps (iii) and (iv) for the median for sample sizes $n = 50$ and $n = 10$.

I entered my results for the performance (number of counts out of 1000 divided by 10) into the following table:

Sample Size	Bootstrapped CI performance	Wald CI performance
n = 100	98.1%	94.3%
n = 50	100%	93.5%
n = 10	100%	93.9%

Here we see that for all n values we tested, the Wald CI performs better and achieves results that are reasonably close to 95%. The CI produced using Bootstrapping does not perform as well for $n = 100$ and produces very poor results for smaller n values as we can see it counted the true median in the interval each time for $n = 50$ and $n = 10$.

The Wald Confidence interval performs better for large n values since $I(\lambda) \rightarrow \mathcal{I}(\lambda)$, for large n.

We see that the bootstrapping technique works better for generating confidence intervals on larger samples. While it does not perform as well as the Wald confidence interval here, I can see that this is a very useful technique for generating confidence intervals on data without having to make assumptions on the distribution of the population (i.e. there is no requirement for the population to be normally distributed or for the population standard distribution to be known). This is also a very easy confidence interval to implement using computer programmes like R to compute the samples for us.