Lab: Introduction to R

Hanh Nguyen

Spring 2020

In this lab we will both practice R syntax/coding and introduce RMarkdown for presenting lab reports.

#### R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

## Task 1: A Basic Simulation Event

Much of our applied probability computing work in this class will be simulating events. This means that we generate an event at random. The R function for simulating random numbers is sample; check out the help screen.

help(sample)

## starting httpd help server ... done

### Code set-up

Let us try simulating a die roll: The parameter replace = TRUE is important here as we are rolling the die over and over again, not drawing marbles out of a bag. Here is how to roll a 6-sided die five times in R, and then compute the average of the rolls. Try running it!

x = 1:6 # sides of the die  
x <- 1:6  
roll = sample(x, 5, replace = TRUE) # tell R how many sides (x) and how many rolls (5)  
  
mean(roll) # average of the 5 rolls

## [1] 3.6

### The problem

A tetrahedron die is a four-sided die with labels {1, 2, 3, 4}. Have R make 10 rolls of the tetrahedron die and compute the average. (Can think of this as rolling 10 different tetrahedron dice as well.) Keep the output in this RMarkdown file for grading purposes.

# [Place code here]  
x = 1:4   
tet\_roll = sample(x, 10, replace = TRUE)   
mean(tet\_roll)

## [1] 2.4

### Question:

What value do you expect to get for the average of the 10 rolls?

*2.5*

## Task 2: Playing with for-loops

For-loops are central to the simulation studies we will be performing in this class. In these experiments, simulation tasks are repeated over and over again. The for-loop can easily perform this replication for us. The trick is appropriately storing your results for analysis. The syntax for a for-loop in R is for(var in seq){task}, read “for a given variable in a specified sequence.” The for-loop steps that variable through the sequence and performs the task each time.

### Code set-up

Let us apply a for-loop for simulating a 6-sided die roll. That is, repeat 1000 times the experiment of rolling a 6-sided die five times and computing the average.

S = 1000 # number of simulation experiments performed  
# store results in a (1000-dimensional) vector called rolls.avgs  
rolls.avgs = vector(length=S) # setting up the variable rolls.avgs to store the average roll for each experiment  
# this for-loop steps the variable simnum through the sequence 1 to 1000,  
# repeating 1000 times the die rolling tasks inside the curly brackets {...}.  
for(simnum in 1:S){  
 # Use our die rolling code from Task 1!  
 x = 1:6 # sides of the die  
 roll = sample(x, 5, replace = TRUE) # simulate a die roll  
 rolls.avgs[simnum] = mean(roll) # store the average roll  
}  
# take a look at the first 6 simulation results  
head(rolls.avgs)

## [1] 4.6 3.0 2.8 2.4 5.2 4.0

# compute the mean of the 1000 experiments  
mean(rolls.avgs)

## [1] 3.46

### The problem

Repeat 1000 times the experiment you performed in Task 1, that is rolling a tetrahedron die 10 times and computing the average. Report the average and standard deviation of the 1000 experiments. The standard deviation function in R is sd(x).

# [Place code here]  
S = 1000 #number of simulation experiments performed   
rolls.avgs = vector(length = S) #store results in this vector  
  
#for-loop to repeat rolling 1000 times   
for (simnum in 1:S){  
 #Use a tetrahedron die   
 x = 1:4 #a tetrahedron die has 4 sides  
 tet\_roll = sample(x, 10, replace = TRUE) #simulate a die roll  
 rolls.avgs[simnum] = mean(tet\_roll) #store the average roll  
}  
  
head(rolls.avgs)

## [1] 3.1 2.3 2.6 2.6 2.3 2.6

mean(rolls.avgs)

## [1] 2.4917

### Questions:

* Is the mean closer to the value you would expect than the average you had in Task 1? Why?

*The mean of the 1000 simulations should be closer to the expected value (2.5) compared to the average obtained in Task 1, because the mean of 1000 experiments is more likely to reflect the true underlying probability distribution compared to a single experiment. By repeating the experiment 1000 times and computing the average of each experiment, we obtain a more accurate estimate of the expected value (2.5) because it accounts for the natural variability that occurs due to the randomness of the die rolls*

* How do you interpret the standard deviation in this problem? *The standard deviation in this problem represents the variability of the average roll obtained from the 1000 simulations. It gives us a measure of the spread or dispersion of the 1000 results around the mean.A smaller standard deviation indicates that the results are more tightly clustered around the mean, while a larger standard deviation indicates that the results are more spread out. In this problem, the standard deviation can be interpreted as the average deviation from the mean value of the average rolls over the 1000 simulations.*

## Task 3: Presenting tables in RMarkdown

Let us present a table of our die rolls. We will use xtable and pander R packages. Make sure to install the pander package prior to running the code chunk. In this task, we will also try the replicate() function in R to replace the for-loop.

Have R make 5 rolls of the tetrahedron die and repeat that 4 times. Present the results in a table.

#### R Markdown

The exact code is provided for you below. In this way you can cut-and-paste this code for table-making in future labs. Three parameters were added to the code chunk: The echo = FALSE parameter was added to prevent printing of the R code that generated the table. The results='asis' parameter was added to have R present the results as is for the table generation. The warning=FALSE parameter suppresses warning messages from R that are often presented when loading packages.

As an aside, a fourth common parameter is include=FALSE, which prevents R from printing output when running the code chunk.

Replicate 5 rolls of a tetrahedron die two times.

| Replicate 1 | Replicate 2 | Replicate 3 | Replicate 4 |
| --- | --- | --- | --- |
| 1 | 3 | 1 | 4 |
| 4 | 4 | 1 | 1 |
| 2 | 3 | 4 | 1 |
| 4 | 2 | 4 | 4 |
| 1 | 4 | 1 | 2 |

### Question:

What do you observe across the replicates?

*I observe that the results across the replicates are different, as each time the code is run, a different set of random numbers will be generated. This is because the random number generator is initialized with the current system time, and the results will be different each time the code is run. Hence, we can observe different outcomes for the rolls of the tetrahedron die across the replicates.*

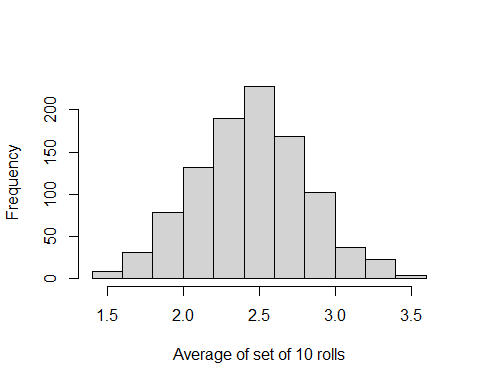
## Task 4: Presenting graphs in RMarkdown

Graphs are easy to display in an RMarkdown file.

### Code set-up

Let us draw a histogram of our 1000 die rolls from earlier.

hist(rolls.avgs, main="", xlab="Average of set of 10 rolls")



### The problem

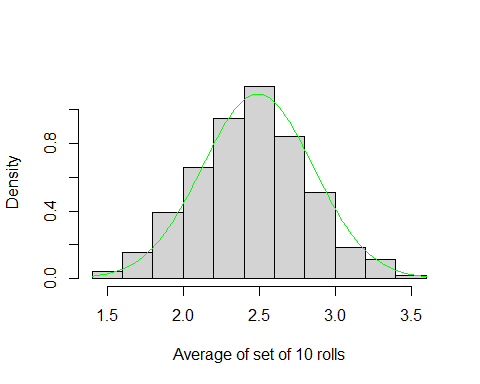
Let us add a normal approximation (bell curve) to the histogram. We will cover the normal distribution later in the course. But hopefully you recall it from your Statistics course! To add a density curve to the plot, need to change the y-axis to a ‘density’ scale. This is done by setting the parameter prob = TRUE. The curve function adds a curve to the plot. We will use a normal distribution with mean and standard deviation set at the values obtained in Task 2. Here is the code

hist(rolls.avgs, prob = T, main=““, xlab=”Average of set of 10 rolls”) # histogram

curve(dnorm(x, mean=mean(rolls.avgs), sd=sd(rolls.avgs)), add=TRUE, col=“green”) # normal approximation

Add these to the code chunk to present a histogram with a normal approximation

# [Place code here]  
hist(rolls.avgs, prob = T, main="", xlab="Average of set of 10 rolls") # histogram  
   
curve(dnorm(x, mean=mean(rolls.avgs), sd=sd(rolls.avgs)), add=TRUE, col="green") # normal approximation



### Questions:

* Interpret the histogram–shape, skew, spread, center.

*From the histogram, we can see it has the bell shape.This shape occurs when the data being plotted is approximately normally distributed. This histogram is considered to be symmetrical since the left and right tails are equal of length. The spread of the histogram can be measured by the standard deviation, which is indicated by the width of the bell. The center is indicated by the mean of the average values, which is around 2.5*

* Does this follow what you would expect to see?

*This histogram shows what I would expect to see since the mean is around 2.5. The normal approximation curve should closely match the shape of the histogram, indicating a good fit of the data to a normal distribution. Additionally, it is expected to see a symmetrical skew, a spread that can be measured by the standard deviation, and a center that is represented by the mean.*

## Task 5: Boolean expressions

Another useful task is making logical statements in R.

### Code set-up

Let us first make 10 rolls of a die and see how often a 6 is rolled.

x = 1:6 # 6-sided die  
rolls = sample(x, 10, replace = TRUE) # roll the die five times  
# Boolean expression: how often is the roll EXACTLY 6, use double-equals sign  
sum(rolls == 6)

## [1] 2

Now repeat 1000 times the experiment of rolling a die 10 times as in Task 2. We will see how many times a six occurs at least once out of ten rolls across all these experiments. The code for this counting exercise is sum(roll==6)>0 since a “success” is an experiment where the total number of sixes showing on ten rolls is more than zero!

six = 0 # start a counter for number of times at least one six shows in 5 rolls  
S = 1000 # number of experiments  
# for-loop to repeat die rolling experiment 1000 times  
for(simnum in 1:S){  
 x = 1:6 # 6-sided die  
 roll = sample(x, 10, replace = TRUE) # roll the die five times  
 # two ways to count: with an if-then statement, or more elegantly with a Boolean computation  
 #if(sum(roll==2) > 0){st = st + 1} # if-then statement  
 six = six + (sum(roll==6)>0) # Boolean computation: add one to the counter only if at least one 6 shows.  
}  
six

## [1] 842

### The problems

#### First problem

How often in 5 rolls of a tetrahedron die is a two rolled?

# [Place code here]  
x = 1:4 # 4-sided die  
rolls = sample(x, 5, replace = TRUE)  
sum(rolls == 2)

## [1] 2

### Questions:

* Run the code multiple times. What values do you get?

*After running the code multiple times, I see that the results are usually 1 or 2.*

* Are the values different? Is that what you expect?

*The values are different after each time. This is what I expect since we roll the die randomly and all sides are equally likely*

#### Second problem

This is heading towards a probability calculation. Roll the tetrahedron die 5 times and repeat this experiment 1000 times as in Task 2. Report the *proportion* of 1000 simulations where a two occurred. (This derives from Dobrow problem 1.44: Probability of rolling at least one 2 in five rolls of a tetrahedron die.)

##### Dobrow problem 1.30: Exact answer is 0.7627

# [Place code here]  
two = 0 #counter for number of times where a two occurred   
S = 1000 #number of experiments   
#for-loop to repeat die rolling experiment 1000 times   
for(simnum in 1:S){  
 x = 1:4 #tetrahedron has 4 sides   
 roll = sample(x, 5, replace = TRUE) #roll the die five times   
 two = two + (sum(roll == 2) > 0)  
}  
two

## [1] 777

prop\_of\_two = two/S   
prop\_of\_two

## [1] 0.777

### Questions:

* Is the value you get close to the truth (0.7627)?

*Yes, the calculated proportion I get is pretty close to 0.7627*

* How can we modify the simulation experiment to get a value even close to the truth?

*To get a value even closer to the truth, we can increase the number of simulations (S) and/or reduce the variability of the ‘sample’ function. Using the ‘set.seed’ function before running the simulation can help reduce the variability of the sample.*

## Task 6: Finalize your output for submitting to Blackboard

As we noted, you can run each code chunk using the “play” button on the top right corner of the chunk. You may also run individual lines of code in the RStudio console for debugging purposes.

To run the whole document and preview the Word document, press the “Knit” button in the menu bar underneath the tabs. This should present a preview of the Word file and save a .docx file in your working directory.

Alternatively, you may render the Word file in the R console using the render command.

* Save your file as a .rmd file to your desired working directory.
* Then in the R console, place the following:
* library(rmarkdown)  
    
   render("filename.rmd")

This will save the .docx file of the report to your working directory.