Q1

R code for creating function that generates a sample of poisson data and calculates the mean of sample.

```
#Creating function
meanpois <- function(n,lambda) {
  sample_pois <- rpois(n,lambda)
  meanpois_sample <- mean(sample_pois)
  return(meanpois_sample)
}
```

Used formulas below to calculate:

mean square error = var(thetha tilda) + [bias(thetha tilda)]^2

bias (thetha tilda) = E(thetha tilda) – thetha, where E(thetha tilde) becomes mean of our result

efficiency (lambda tilda) = crlb/ var(lambda tilda)

CRLB = n/lambda

When:

Lambda = 1 and n = 10, for the 1000 samples: mean= .9907, variance = .1034069, mean square error = 0.1034934

Lambda = 1 and n = 20, for the 1000 samples: mean= .99375, variance = .05221, mean square error =.05224971

Lambda = 1 and n = 50, for the 1000 samples: mean= .996, variance = .02053974, mean square error =.02055574

Lambda = 1 and n = 200, for the 1000 samples: mean= .997185, variance = .004784685, mean square error   =.00479261

Lambda = 1 and n =400, for the 1000 samples: mean= .9978075, variance = .00233792, mean square error   =.002342732

Lambda = 1 and n =1000, for the 1000 samples: mean= .998583, variance = .00099, mean square error   =.0010015

As we can see increasing the size of n, makes our mean more accurate, making it closer to true mean of 1. We are also getting more precise since the variance and mse are also getting smaller.

Lambda = .5 and n = 20, for the 1000 samples: mean= .498, variance = .02665265, mean square error =.02665665

Lambda = .5 and n = 50, for the 1000 samples: mean= .5, variance = .01006206, mean square error =.01006206

Lambda = .5 and n = 200, for the 1000 samples: mean= .499665, variance = .00236412, mean square error =.002364239

Lambda = .5 and n = 400, for the 1000 samples: mean= .499837, variance = .00119637, mean square error =.001196403

Lambda = .5 and n = 1000, for the 1000 samples: mean= .499821, variance = .0004835, mean square error =.0004835365

Since lambda is the rate of occurrence, the true mean is now .5, and again increasing the size of the sample, will bring you closer to the true mean, and make the variance and mse smaller and more precise. Although n=50, sample mean = .5 = true mean. I imagine there was a bit of luck involved with this random sample to get exactly the true mean.

Lambda = 4 and n = 20, for the 1000 samples: mean= 3.98535, variance = .2107, mean square error =.2109232

Lambda = 4 and n = 50, for the 1000 samples: mean= 3.98974, variance = .08314, mean square error =.08324914

Lambda = 4 and n = 200, for the 1000 samples: mean= 3.9928, variance = .01925, mean square error =.01930653

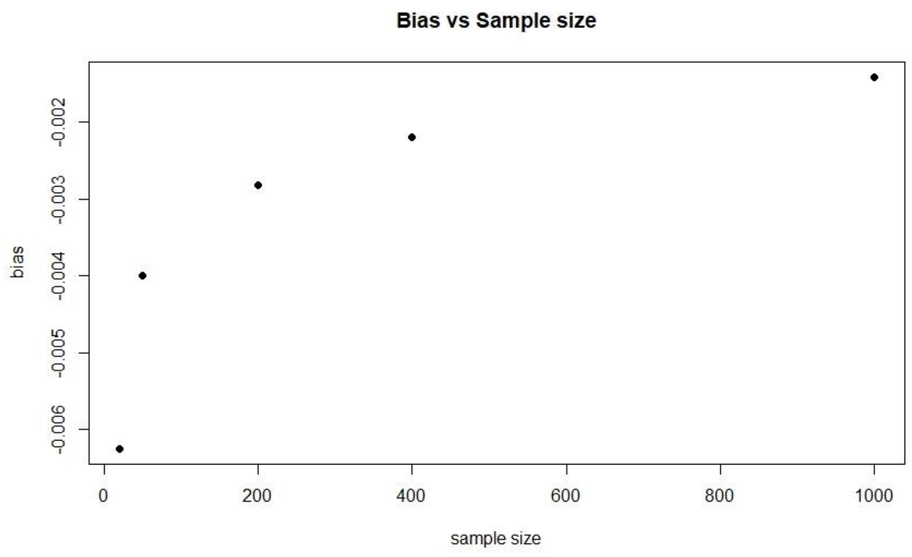
Lambda = 4 and n = 400, for the 1000 samples: mean= 3.9944, variance = .00946, mean square error =.00950017

Lambda = 4 and n = 1000, for the 1000 samples: mean= 3.9968, variance = .0039, mean square error =.003982503

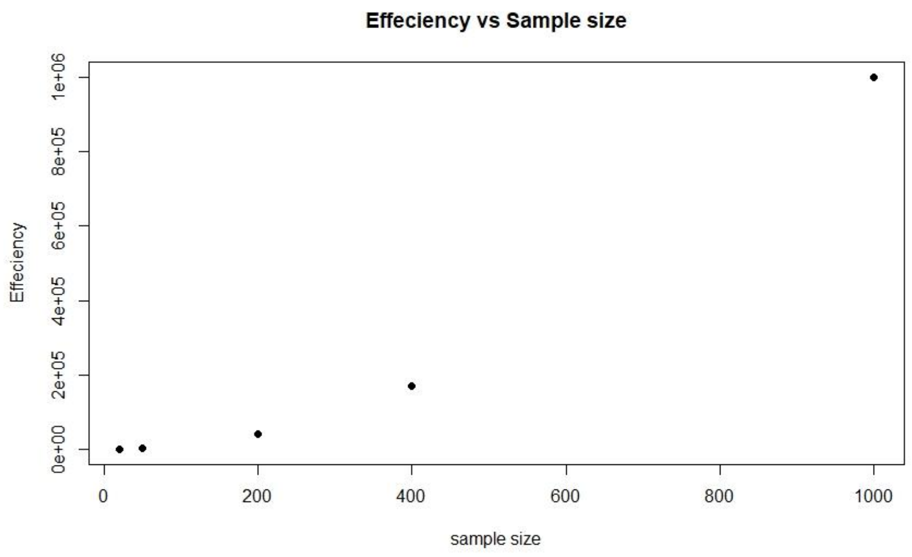
Again, true mean is 4, which when increasing sample size, the mean gets closer to 4. And the variance and mse get smaller and more precise as n increases.

Code I used to plot graphs, for the, lambda = 1 case:

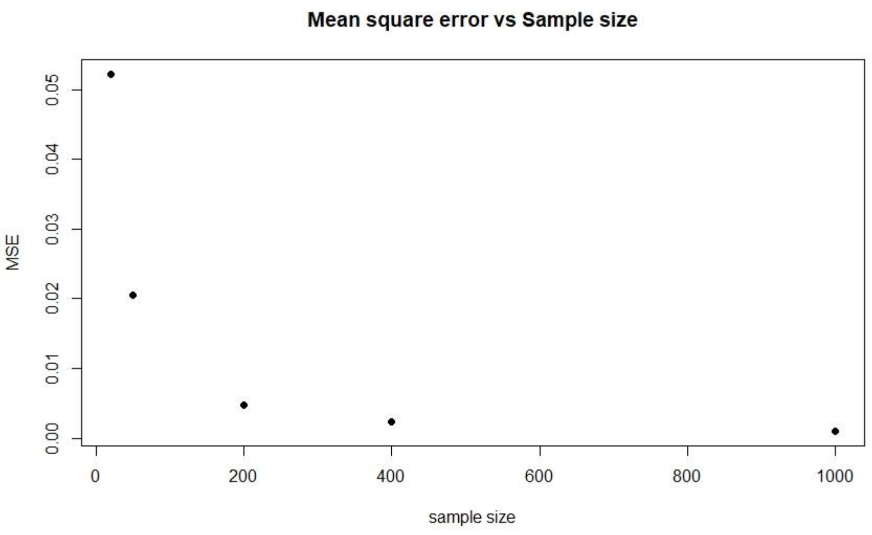
```
biasPlot <- c(bias_20,bias_50,bias_200,bias_400,bias_1000)
samplesize_plot <- c(20,50,200,400,1000)
plot(samplesize_plot, biasPlot, main ="Bias vs Sample size" , xlab =
"sample size", ylab="bias",pch=19)
```



```
effPlot <- c(eff_20,eff_50,eff_200,eff_400,eff_1000)
samplesize_plot <- c(20,50,200,400,1000)
plot(samplesize_plot, effPlot, main ="Effeciency vs Sample size" ,
xlab = "sample size", ylab="Effeciency",pch=19)
```



```
msePlot <- c(mse_20,mse_50,mse_200,mse_400,mse_1000)

samplesize_plot <- c(20,50,200,400,1000)

plot(samplesize_plot, msePlot, main ="Mean square error vs Sample
size" , xlab = "sample size", ylab="MSE",pch=19)
```



Q2

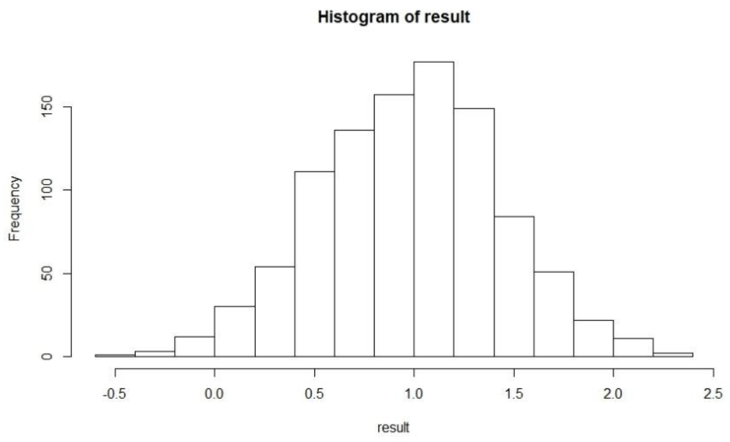
```
#Creating function

meannormal <- function(n,mean,sd) {

  sample_normal <- rnorm(n,mean,sd)

  meannormal_sample <- mean(sample_normal)

  return(meannormal_sample)

}
```

Code used to generate histogram and qq plots for normal distribution:

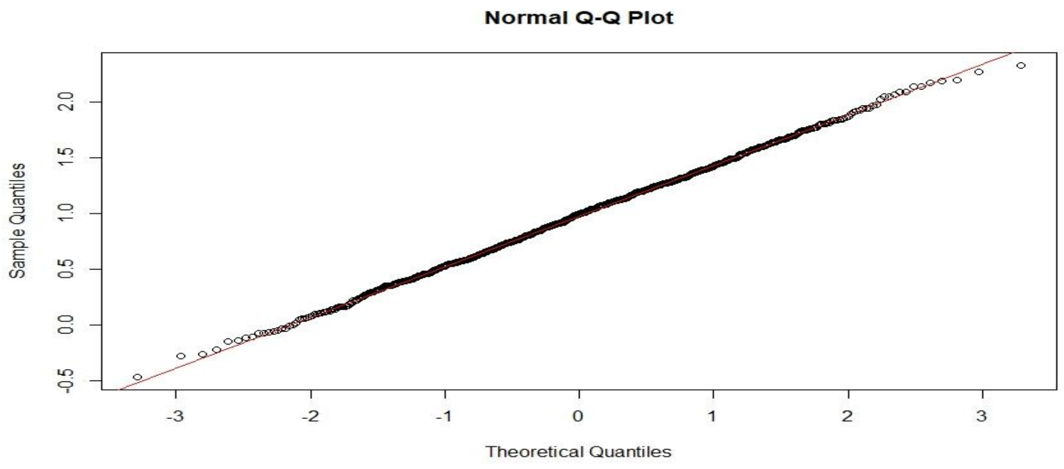
```
set.seed(16171659)

result = rep(NA,1000)

for(i in 1:1000) {

  result[i] = meannormal(5,1,1)

}


mean(result)
```

```
#histogram of the 1000 mean results
hist(result)


#qq plot
qqnorm(result)
# red line is expected values if data is normally distributed
qqline(result, col='red')
```

Normal Distribution n= 5 Plots

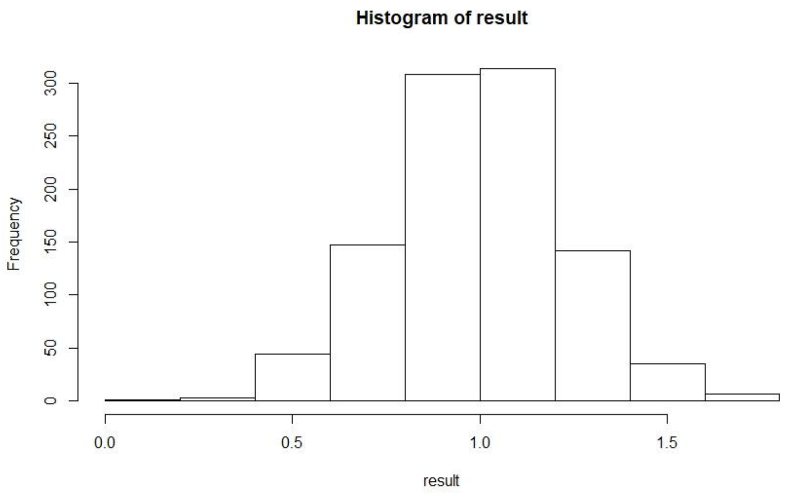


Looks normally distributed. Which we can see in the q-q plot as most of the data lies along the red line, which is the expected normal distribution. Small variation near ends of the line.

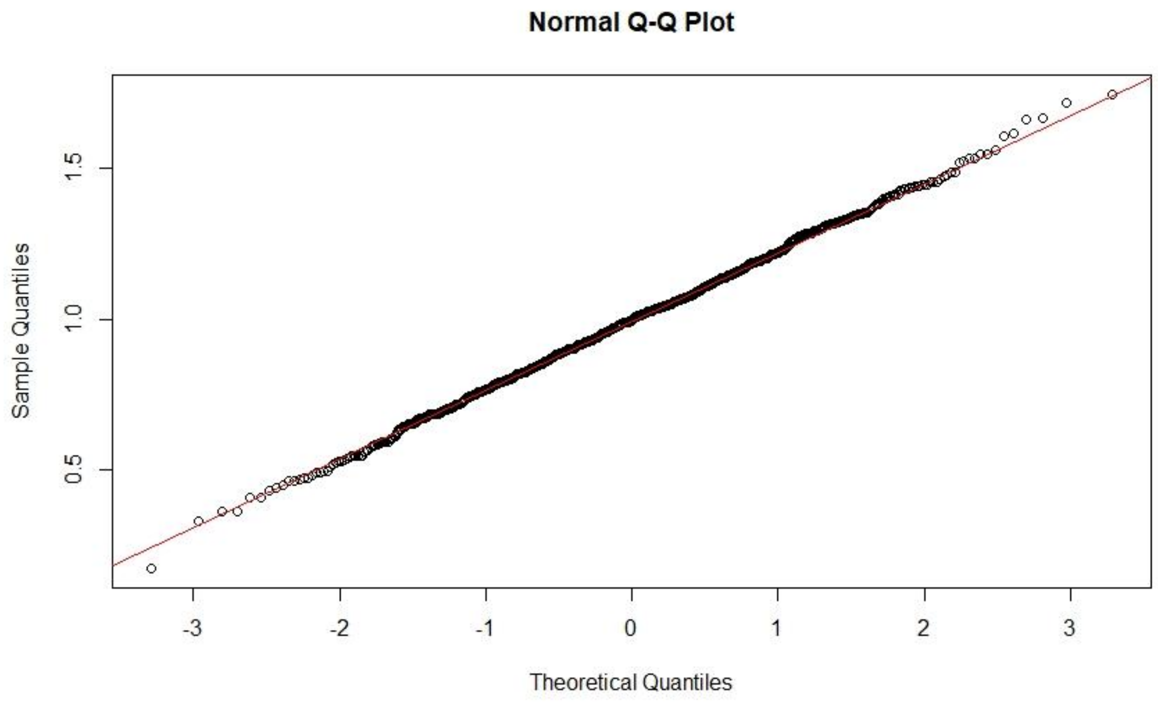


Same code is used to generate for n= 20, only changing for n in function at start of Q2.

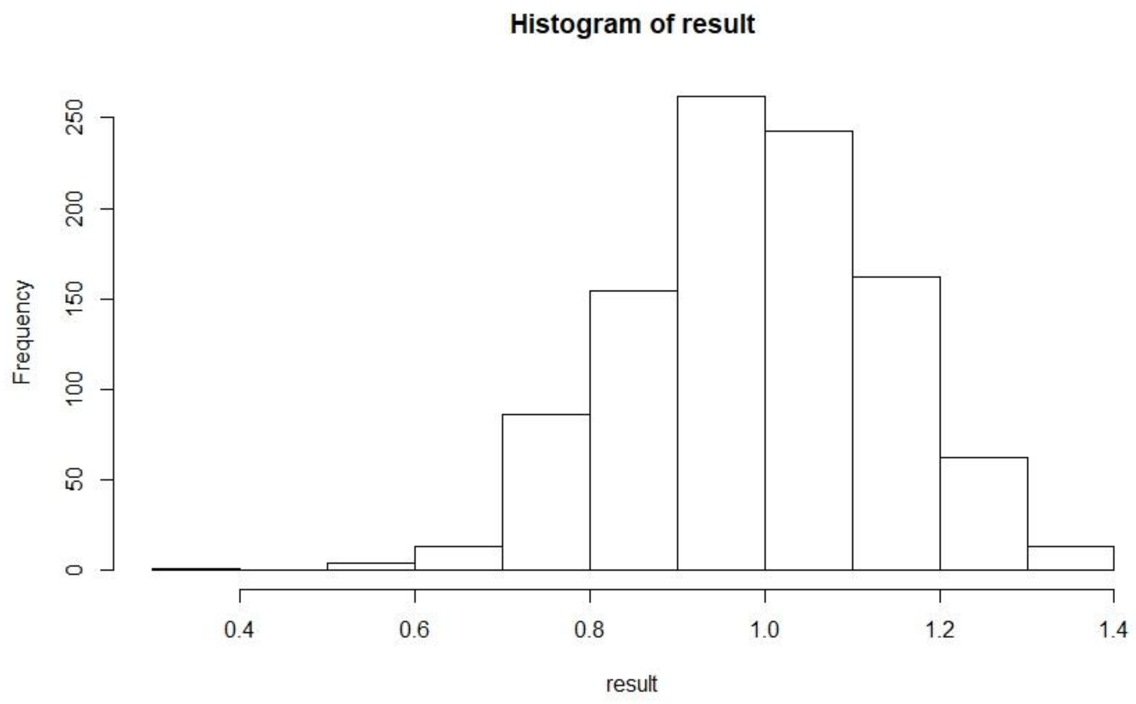
Normal dist. n = 20



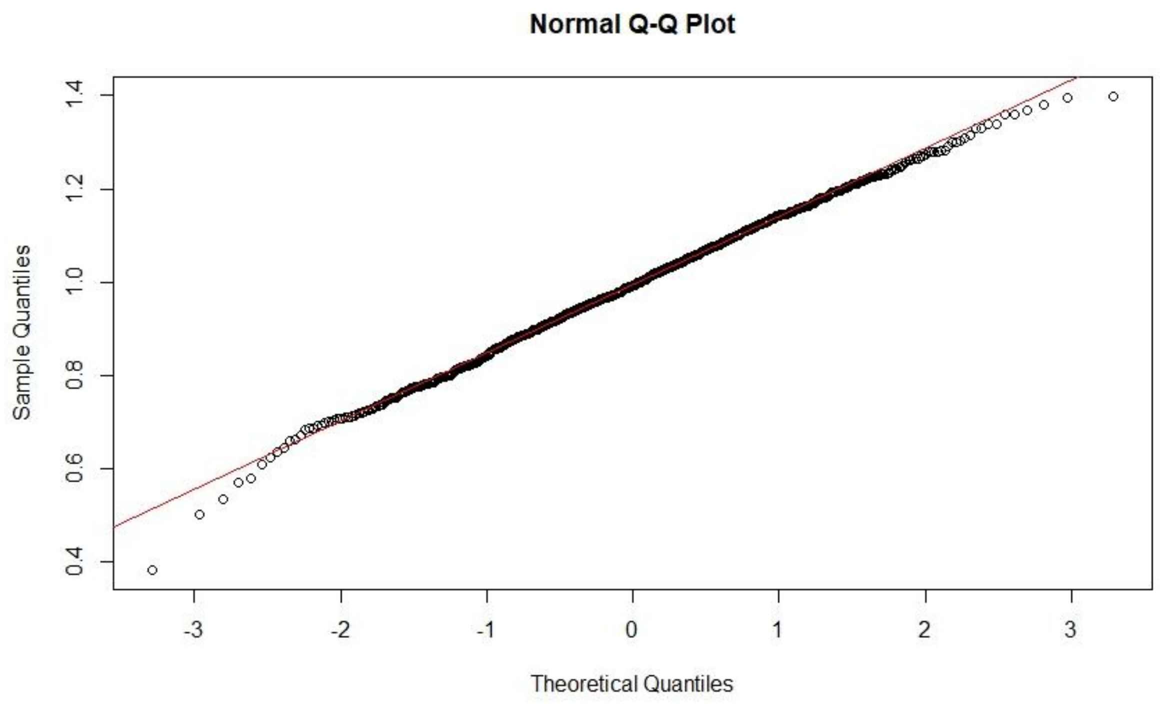
Again same as above looks normally dist. Most data lies along red line except for small variation at ends, on q-q plot.



Normal dist n=50



Again looks normally distributed. Slightly skewed to left because of small value below .4, which causes the start of the q-q plot to be a bit off the expected values.



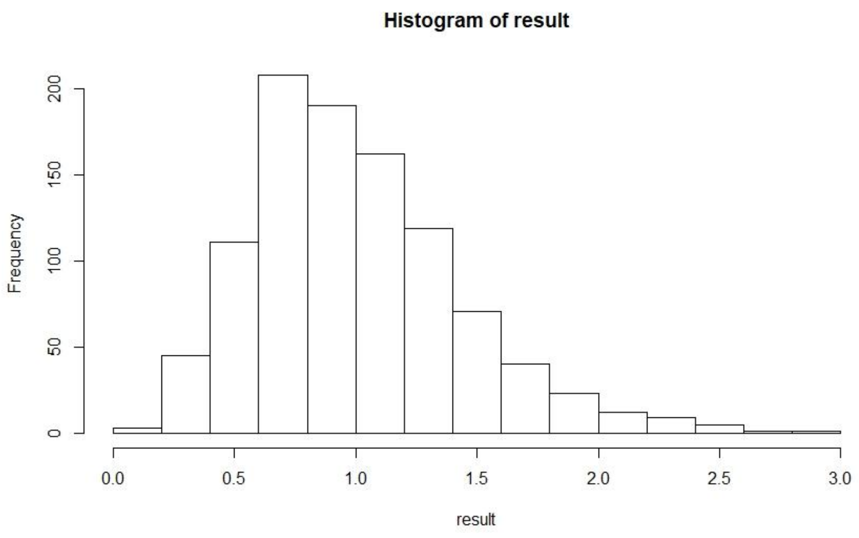
Code for exponential function:

```r
#Creating function
meanexp <- function(n,lambda) {

  sample_exp <- rexp(n,lambda)
  meanexp_sample <- mean(sample_exp)
  return(meanexp_sample)
}
```

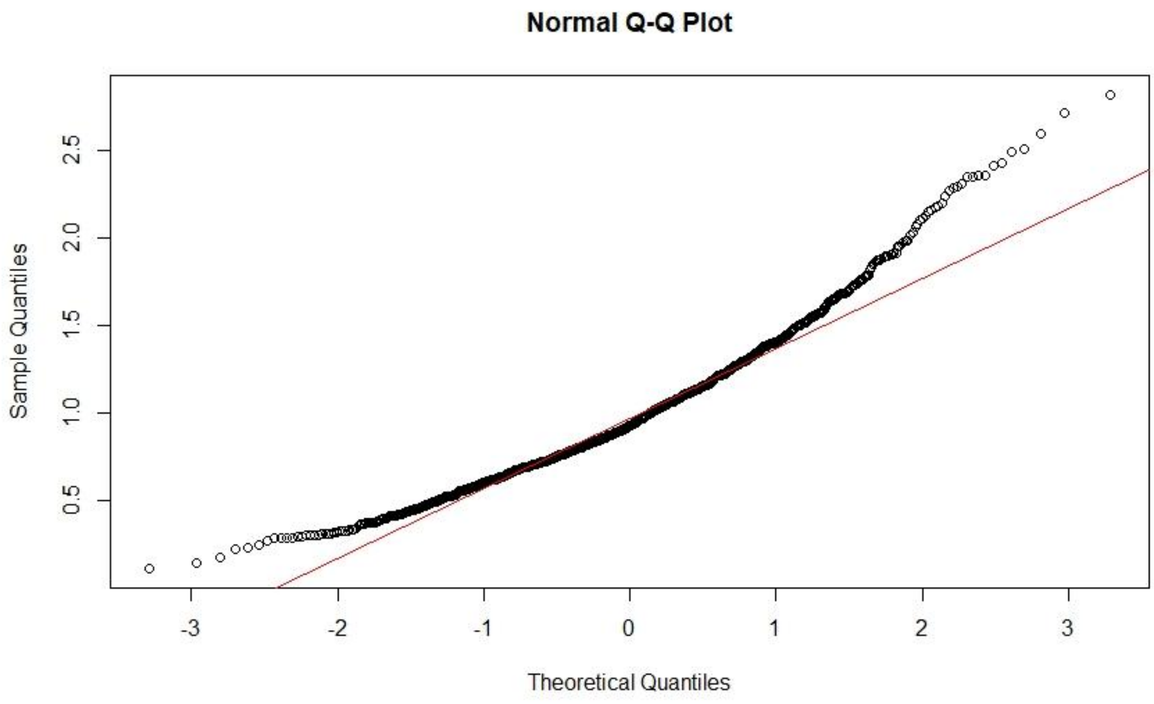
Code to generate histogram and q-q plots.

```r
set.seed(16171659)
result = rep(NA,1000)
for(i in 1:1000) {
  result[i] = meanexp(5,1)
}
mean(result)

#histogram of the 1000 mean results
hist(result)

#qq plot
qqnorm(result)
# red line is expected values if data is normally distributed
qqline(result, col='red')
```

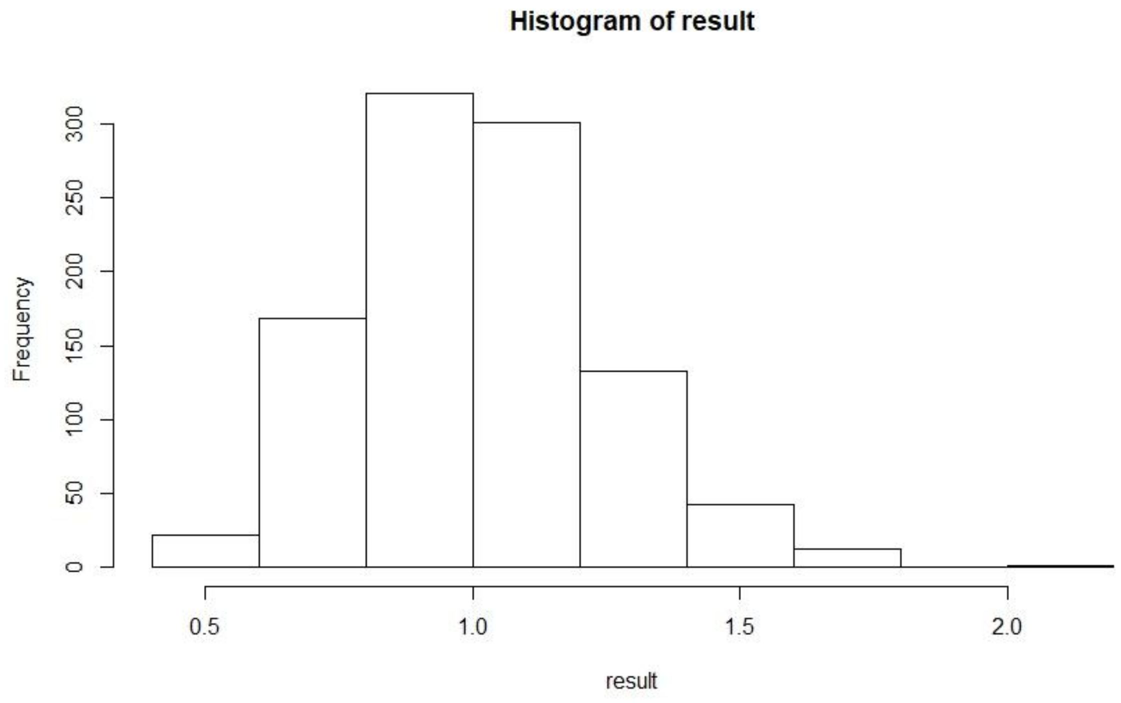
Exp distribution for n= 5



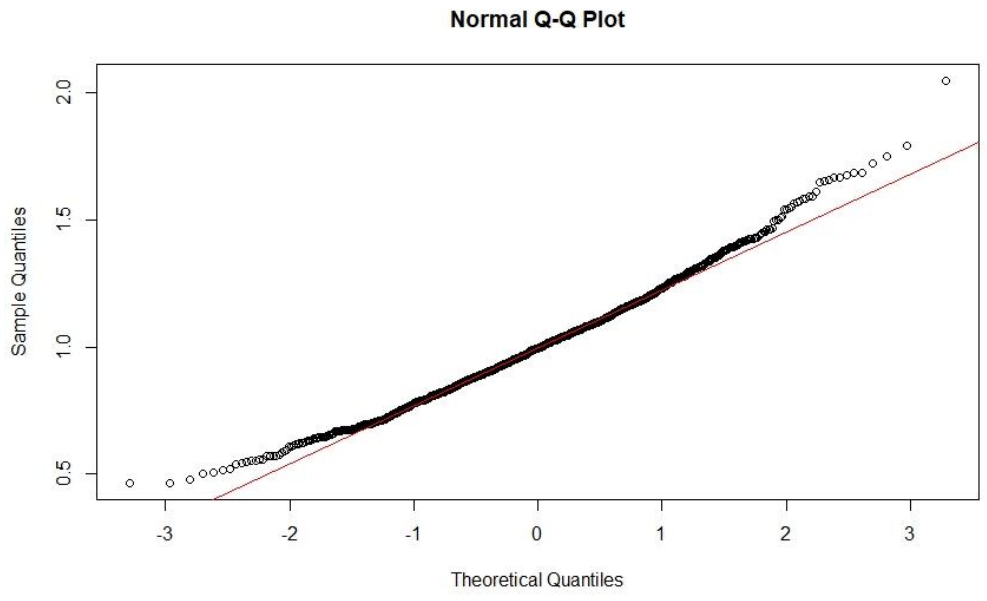
Skewed to right, which explains the q-q plot below.



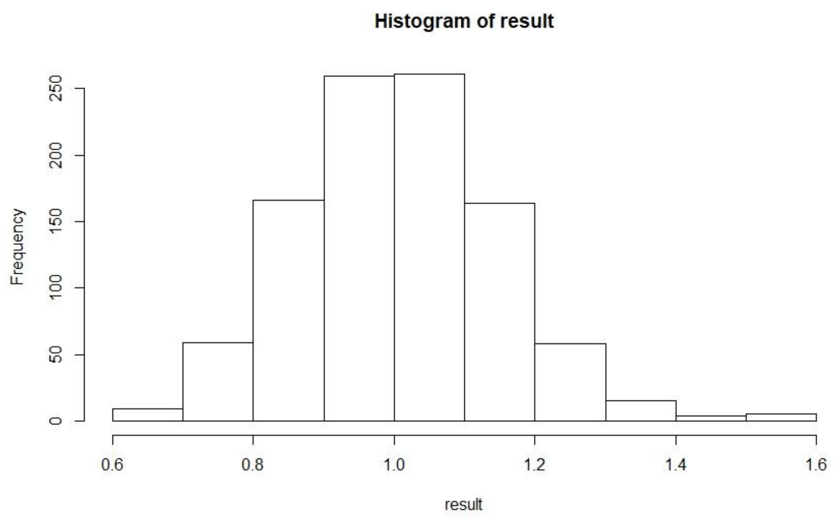
Exp dist for n= 20



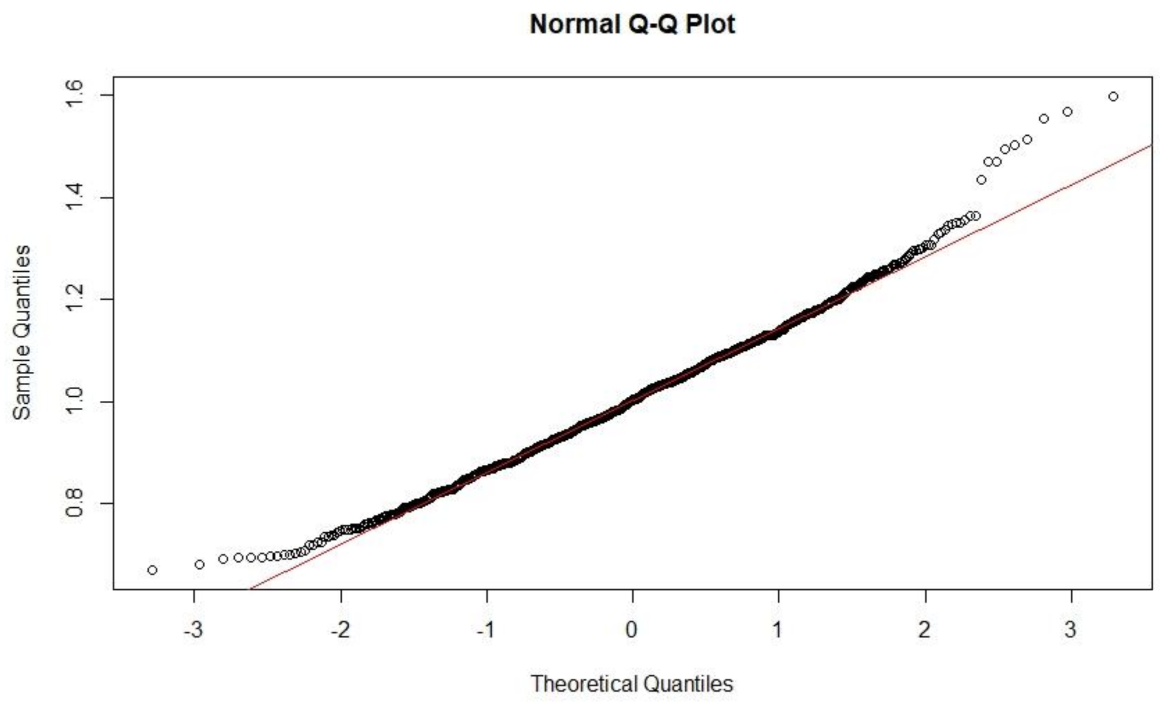
Skewed to the right but more normally distrusted than n = 5 histogram. Again, we can see this in q-q plot as the values are closer to the line.



Exp dist for n = 50



More normally distrusted than previous 2 plots for n= 5, and 20. Which is shown on q-q plot by more data on the red line. This is expected because of the Central Limit theorem, as n goes larger, the distribution will tend to the normal distribution.



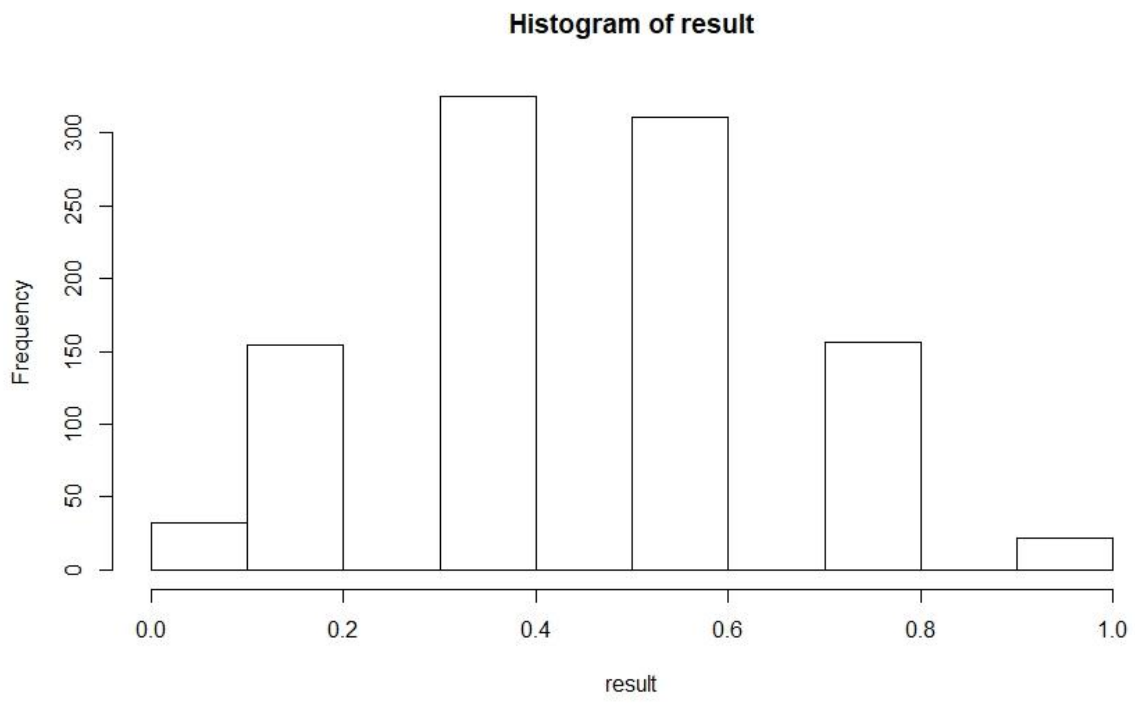
Code to bernouilli function

```
#Creating function
meanber <- function(n,size,p) {

  sample_ber <- rbinom(n,size,p)
  meanber_sample <- mean(sample_ber)
  return(meanber_sample)
}
```

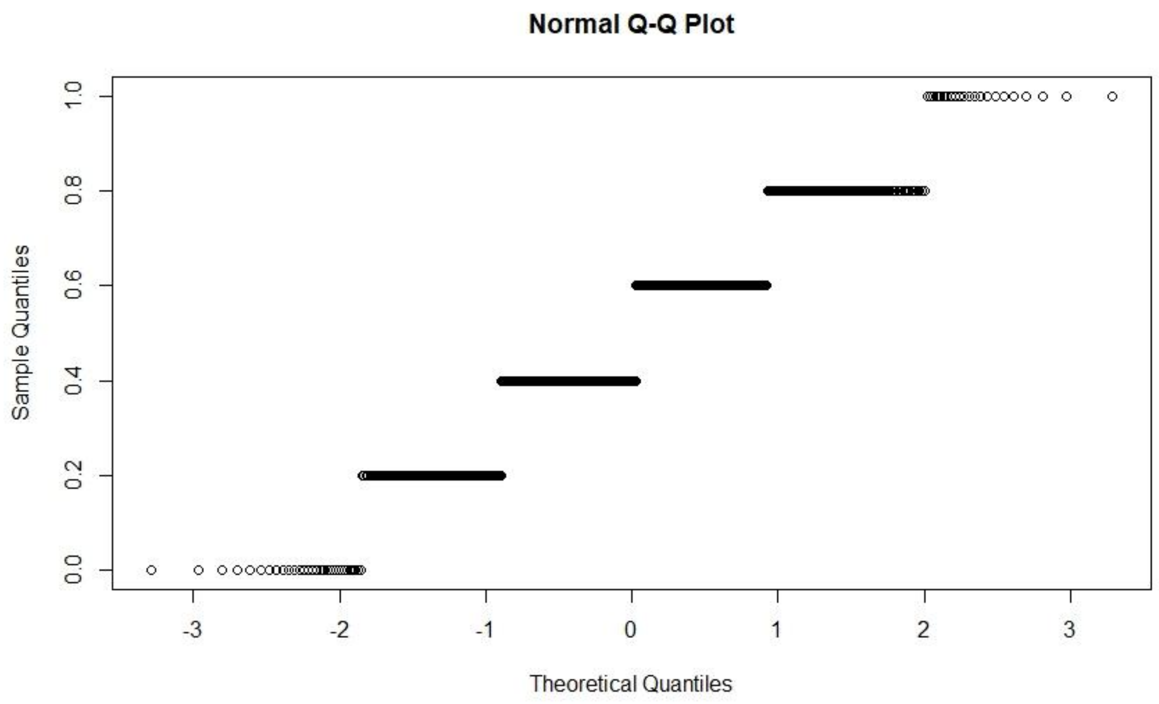
Code to generate histogram and q-q plot:

```
set.seed(16171659)
result = rep(NA,1000)

for(i in 1:1000) {
  result[i] = meanber(5,1,.5)
}
mean(result)

#histogram of the 1000 mean results
hist(result)

#qq plot
qqnorm(result)
```

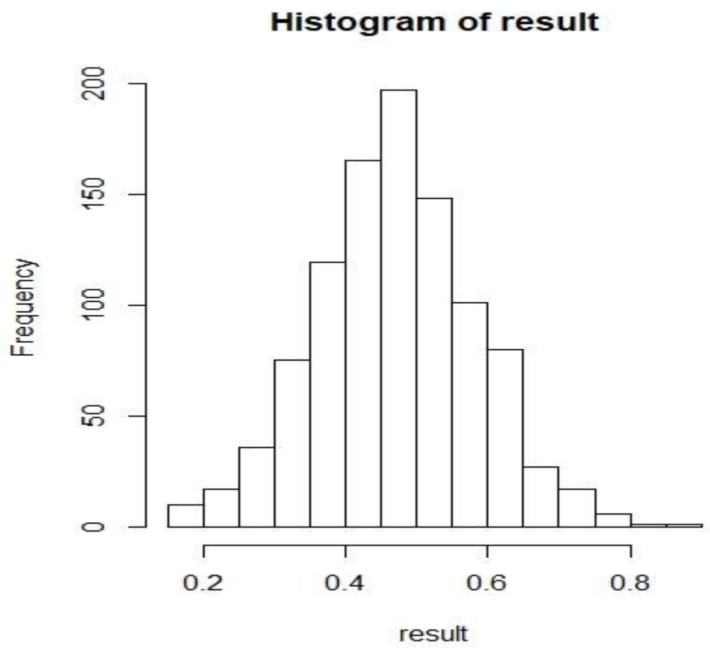
Bernoulli dist p=.5 n=5



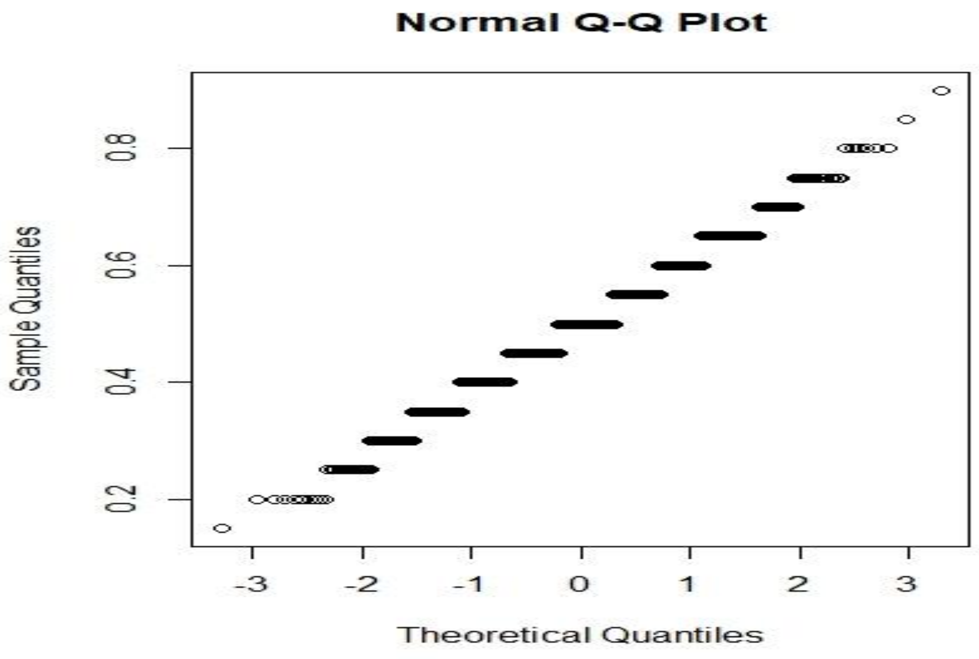
Looks normally distributed. We can see using q-q plot there is a small problem at ends of steps. To be normally distributed steps should be compact.



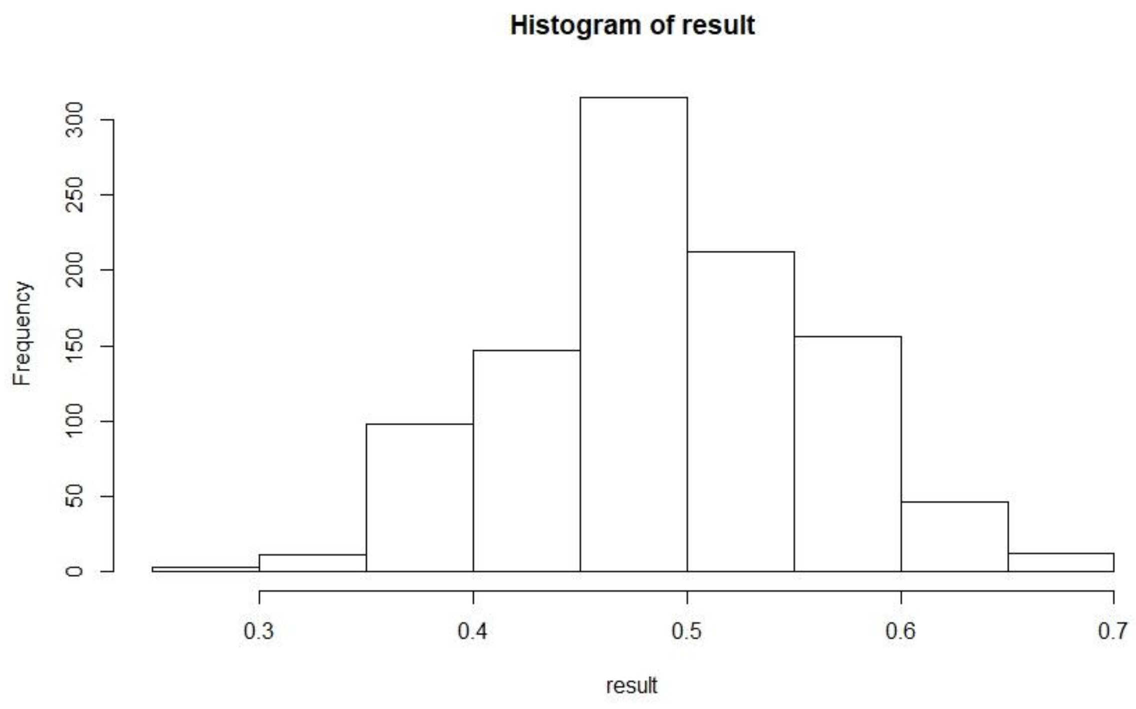
Bernoulli dist p=.5 n = 20



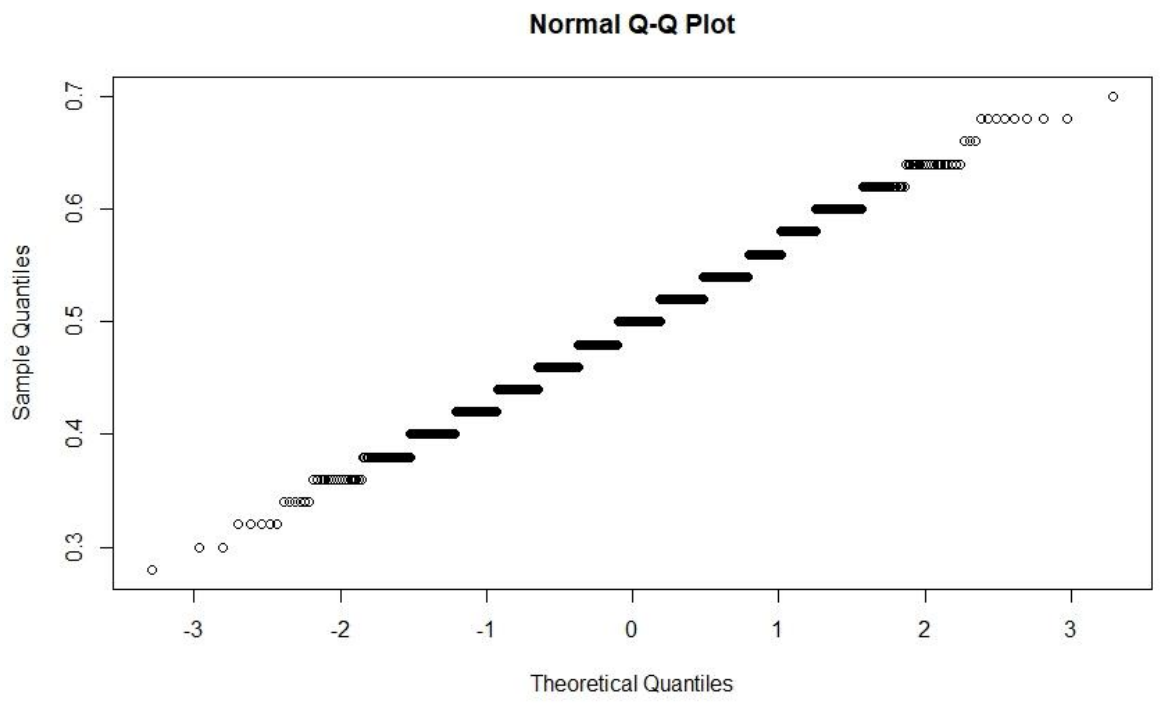
More normally distributed than n=5 plot. Small tail at right, which is seen in q-q plot.



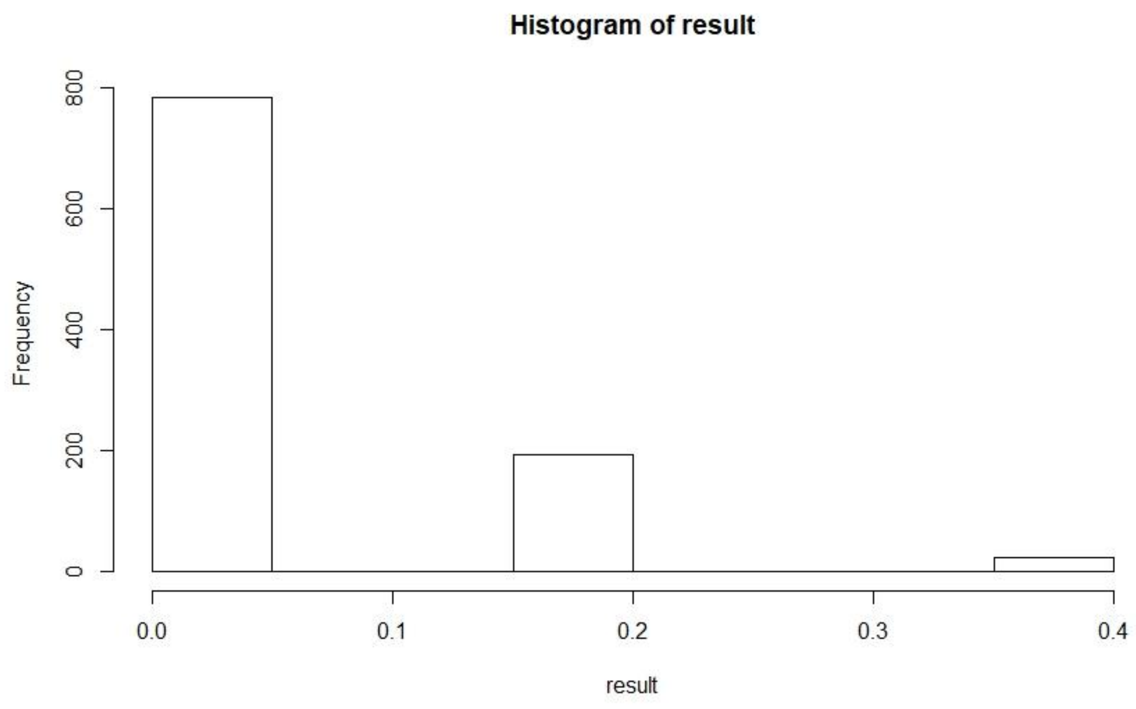
Bernoulli distribution p = .5 n = 50



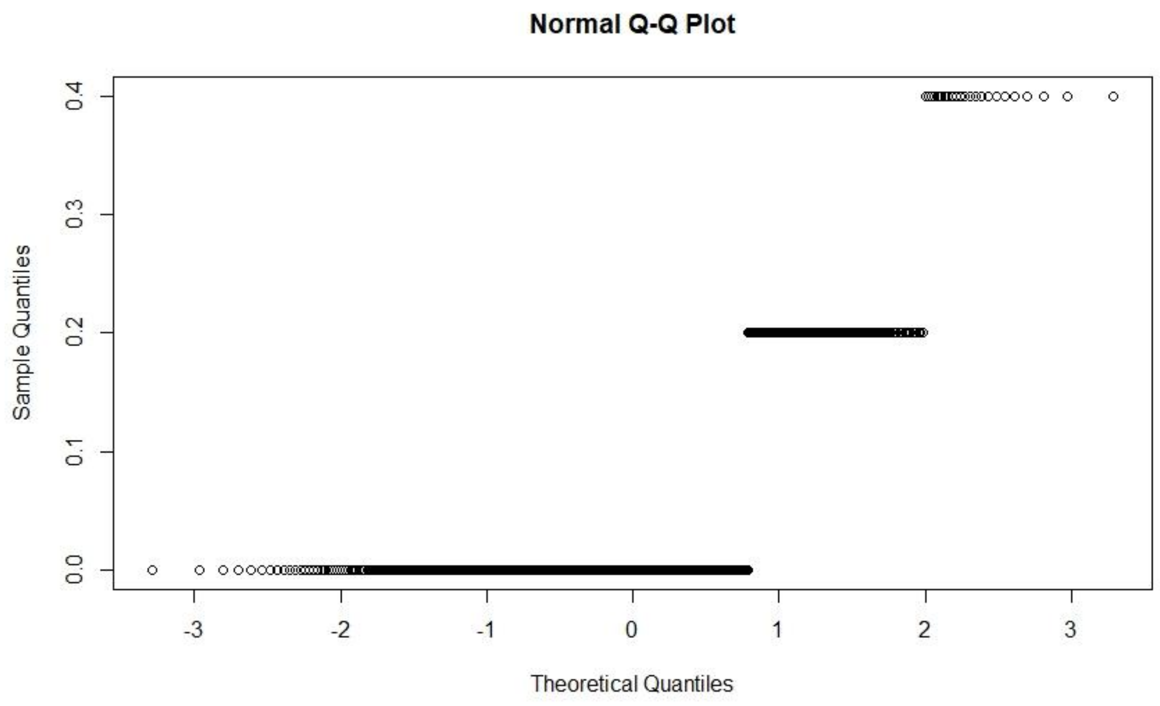
More normally distributed than previous 2. Again, can see in q-q plot.



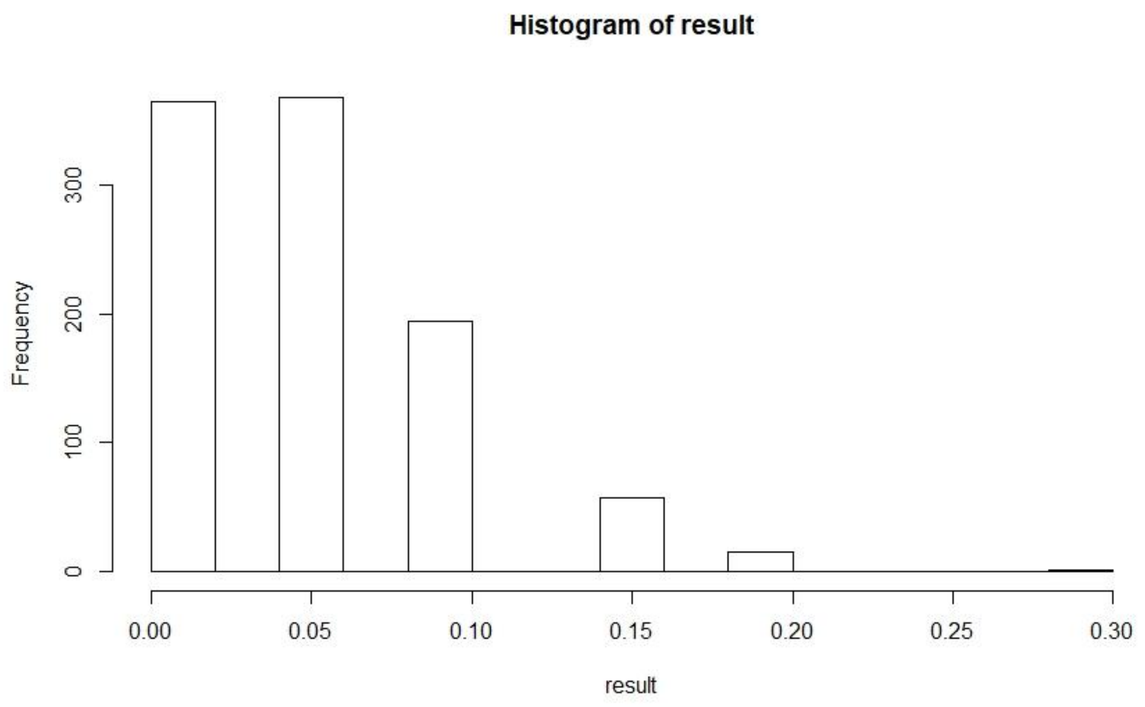
Now same again for Bernuolli p= .05, n=5



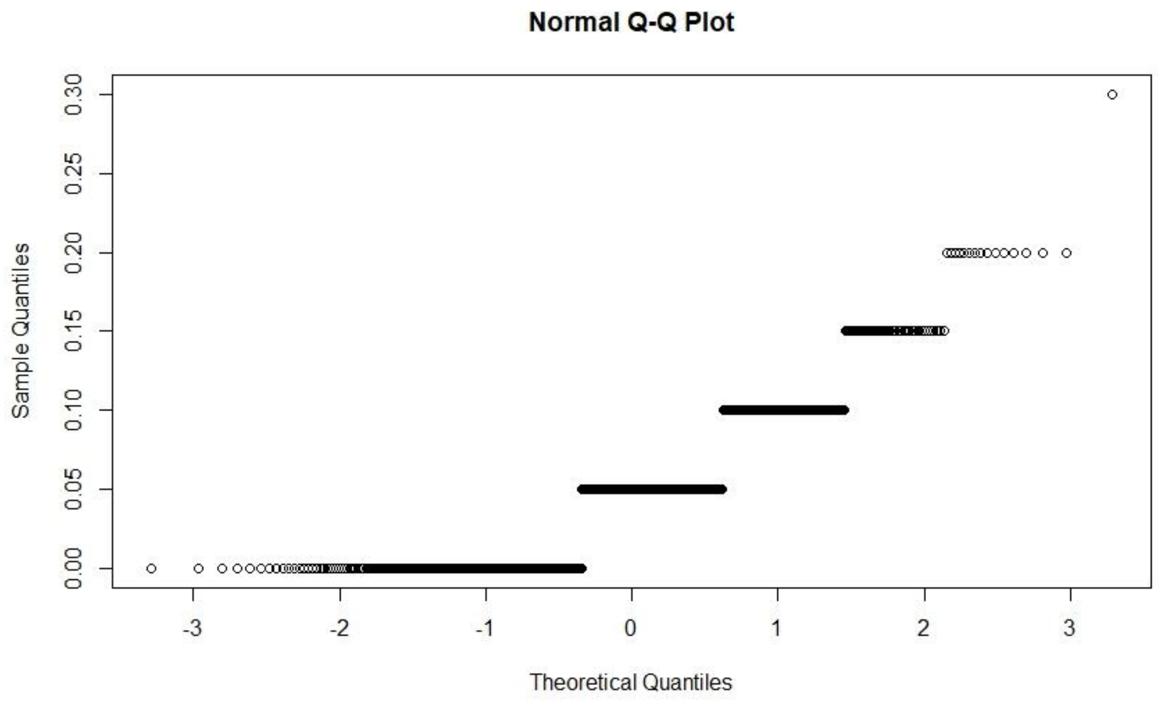
Doesn’t look normal distributed. Q-q plot shows this too.



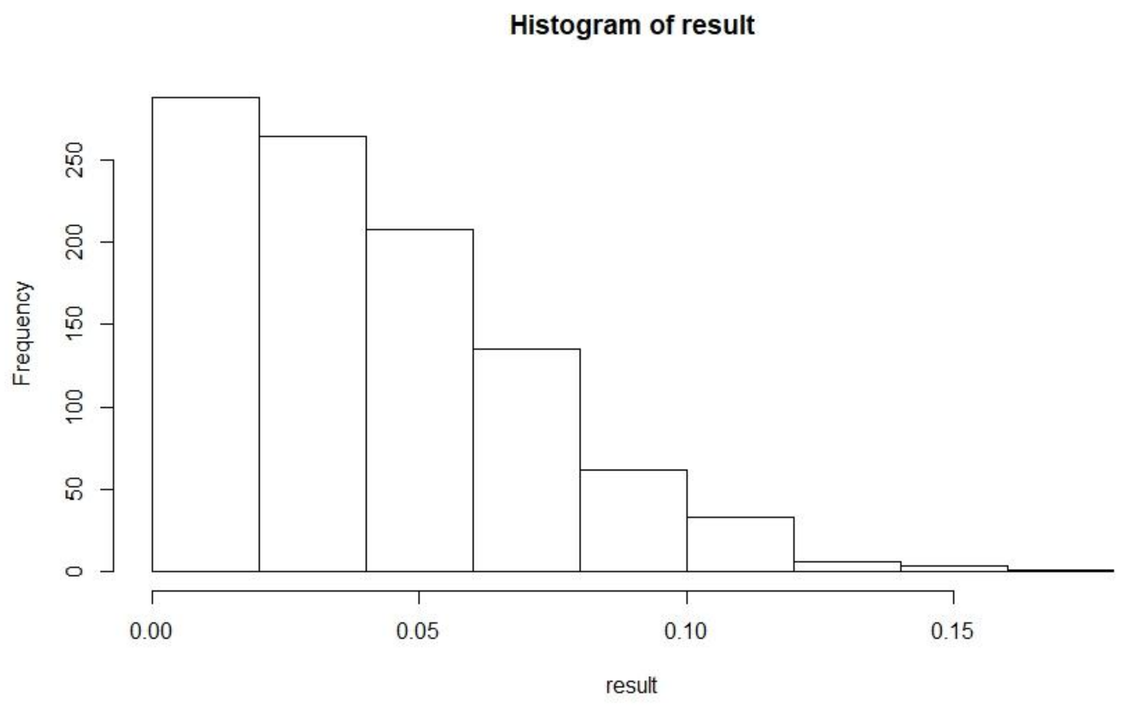
Bernoulli p=.5 n=20



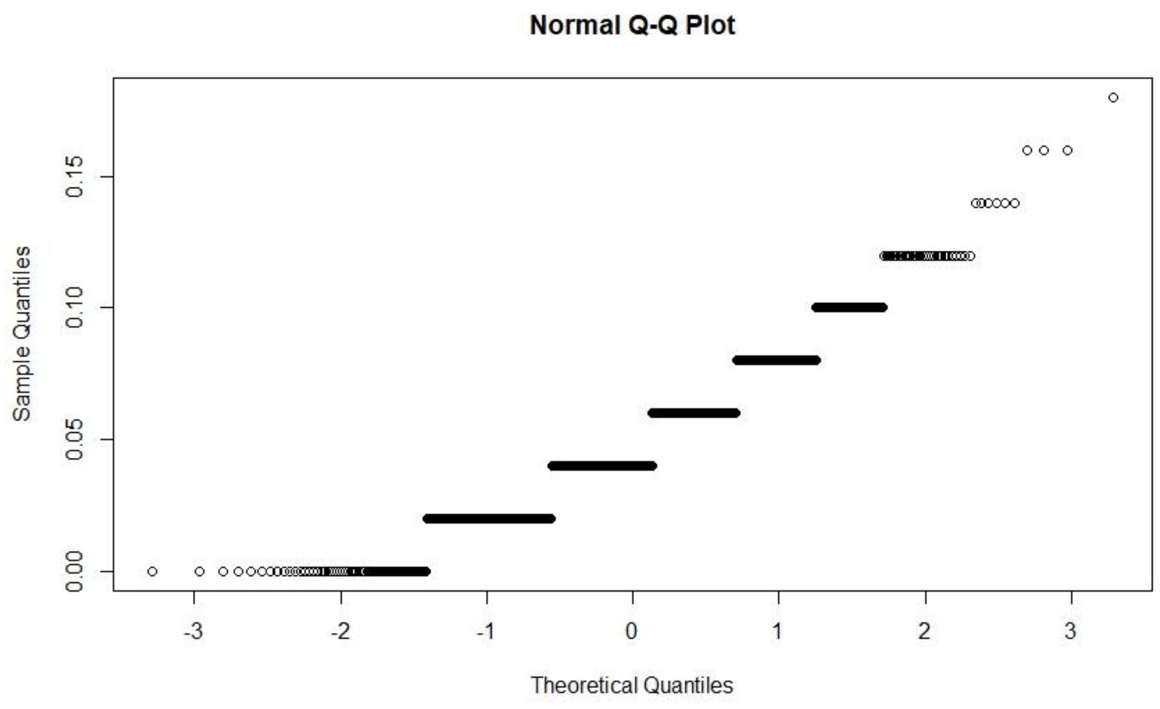
Small improvement from n=5 plot but still doesn't quite look normally distributed, same applies for q-q plot.



Bernoulli p=.05 n= 50



Again, an improvement on n=20 but still doesn't look like normal distribution. Same applies for the q-q plot,



Q3

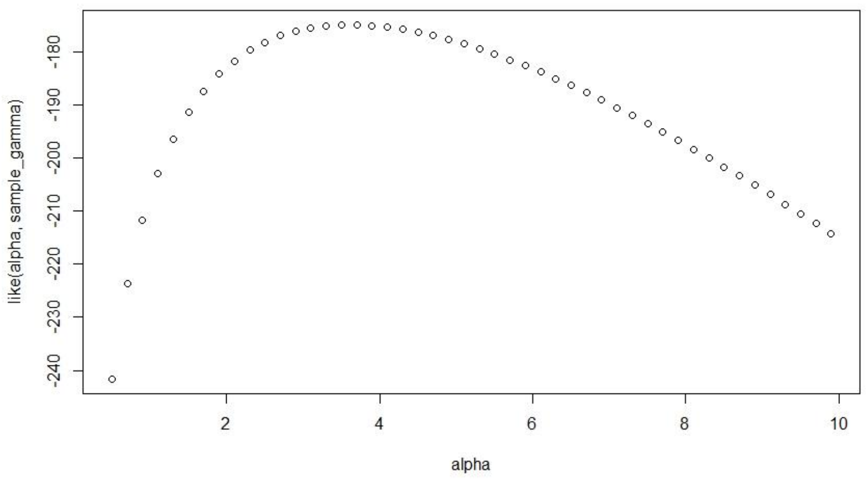
Code to generate random sample of gamma distribution

```
 sample_gamma <- rgamma(100,3,1)
```

Maximum Likelihood estimator = n*alpha / sum of xi   or alpha / xbar

Code to generate plot likelihood function

```
  like <- function(alpha, x) {
   a = alpha
   n=length(x)
  lambda = (n * a)/sum(x)
    like = n*a*log(lambda) + (a-1)* sum(log(x)) - lambda*sum(x) -
n*log(gamma(a))
    like
  }

  alpha = seq(from=0.5, to = 10, by = .2)
  plot(alpha,like(alpha,sample_gamma))
From graph, alpha hat is roughly ~ 3.6, which gives lambda hat =
(alphahat/xbar) ~1.2
```



Code for minus log likelihood, and nlm, to calculate alpha hat and lambda hat

```
set.seed(16171659) ; alpha = 3 ; lambda = 1
sample_gamma <- rgamma(100,shape = alpha ,scale = lambda )

  gll <- function(thetha,datta)
  {
    a <- thetha[1] ; b<- thetha[2]
    n = length(datta)

    sumg = sum(datta)
    sumlogg <- sum(log(datta))

    gll <- n*a*log(b) + n *lgamma(a) + sumg/b - (a-1)*sumlogg
    gll

  }

  momalpha <- mean(sample_gamma)^2 / var(sample_gamma)
  momlambda <- var(sample_gamma)/ mean(sample_gamma)

  gammasearch = nlm(gll,c(momalpha,momlambda), hessian =T, datta =
sample_gamma)

  V_hat = solve(gammasearch$hessian)

  SEalphahat <- sqrt(V_hat[1,1])
  SElambdahat <- sqrt(V_hat[2,2])

  alphahat = gammasearch$estimate[1]
  lambdahat = gammasearch$estimate[2]

  Lalpha = alphahat - 1.96* SEalphahat
  Ualpha = alphahat + 1.96* SEalphahat


  Llambda = lambdahat - 1.96*SElambdahat

  Ulambda = lambdahat + 1.96*SElambdahat


  cat("\nEstimated alpha = " , round(alphahat,3) , " 95 % CI is ",
round(Lalpha,3), " to " , round(Ualpha,3), "\n\n")


  cat("\nEstimated lambda = " , round(lambdahat,3) , " 95 % CI is ",
round(Llambda,3), " to " , round(Ulambda,3), "\n\n")
```

Result from running above code,

Estimated alpha = 3.605 95 % CI is .647 to 4.564

Estimated lambda = 0.809 95 % CI is 0.578 to 1.039

Q4

Code for (i)

```
  normal_95CI <- function (n,u,s)
  {
    #n is sample size
    #u is sample mean
    #s is sample standard deviation
    sample <- rnorm(n,u,s)
    x_bar = mean(sample)
    s_d = sd(sample)

    error <- qnorm(.975) * s_d/sqrt(n)
    Z_Lbound = x_bar - error
    Z_Ubound = x_bar + error

    cat(" 95 % CI is ", round(Z_Lbound,3), " to " ,
round(Z_Ubound,3), "\n\n")
}
```

(ii)

```
  t_95CI <- function(n,u,s)
  {
    #n is sample size
    #u is sample mean
    #s is sample standard deviation
    sample <- rnorm(n,u,s)
    x_bar = mean(sample)
    s_d = sd(sample)

    error <- qt(.975, df = n-1) * s_d/sqrt(n)

    T_Lbound <- x_bar - error
    T_Ubound <- x_bar + error

    cat(" 95 % CI is ", round(T_Lbound,3), " to " ,
round(T_Ubound,3), "\n\n")
     }
```

(iii)

Use t distribution if sample size is smaller than 30 (n < 30). The t-distribution incorporates the fact that for smaller sample sizes the distribution will be more spread out, by using degrees of freedom. That's why it is better to use t-distribution for samples of size less than 30, rather than normal distribution.

Code for testing above function by running 1000 times and checking if true mean lies within CI.

```
  count_z_10 = 0 #count the number of times true mean is containing
with z CI

  #we know true mean is 1, since we set it, so for it to be
contained in interval, our lower bound must be less than one and
upper greater than 1
```

```
  # n=10 u = 1 s = 1, using z

  for(i in 1:1000 )
  {
     # intervals attained in (i) by
    x <- rnorm(10,1,1)
    x_bar = mean(x)
    s_d = sd(x)
    error <- qnorm(.975) * s_d/sqrt(10)

    l = x_bar - error
    ub = x_bar + error

    if( l< 1 &  ub > 1 ) count_z_10 = count_z_10+1
    else count_Z_10 = count_z_10


  }
   count_z_10
```

The true mean was contained in 927 out of 1000.

Code for t distribution, n=10

```
   # n=10 , u = 1, s = 1  using t

   count_t_10 = 0

  for(j in 1:1000)
  {
    # intervals attained in (ii) by
    sample <- rnorm(10,1,1)

    x_bar = mean(sample)
    s_d = sd(sample)

    error <- qt(.975, df = 10-1) * s_d/sqrt(10)

    T_Lb <- x_bar - error
    T_Ub <- x_bar + error

    if( T_Lb < 1 &  T_Ub > 1 ) count_t_10 = count_t_10+1
    else count_t_10 = count_t_10

  }
  count_t_10
```

The true mean was contained in 937 out of 1000. Which is more accurate than using normal distribution CI.

Now for n = 500

```
  # n=500 u = 1 s = 1, using z
  count_z_500 = 0

  for(k in 1:1000 )
  {
    # intervals attained in (i) by
    x <- rnorm(500,1,1)
    x_bar = mean(x)
    s_d = sd(x)
    error <- qnorm(.975) * s_d/sqrt(500)

    l = x_bar - error
    ub = x_bar + error
    if( l< 1 &  ub > 1 ) count_z_500 = count_z_500+1
    else count_Z_500 = count_z_500

  }
  count_z_500
```

The true mean was contained in 947 out of 1000. Which is more accurate than n=10 samples.

Now for n=500 using t-distribution, which tends to normal distribution when n is large (Central Limit Theorem)

```
  # n=500 , u = 1, s = 1  using t

  count_t_500 = 0

  for(m in 1:1000)
  {
    # intervals attained in (ii) by
    sample <- rnorm(500,1,1)

    x_bar = mean(sample)
    s_d = sd(sample)

    error <- qt(.975, df = 500-1) * s_d/sqrt(500)

    T_Lb <- x_bar - error
    T_Ub <- x_bar + error

    if( T_Lb < 1 &  T_Ub > 1 ) count_t_500 = count_t_500+1
    else count_t_500 = count_t_500

  }
```

```
count_t_500
```

The true mean was contained in 958 out of 1000. Which again is more accurate than n=10 samples.

Q5

Bootstrapping - is a method that generates multiple samples of size n by sampling a sample with replacement. This technique allows estimation of the sampling distribution of almost any statistic using random sampling methods

example, say sample is 1,2,3,4,5, then possible bootstrap samples are:

1,2,3,4,4

2,3,4,5,2

3,3,3,3,3

4,4,2,2,1,

(ii) Code to generate exponential data and bootstrap, and produce the confidence intervals.

```
  set.seed(16171659)


   # sample of 100 random exp distribution

    sample_exp = rexp(100,1)


  #obtaining 1000 bootstrap values

  resamples <- lapply(1:1000, function(i) 
sample(sample_exp,length(sample_exp),replace = TRUE))


  r.median <- sapply(resamples, median, na.rm = TRUE)

   quantile(r.median, c(.025,.975))
```

(iii) Code to test the confidence interval

```
# true median for exp distribution is, median = lambda * ln2


  true_median = 1 * log(2)

  count_exp= 0


  for(n in 1:1000)

  {


    # sample of 100 random exp distribution

    sample_exp = rexp(100,1)


    #obtaining 1000 bootstrap values

    resamples <- lapply(1:1000, function(i)
sample(sample_exp,length(sample_exp),replace = TRUE))


    r.median <- sapply(resamples, median, na.rm = TRUE)


    E_Lb = quantile(r.median,.025)

    E_Ub = quantile(r.median,.975)


    if( E_Lb <= true_median  &&  E_Ub >= true_median ) count_exp =
count_exp+1

    else count_exp = count_exp


  }


  count_exp
```

The true median was contained in 941 of the confidence intervals.

(iv) Wald type CI for lambda and then hence median

```
#Asymptotic distribution of the mle (wald).
# I(lambda) = lambda^-2
# I(lambda) = 1/lambda^2
# So CI is :
# lambda +- 1.96 sqrt(lambda^2 /n)
# Using mle for lambda in above
# xbar +- 1.96 sqrt(xbar^2 /n)



  set.seed(16171659)


  # true lambda is 1
  true_lambda = 1
  true_median = 1 * log(2)


  count_med =0
  count_lambda= 0



  for(n in 1:1000)
  {


    # sample of 100 random exp distribution
    sample_exp = rexp(100,1)


    #obtaining 1000 bootstrap values
    resamples <- lapply(1:1000, function(i)
sample(sample_exp,length(sample_exp),replace = TRUE))


    r.mean <- sapply(resamples, mean, na.rm = TRUE)


    mle <- mean(r.mean)
    lambda_Lb= mle - 1.96 * sqrt(mle^2 / 100)
    lambda_Ub= mle + 1.96 * sqrt(mle^2 / 100)



    if( lambda_Lb <= true_lambda  &&  lambda_Ub >= true_lambda )
      count_lambda = count_lambda+1
    else
      count_lambda = count_lambda

    # now using for median
    #median = ln(2) / lambda
    median_exp = log(2)/ (mle)

    # now use median_exp instead of mle in CI formula

  med_Lb = median_exp - 1.96 * sqrt(median_exp^2 / 100)
  med_Ub =  median_exp + 1.96 * sqrt(median_exp^2 / 100)



  if( med_Lb <= true_median  &&  med_Ub >= true_median )
      count_med = count_med+1
  else
      count_med = count_med


  }

  count_lambda
  count_med
```

True lambda was contained 941 times in the CI. While true median was contained 941 times also.

(v)

Result for (iii) n = 10

The true median was contained in the CI 934 times.

(iii) n= 50

The true median was contained in the CI 947 times. Better n=10 sample.

Result for (iv) n =10

True lambda was contained in the CI 914 times.

True median was contained in the CI 946 times.

(iv) n = 50

True lambda was contained in the CI 952 times.

True median was contained in the CI 953 times.