SS2864B Assignment 5

Ali Al-Musawi

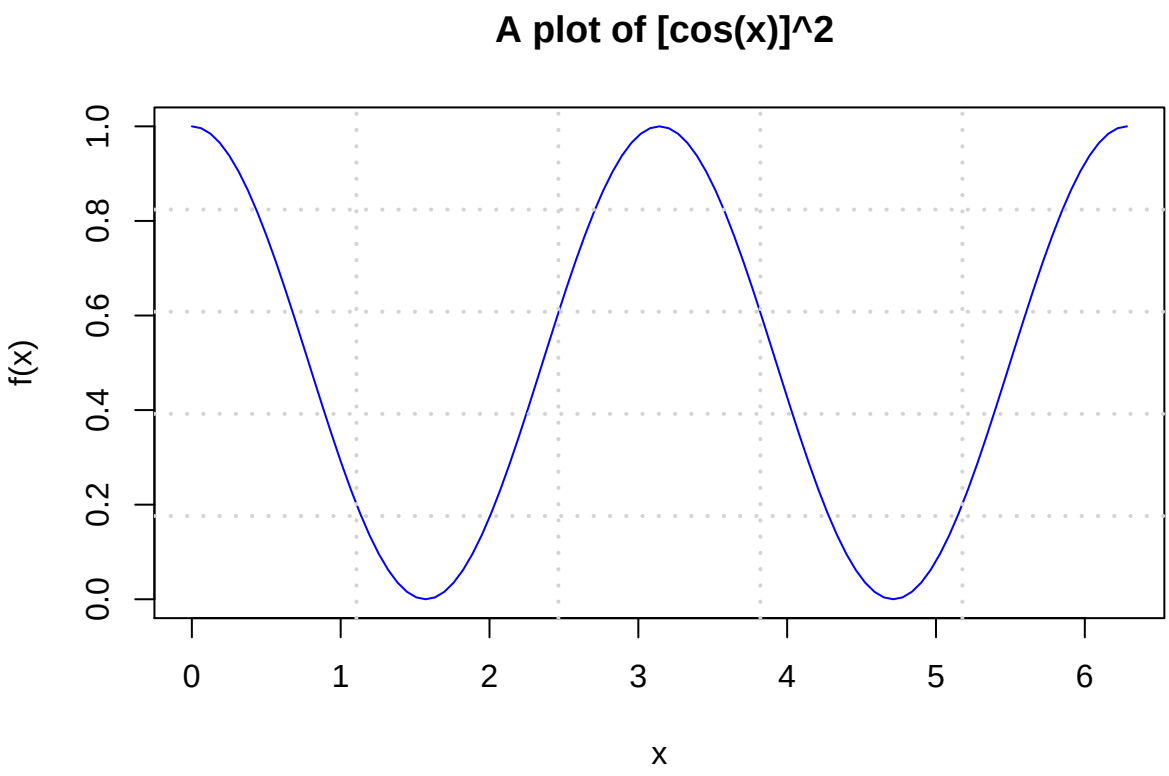
13/03/2020

Question 1

```
# define the given function
f1 = function(x) {
  return ((cos(x))^2)
}

# define a CI function
ci.t.95 = function(x) {
  # compute lower and upper bounds
  lower = mean(x) - 1.96*sd(x)/sqrt(length(x))
  upper = mean(x) - 1.96*sd(x)/sqrt(length(x))
  # construct a list object
  result = list(Lower.Bound = lower, Mean = mean(x), Upper.Bound = upper)
  return (result)
}
```

(a)

```
curve(f1, from = 0, to = 2*pi, xname = 'x',
      ylab = 'f(x)', main = ' A plot of [cos(x)]^2',
      col = 'blue')
grid(5, lwd = 2)
```



(b)

```r
# set seed for reproducibility
set.seed(123)
# define a random uniform variable on [0, 2pi]
U = runif(n = 1e6, min= 0, max = 2*pi)
# perform MC integration
I1 = mean(f1(U))*(2*pi)
# compute a 95% ci
ci = ci.t.95(f1(U)*(2*pi))
# display ci
ci
```

```
## $Lower.Bound
## [1] 3.13889
##
## $Mean
## [1] 3.143241
##
## $Upper.Bound
## [1] 3.13889
```

(c)

We use some identities to simplify the square of cosine before integrating it:

$$\cos^2(x) = 1 - \sin^2(x) \text{ (Pythagorean Identity)}$$

$$\sin^2(x) = \cos^2(x) - \cos(2x) \text{ (Cosine Double Angle Identity)}$$

$$\therefore 2\cos^2(x) = 1 - \cos(2x) \text{ (Combine Two Identities)}$$

$$I = \int_0^{2\pi} f(x)dx = \int_0^{2\pi} \cos^2(x)dx = \frac{1}{2}\int_0^{2\pi} 1 - \cos(2x)dx$$

$$I = \frac{1}{2}\left(x - \frac{1}{2}\sin(2x)\right)\Big|_0^{2\pi}$$

$$\therefore I = \frac{1}{2}(2\pi) = \pi$$

Based on the output in part **b**, indeed, the CI covers the true value if we assume the true value = 3.14.

Question 2

(a)

```
set.seed(222)
# Generate 100 x_i such that X~N(10, 4)
x = rnorm(n = 100, mean = 10, sd = 2)
# Generate summary statistics of the x_i's
min(x)
```

```
## [1] 4.451066
```

```
max(x)
```

```
## [1] 14.41816
```

```
mean(x)
```

```
## [1] 10.04396
```

```
sd(x)
```

```
## [1] 1.916051
```

(b)

```
# Generate 50000 different samples and calculate their means
out = replicate(50000, mean(sample(x, replace=TRUE)))
# Generate summary statistics of the means
min(out)
```

```
## [1] 9.243029
```

```
max(out)
```

```
## [1] 10.78305
```

```
mean(out)
```

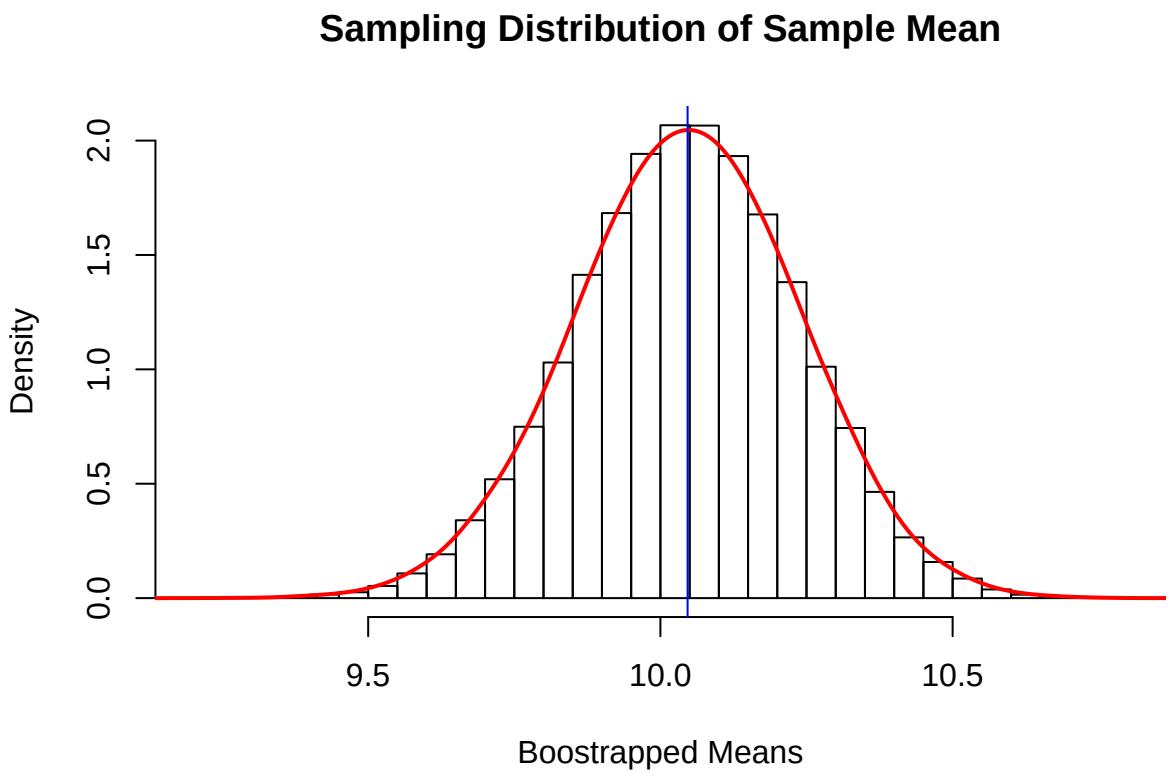
```
## [1] 10.04469
```

```
sd(out)
```

```
## [1] 0.1905054
```

(c)

```r
# Construct a histogram with double the number of bins
hist(x = out, freq = F, xlab = 'Boostrapped Means',
     main = 'Sampling Distribution of Sample Mean',
     breaks=nclass.Sturges(out)*2)
# Add a density line
lines(density(out, adjust=2), col=2, lwd=2)
# Add a verticle line at the centre of the distribution
abline(v = median(out), col = 4)
```

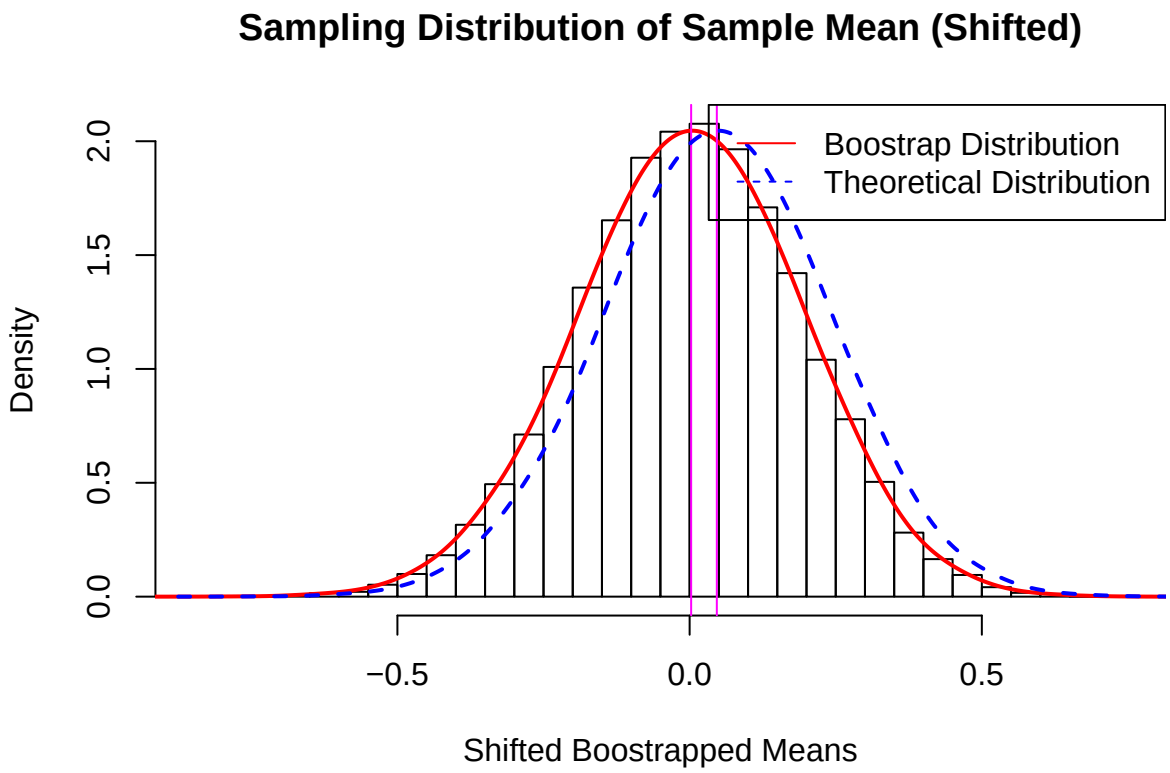


This output looks like a translated normal distribution centered at the true sample mean.

(d)

```r
# Construct a histogram with double the number of bins
hist(x = out-mean(x), freq = F, xlab = 'Shifted Boostrapped Means',
     main = 'Sampling Distribution of Sample Mean (Shifted)',
     breaks=nclass.Sturges(out)*2)
# Add a density line to the histogram
lines(density(out - mean(x), adjust=2), col=2, lwd=2)
# Add a verticle line at the centre of the distribution
abline(v = median(out - mean(x)), col = 6)
# Add the theoretical distribution
lines(density(out - 10, adjust=2), col=4, lwd=2, lty = "dashed")
# Add a verticle line at the centre of the theoretical distribution
abline(v = median(out - 10), col = 6)
# Add a legend
```

```
legend("topright", col = c(2, 4), lty = c(1, 2), legend =
         c('Boostrap Distribution', 'Theoretical Distribution'))
```



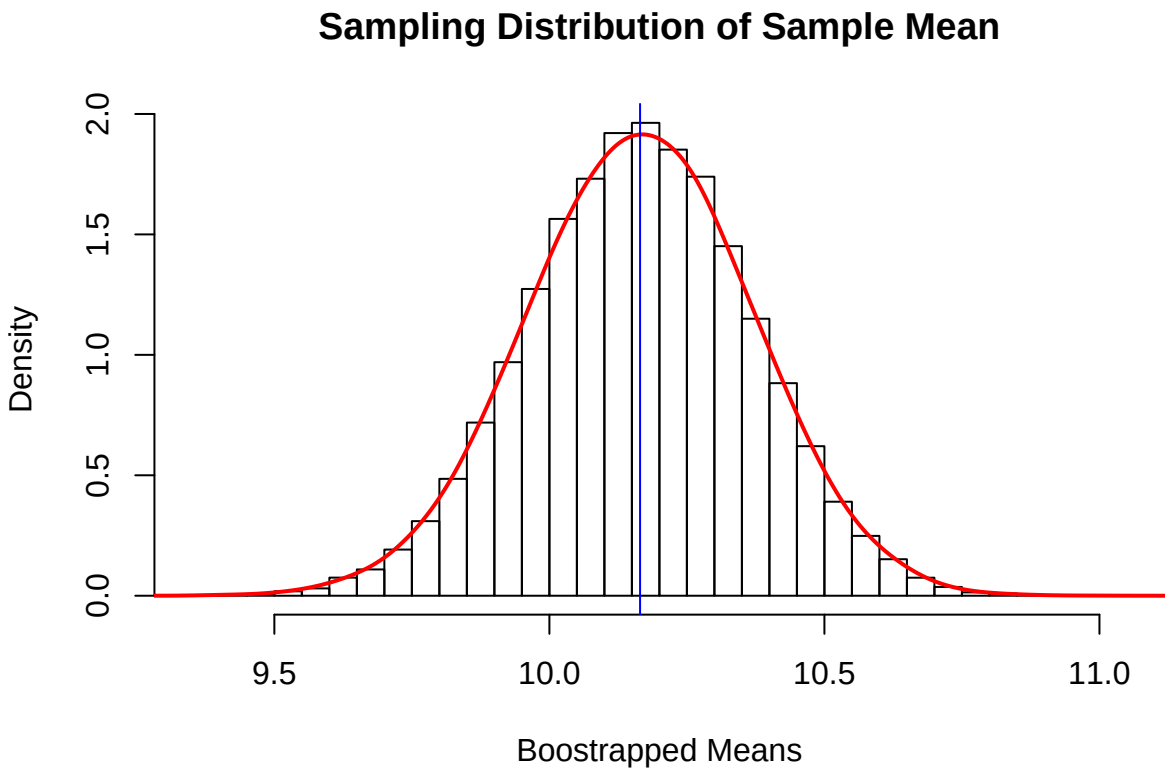
It looks like the boostrap distribution is identical to theoretical distribution except that their centres are shifted (translated). This shift may be called an 'error'. Note that the distributions are equivariant.

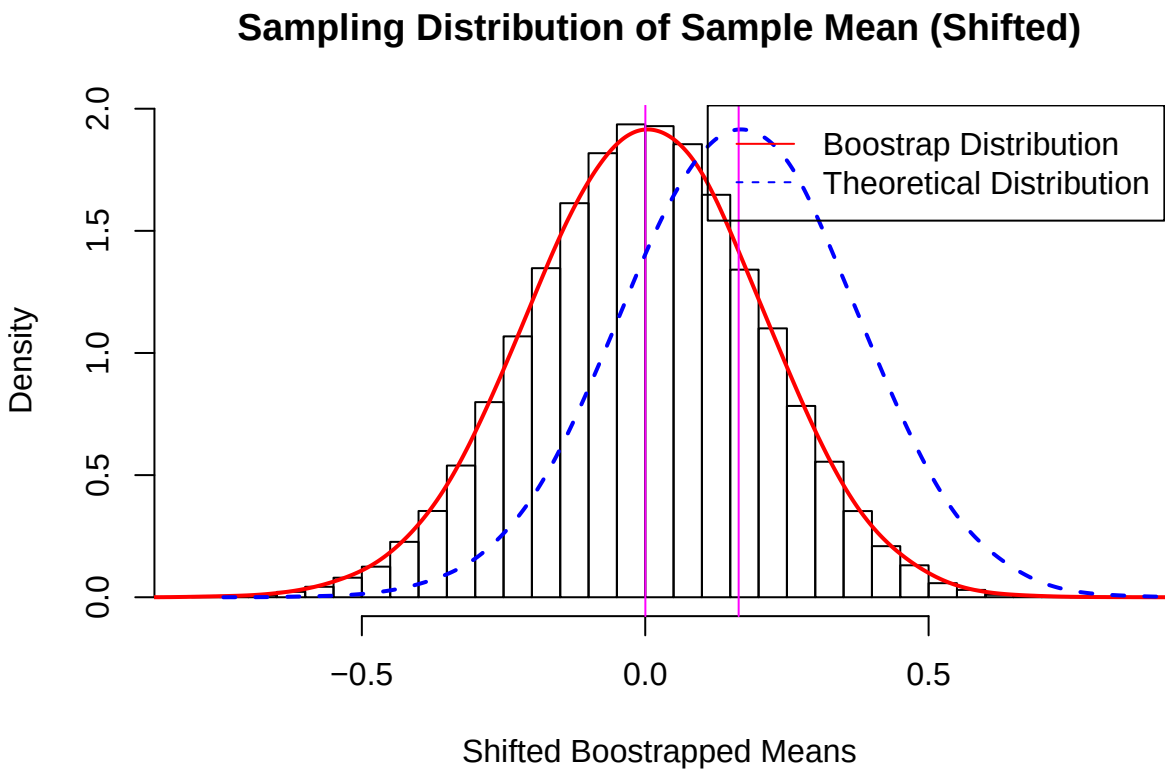
(e)

We will omit the code from the output since it is identical to the last parts. ** Round 2 **

```
## [1] "summary statistics of the sample: "
## [1] "min:"
## [1] 4.162518
## [1] "max:"
## [1] 14.9448
## [1] "mean:"
## [1] 10.16458
## [1] "sd:"
## [1] 2.039159
## [1] "summary statistics of the means (boostrapped data): "
## [1] "min:"
## [1] 9.381324
```

```
## [1] "max:"
## [1] 11.00874
## [1] "mean:"
## [1] 10.16348
## [1] "sd:"
## [1] 0.2026687
```



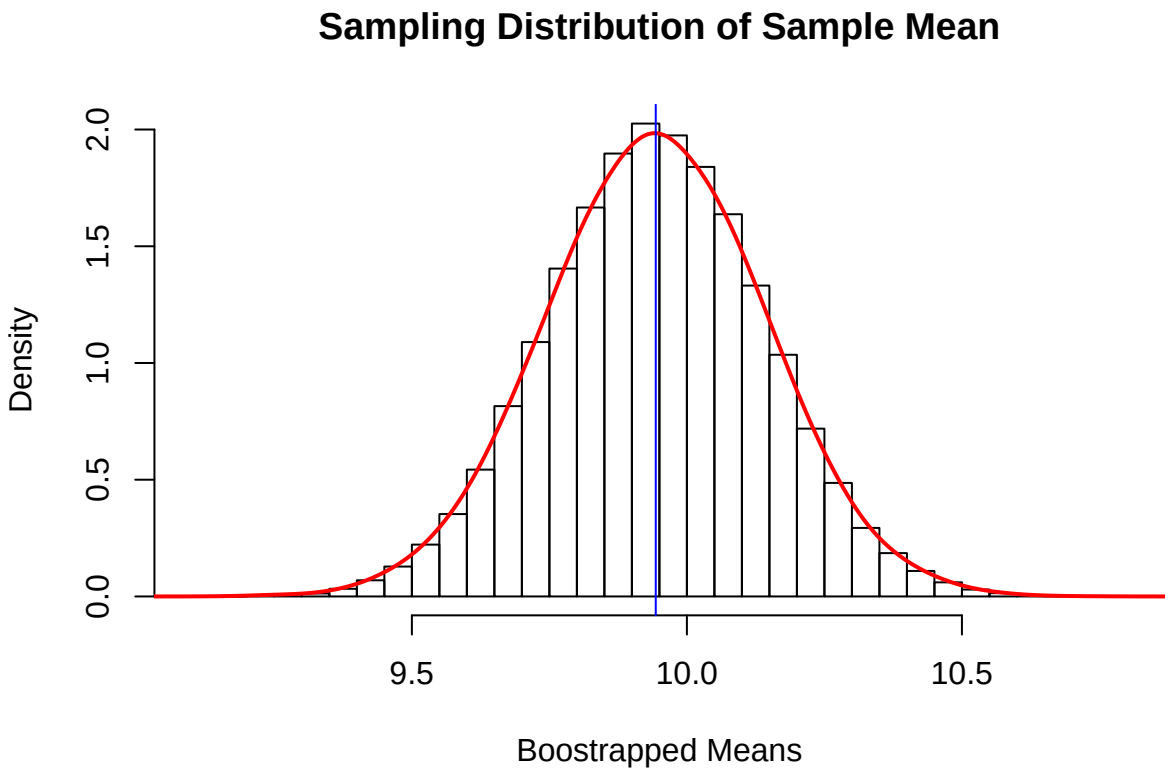


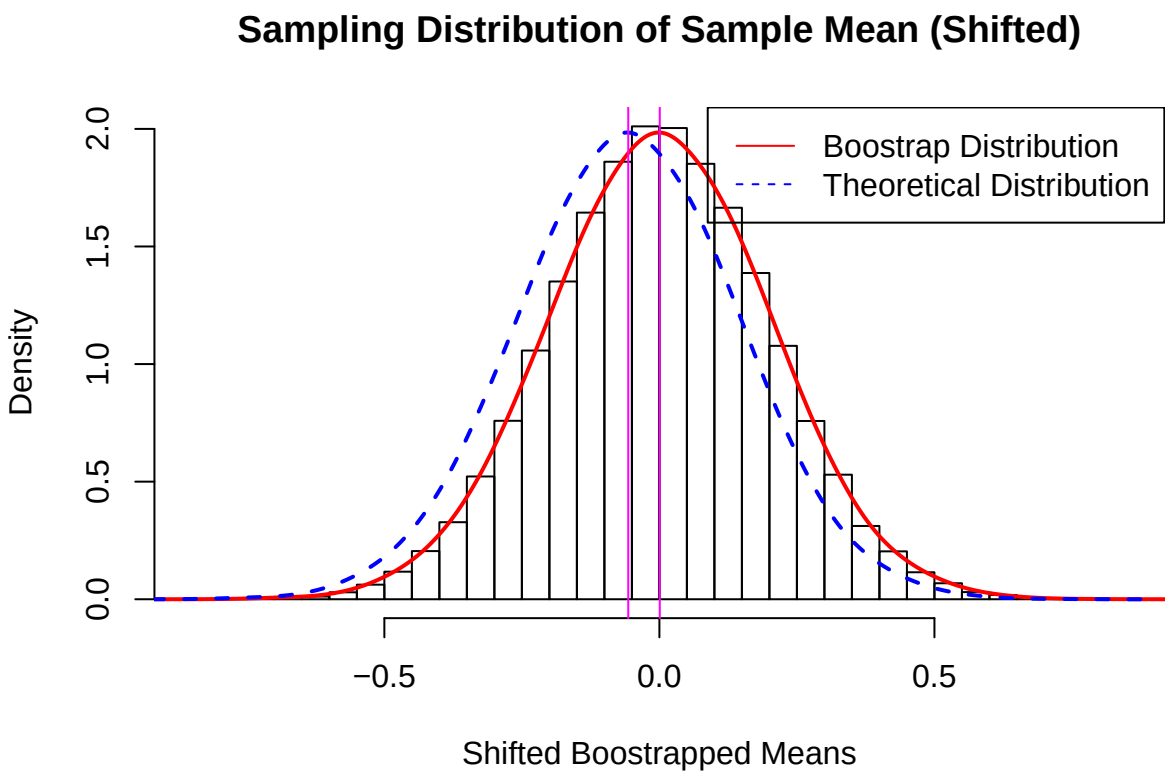
Note in general the results are similar, except that this time, the 'error' is larger.

** Round 3 **

```
## [1] "summary statistics of the sample: "
## [1] "min:"
## [1] 5.251747
## [1] "max:"
## [1] 14.22127
## [1] "mean:"
## [1] 9.94272
## [1] "sd:"
## [1] 1.991693
## [1] "summary statistics of the means (boostrapped data): "
## [1] "min:"
## [1] 9.14236
## [1] "max:"
## [1] 10.75656
## [1] "mean:"
## [1] 9.943207
```

```
## [1] "sd:"
## [1] 0.1979312
```





Again, we get similar results, and this time the 'error' has shrunk.

Question 3

(a)

```
my.obj = function(theta, x) {
  # We allow theta to be a vector.
  # If dim(theta) > 1, we return a vector of dim(theta)
  #   and we re-use the data in x for all theta given
  # Else, we return a scalar
  if (length(theta) > 1) {
    x.mat = matrix(rep(x, times = length(theta)), nrow = length(x))
    theta.mat = matrix(rep(theta, each = length(x)), nrow = length(x))
    result = colSums(abs(x.mat - theta.mat))
  }
  else {
    result = sum(abs(x - theta))
  }
  return(result)
}
```

(b)

```
# define a function that minimizes my.obj with respect to theta
my.minimizer1 = function(x, interval = c(min(x), max(x))) {
```

```
  o = optimize(my.obj, interval = c(min(x), max(x)), x)
  return(o$minimum)
}

# test the function on the given data
data = c(3, 7, 9, 12, 15, 18, 21)
my.minimizer1(data)
```

```
## [1] 12
```

```
median(data)
```

```
## [1] 12
```

Our function works perfectly!

(c)

```
# define a function that minimizes my.obj with respect to theta
my.minimizer2 = function(x, start = mean(x)) {
  o = nlminb(start = start, objective = my.obj, x = x)
  return(o$par)
}

# test the function on the given data
my.minimizer2(data)
```

```
## [1] 12
```

```
median(data)
```

```
## [1] 12
```

This function works just as good too.

(d)

```
# create test data
data = c(1, 3, 7, 9, 12, 15, 18, 21)
# test the functions from b and c
tb1 = my.minimizer1(data, interval = c(min(data) - 2, max(data) + 2))
tb2 = my.minimizer1(data, interval = c(min(data) - 4, max(data) + 4))
tb3 = my.minimizer1(data, interval = c(min(data) - 6, max(data) + 6))
tc1 = my.minimizer2(data, start = min(data))
tc2 = my.minimizer2(data, start = max(data))
tc3 = my.minimizer2(data, start = mean(data)/2)
tb1
```

```
## [1] 10.2539
```

```
tb2
```

```
## [1] 10.2539
```

```
tb3
```

```
## [1] 10.2539
```

```
tc1
```

```
## [1] 10.33333
```

```
tc2
```

```
## [1] 9
```

```
tc3
```

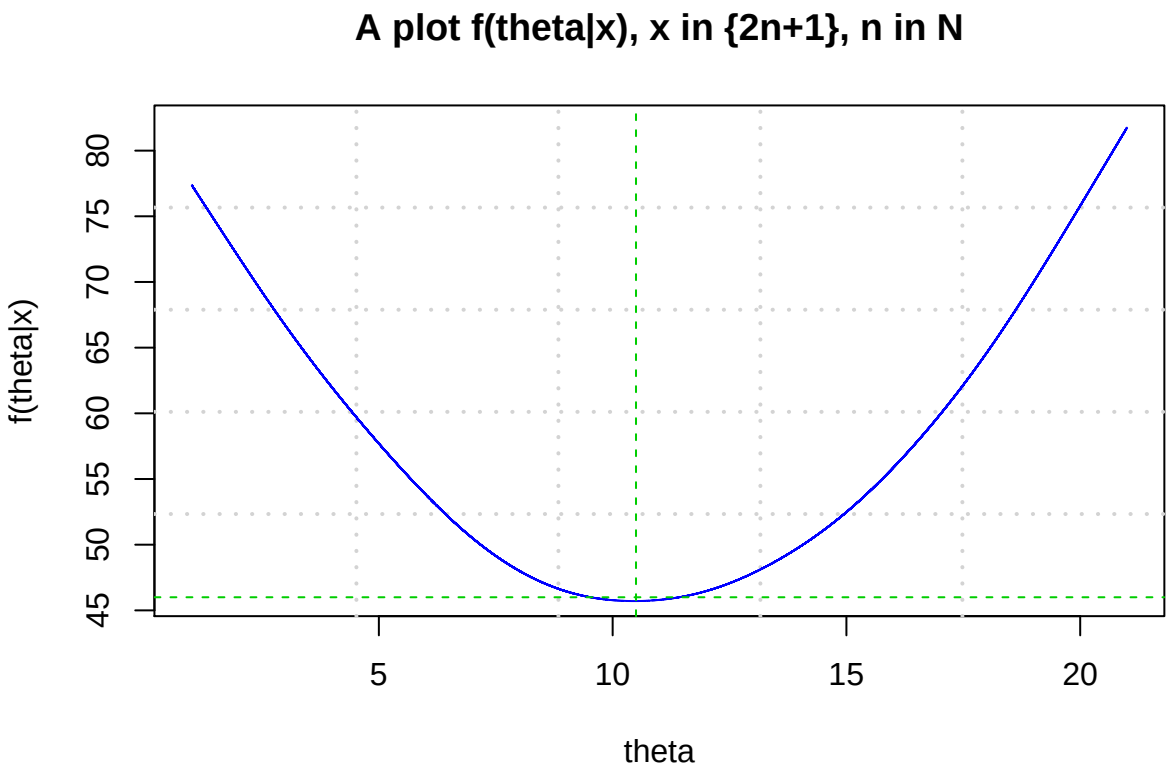
```
## [1] 9.375
```

```
median(data)
```

```
## [1] 10.5
```

It seems that both functions fail to converge to the minimizer.

```
library('MASS')
# evaluate theta on interval [1, 21]
theta = seq(min(data), max(data), length.out = 1e5)
# evaluate f on theta and the given data
f = my.obj(theta, data)
# set up a grid and plotting area
plot(x = theta, y = f, type = 'n', main = 'A plot f(theta|x), x in {2n+1}, n in N',
     xlab = 'theta', ylab = 'f(theta|x)', col = 4)
grid(5, lwd = 2)
# create a smooth plot of the data
sm = smooth.spline(theta, f, spar=1)
# add the smooth plot
lines(sm, col = 4)
abline(v = median(data), col = 3, lty = 2)
abline(h = my.obj(median(data), data), col = 3, lty = 2)
```



We find that indeed, the value of θ that minimizes $f(\theta|x)$ is the median of x. In this case, it is 10.5.

Question 4

We will omit the code from showing on the output due to its length.

(a)

```
mean(huron)
```

```
## [1] 579.3091
```

```
sd(huron)
```

```
## [1] 1.335657
```

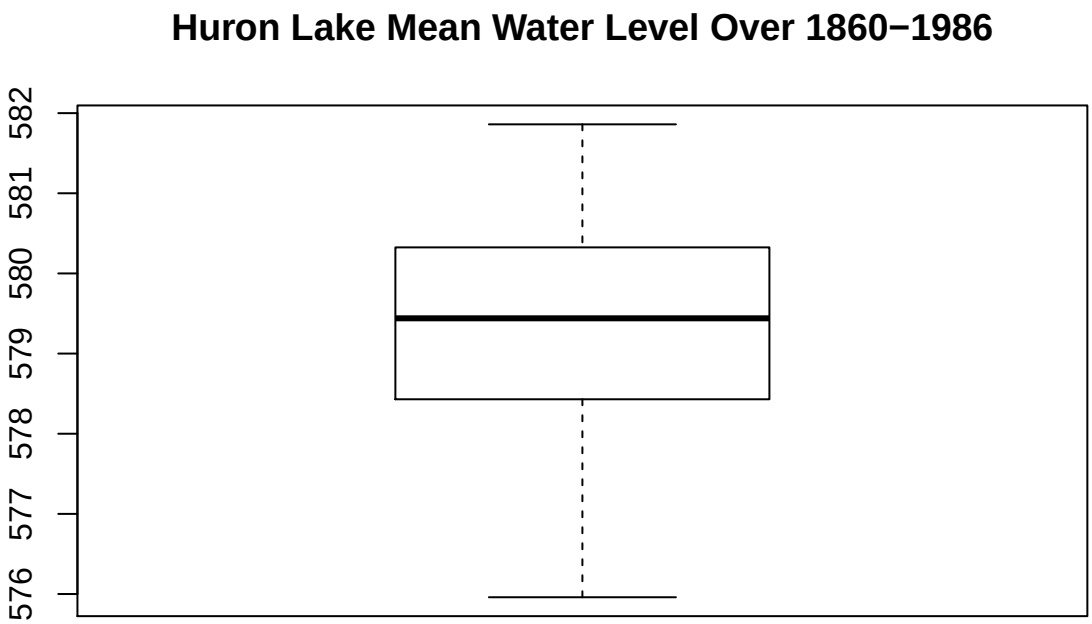
```
min(huron)
```

```
## [1] 575.96
```

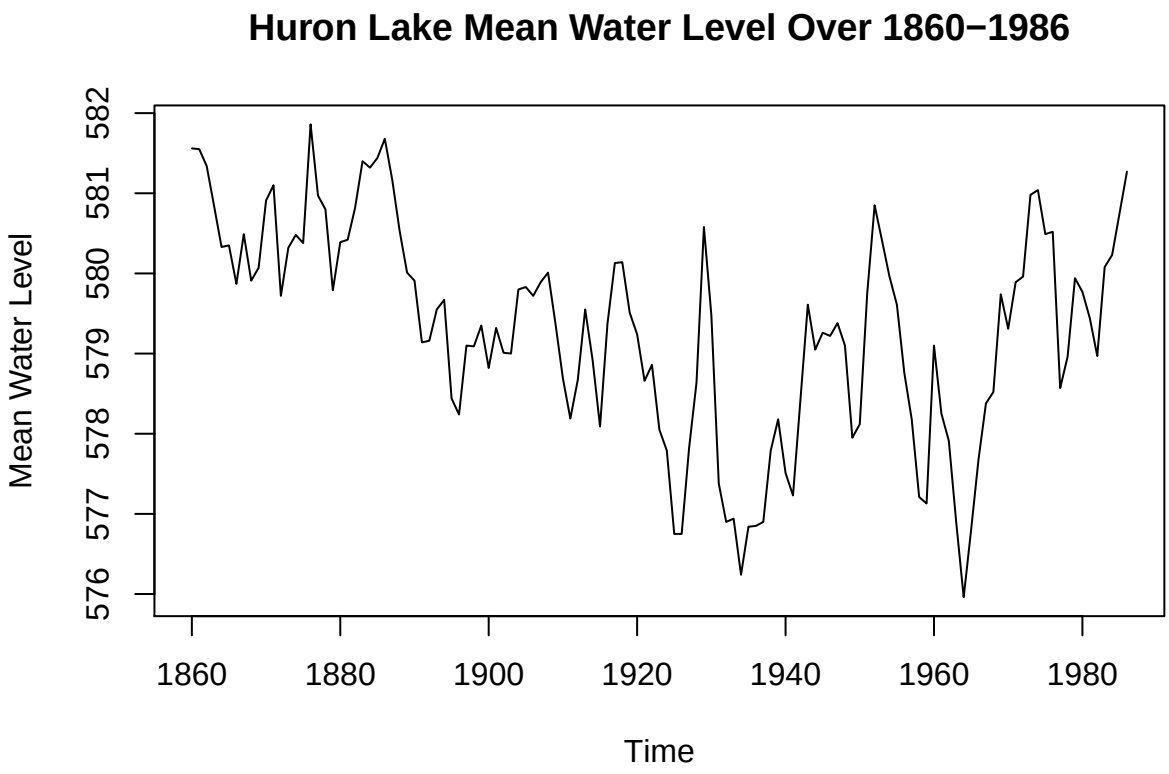
```
max(huron)
```

```
## [1] 581.86
```

```
boxplot(huron, main = "Huron Lake Mean Water Level Over 1860-1986")
```




```
ts.plot(huron, main = "Huron Lake Mean Water Level Over 1860-1986", ylab = "Mean Water Level")
```



(b)

```
# define the given likelihood function
log.likelihood=function(par, x){
  n=length(x)
  v=x[1]^2
  for (i in 2:n)
    v=v+(x[i]-par[1]*x[i-1])^2
  return(v/par[2]+n*log(par[2]))
}
# define huron2
huron2 = huron - mean(huron)
# do optimization
my.par = nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2)
```

```
## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation

## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation

## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation
```

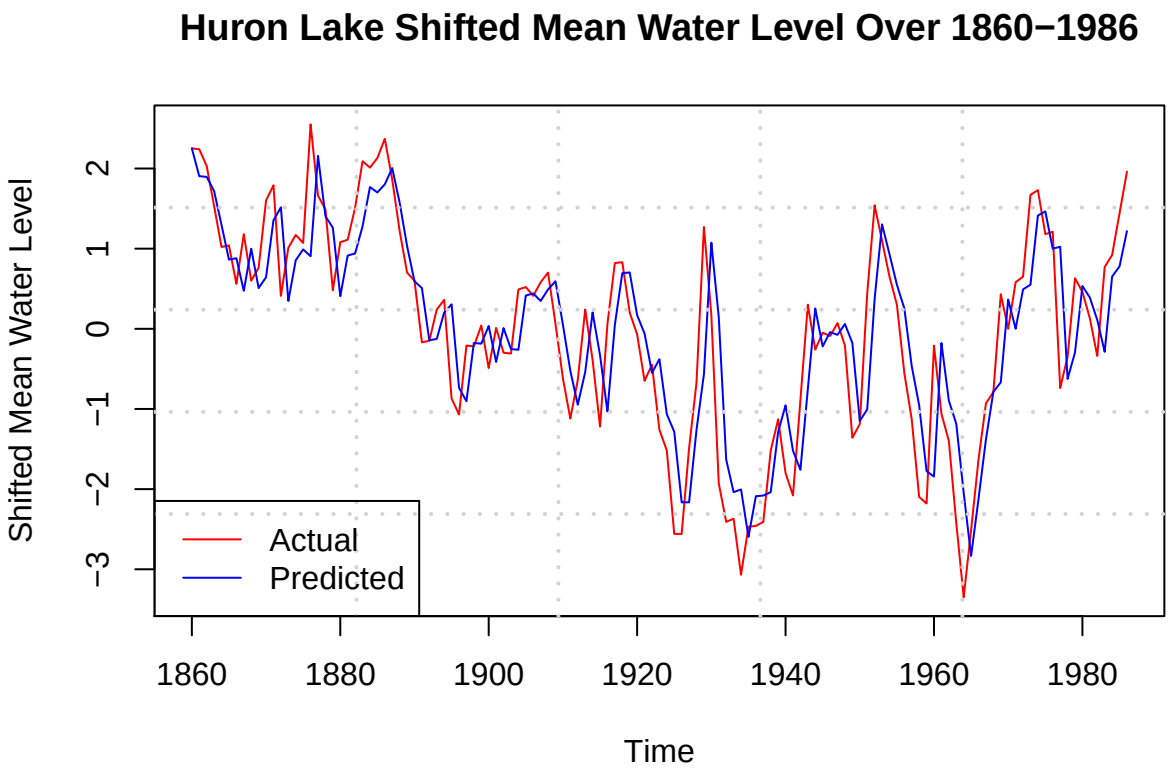
```
## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation

## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation

## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation

## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation

## Warning in log(par[2]): NaNs produced

## Warning in nlminb(start = c(-0.1, 540), objective = log.likelihood, x = huron2):
## NA/NaN function evaluation
```

```r
# extract the parameters
par = my.par$par
# display the parameters
par
```

```
## [1] 0.8459369 0.5250192
```

(c)

```r
# predict the time series data using the first entry in huron2
pred.huron2 = huron2
for (i in 2:length(huron2)) {
  pred.huron2[i] = par[1]* huron2[i-1]
}
# plot the predicted and actual time series
ts.plot(huron2, pred.huron2, col = c('red', 'blue'),
        main = "Huron Lake Shifted Mean Water Level Over 1860-1986",
        ylab = "Shifted Mean Water Level")
# add a legend and a grid
grid(5, lwd = 2)
legend("bottomleft", col = c('red', 'blue'), lty = 1, legend = c("Actual", "Predicted"))
```



Our model captures the fluctuations of the time series data; however, it does not match the original data and we can see it is off by a constant.