HW6_IS457_39

39

10/31/2018

Do not remove any of the comments. These are marked by

HW 6 - Due Wednesday Oct 31, 2018 in moodle and hardcopy in class.

(1). Please upload R code and report to Moodle with filename: HW7_IS457_YourCourseID.

(2). Turn in hard copy of your report in class.

Please ensure that no identifying information (other than your class ID) is on your paper copy.

Part 1: Unfair Dice Simulation

A die is not necessarily fair, in which case the probabilities for 6 sides are different.

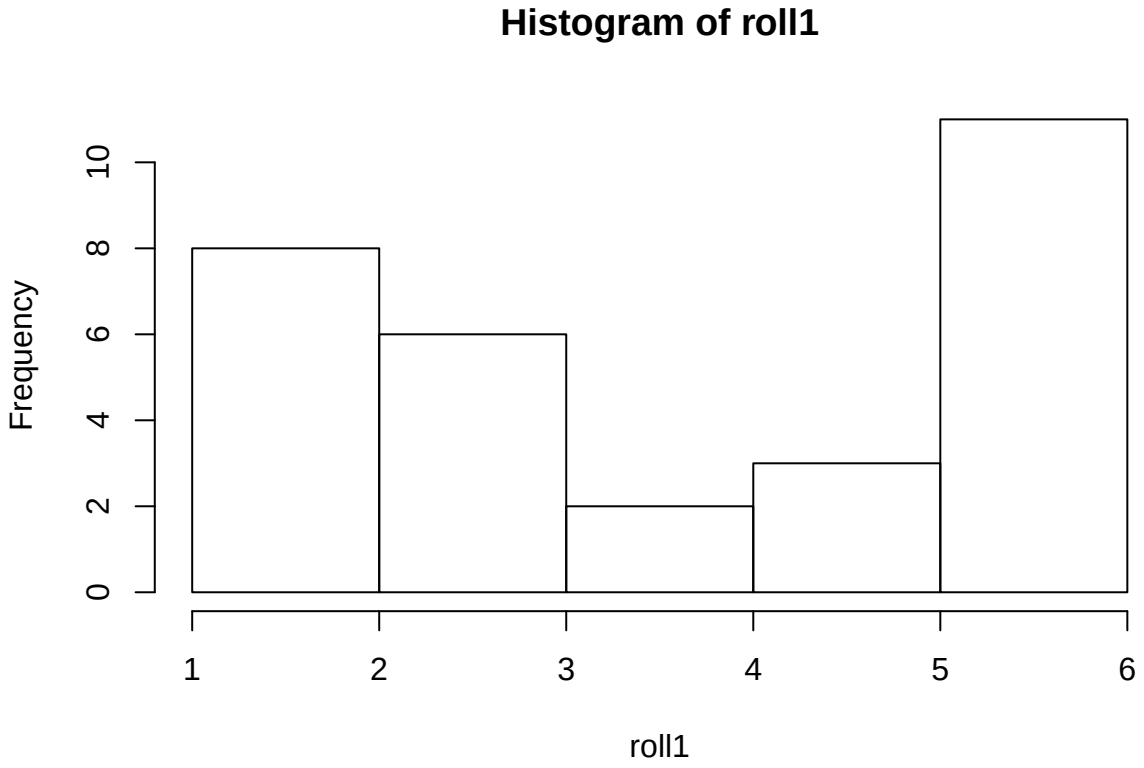
We will look at a way to simulate unfair dice rolls in R

(1) Draw independently from a 6-side die with probability 2/7 for a six and 1/7 for others 30 times,

and save your result in a vector called roll1, make a histogram for the empirical density. (2 pt)

Your code here

```
x1 = c(1:6)
p1 = c(1/7, 1/7, 1/7, 1/7, 1/7, 2/7)
roll1 = sample(x1, 30, replace = TRUE, p1)
help("hist")
hist(roll1)
```

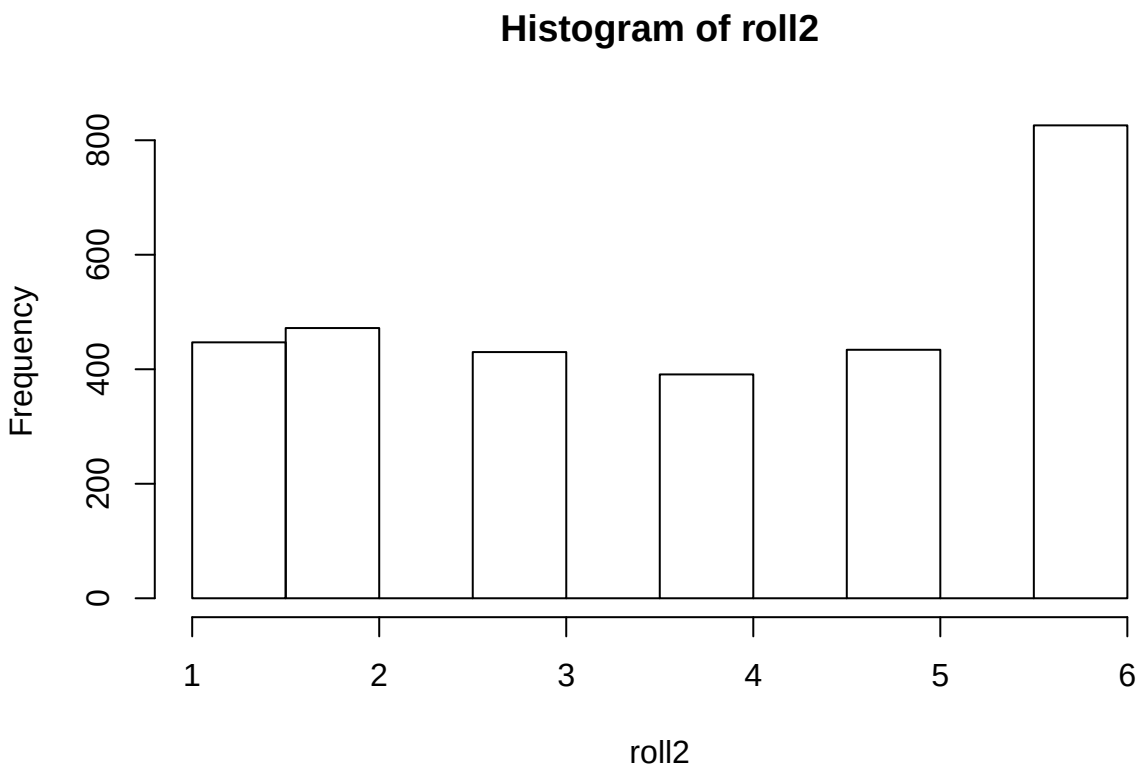


(2) Now, draw independently from a 6-side die with probability 2/7 for a six and 1/7 for others 3000 times,

and save your result in vector roll2, make a histogram for the empirical density. (2 pt)

Your code here

```
roll2 = sample(x1, 3000, replace = TRUE, p1)
hist(roll2, freq = NULL)
```



(3) What do you conclude from comparing these two plots? (2 pts)

Your answer here

```
# With the repeating of experiment increasing, the distribution of empirical density tends to be
# as reasonable as caculated.
```

Part 2: Monte Carlo Simulation

We will use the simulation techniques (Monte Carlo) introduced in class to generate confidence intervals for

our estimates of the distribution mean

(1) As we will generate random numbers, to ensure reproducibility, please set the seed as 457.(1 pt)

NOTE: make sure you run the seed command EVERY time you sample something

Your code here

```
set.seed(457)
```

(2) For this simulation problem, we will sample data from the binomial distribution with parameters n and p.

First, we will estimate an individual experiment.

(a) Generate 100 observations of test data from the binomial distribution, with 20 trials and 0.8 probability and name it test_sample. (1 pt)

Your code here

```
test_sample = rbinom(100, 20, 0.8)
```

(b) What is your estimate of the mean for the test data? call your estimate X_hat. What is the exact mean (use the formula to calculate mean

for a binomial dist)? are they close? what does this say about our random generation?(4 pts)

Your code here

```
mean(test_sample)
```

```
## [1] 16.24
```

Your answer

```
# The estimate of the mean for the test data is 16. The exact mean is 16.24, and they are pretty close.
# This means the output of our random generation distribute well, generally consistent to what we expec
```

(c) What is the 95% confidence interval for X_hat? (2 pts)

Your code here

```
t_mean = mean(test_sample)
t_sd = sd(test_sample)
t_len = length(test_sample)
conf_t = c(t_mean - (1.96 * t_sd)/sqrt(t_len), t_mean + (1.96 * t_sd)/sqrt(t_len))
conf_t
```

```
## [1] 15.89528 16.58472
```

(3) Now use simulation technique to estimate the distribution of X_hat and create confidence intervals for it.

(a) Form a set of X_hat's by repeating B = 1000 times the individual experiment. (2 pts)

HINT: You may want to create a matrix to save those values.

Your code here

```
B = matrix(replicate(1000, rbinom(100, 20, 0.8)), ncol = 1000)
```

(b) Get an estimate for the mean of the X_hat's for each experiment in (3)(a) and save it to a vector X_hat_estimate (length B vector).(1 pt)

Your code here

```
X_hat_estimate = apply(B, 2, mean)
```

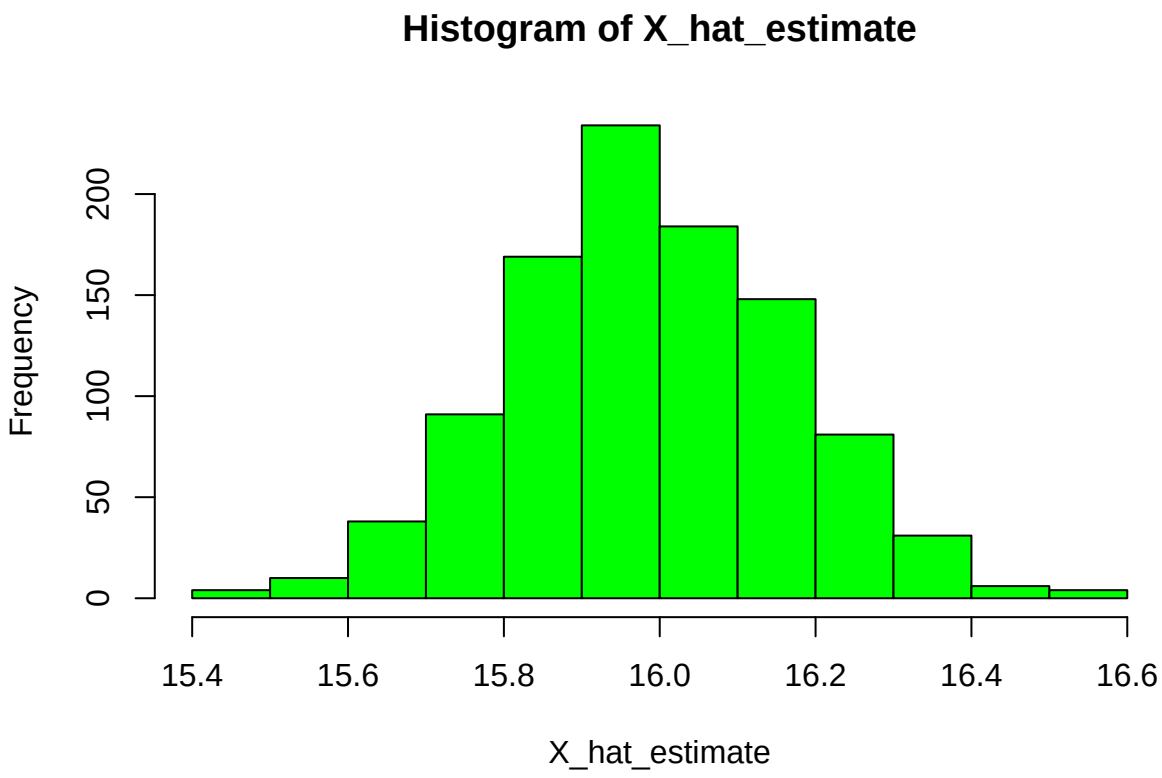
(c) Now use X_hat_estimate to create a "sampling distribution" for X_hat, and create a histogram to show the

distribution. Does the distribution look normal (what are the essential elements of normal dist)? how can you tell?

if yes, what does it say about our random generation? (4 pts)

Your code here

```
hist(X_hat_estimate, col = "green")
```



###

Your answer here

```
# The distribution looks fairly normal because our sampling was random enough. The essential elements o
# distribution contain mean and standard deviation. The mean of X_hat is around 15.9. This result means
# went well and made sense on simulated sampling.
```

(d) Now as we have a simulated sampling distribution of X_hat, we could empirically calculate the standard error using the

X_hat_estimate. What is your 95% confidence interval?(2 pts)

Notice here the standard error is indeed the standard deviation

Your code here

```
mean_X = mean(X_hat_estimate)
sd_X = sd(X_hat_estimate)
n_X = length(X_hat_estimate)
conf_X <- c(mean_X - (1.96 * sd_X) / sqrt(n_X), mean_X + (1.96 * sd_X) / sqrt(n_X))
conf_X
```

```
## [1] 15.9816 16.0038
```

(4) We made some decisions when we used the simulation above that we can now question.

Repeat the above creation of a confidence interval in (3) for a range of settings

(we had our sample size fixed at 100)

and a range of B values (we had B fixed at 1000).

Suppose the sample size varies (100, 200, 300, ... , 1000) and B varies (1000, 2000, . . . , 10000).

You will likely find it useful to write functions to carry out these calculations.

Your final output should be upper and lower pairs for the confidence intervals produced using the bootstrap

method for each value of sample size and B.

(a) Generalize (3) into a function, and vary inputs of sample size and B as we did above. (5 pts)

Your code here

```
Simlt_conf = function(smp_size, smp_B){
  input_X = matrix(replicate(smp_B, rbinom(smp_size, 20, 0.8)), byrow = TRUE, nrow = smp_size)
  input_estimate = apply(input_X, 2, mean)
  mean_input = mean(input_estimate)
  sd_input = sd(input_estimate)
  n_input = length(input_estimate)
  conf_int = c(mean_input - (sd_input * 1.96)/sqrt(n_input), mean_input + (sd_input * 1.96)/sqrt(n_inpu
  return(conf_int)
}

input_limits = c()
for (sample_size in seq(100, 1000, 100)){
  for (sample_B in seq(1000, 10000, 1000)){
    input_limits = rbind(input_limits, c(sample_size, sample_B, Simlt_conf(sample_size, sample_B)))
  }
}
```

(5) Use the function errbar() in Hmisc package.

Plot your confidence interval limits to compare the effect of changing the sample size and

changing the number of simulation replications B (10 pts).

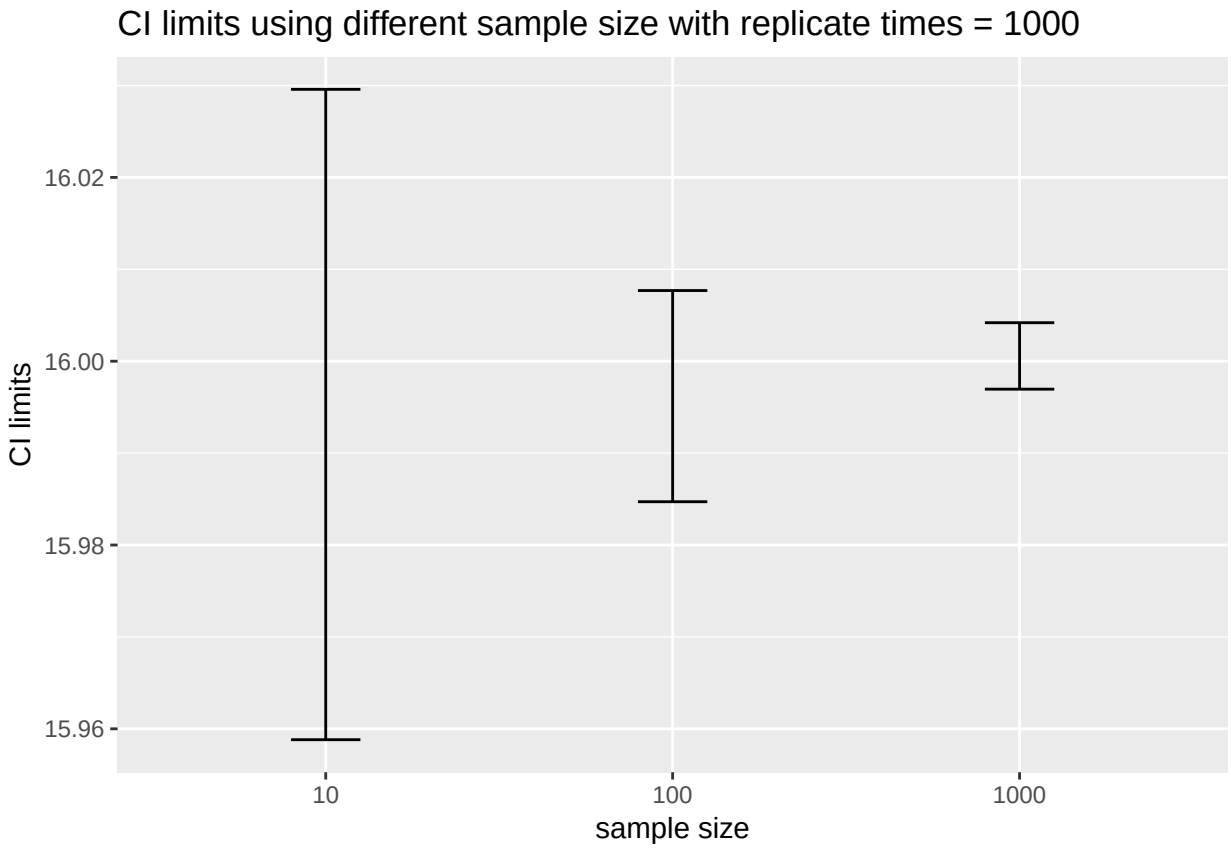
What do you conclude? (4 pts)

Your code here

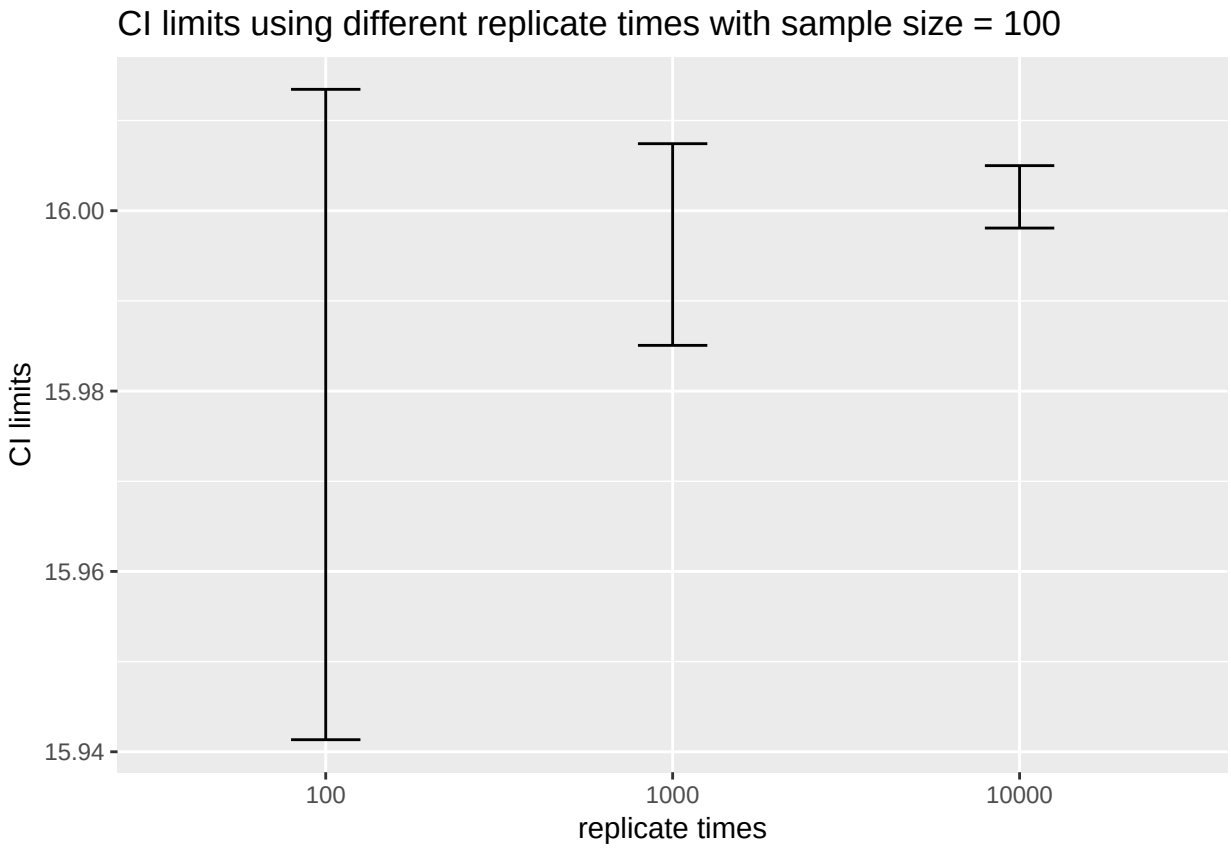
```
require(Hmisc)
```

```
## Loading required package: Hmisc
## Loading required package: lattice
## Loading required package: survival
## Loading required package: Formula
## Loading required package: ggplot2
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
help("errbar")
ss_change = matrix(c(Simlt_conf(10, 1000), Simlt_conf(100, 1000), Simlt_conf(1000, 1000)), nrow = 3, by
df_B1000 = data.frame(size = c("10", "100", "1000"),
                      lower = c(ss_change[1] ,ss_change[2], ss_change[3]), upper = c(ss_change[4] ,ss_c
p1 = ggplot(df_B1000, aes(size)) +
  geom_errorbar(aes(x = size, ymin = lower, ymax = upper), width = .2) +
  xlab("sample size") + ylab("CI limits") +
  ggtitle("CI limits using different sample size with replicate times = 1000")
p1
```




```
rb_change = matrix(c(Simlt_conf(100, 100), Simlt_conf(100, 1000), Simlt_conf(100, 10000)), nrow = 3, by
df_S100 = data.frame(replicate = c("100", "1000", "10000"),
                     lower = c(rb_change[1], rb_change[2], rb_change[3]), upper = c(rb_change[4], rb_c
p2 <- ggplot(df_S100, aes(replicate)) +
  geom_errorbar(aes(x = replicate, ymin = lower, ymax = upper), width = .2) +
  xlab("replicate times") + ylab("CI limits") +
  ggtitle("CI limits using different replicate times with sample size = 100")
p2
```



Your answer here

```
# In conclusion, with the sample size or simulation replication increasing, the CI limits tend to
# getting close to the mean value as well as the CI range getting shorter.
```