Lab 1: Introduction to R

Justin Pelak

"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.

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
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
```

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.

```
dice=c(1,2,3,4)
mean(sample(dice, 10, replace=TRUE))
```

```
## [1] 2
```

Question:

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

I would expect around 2.5 because there is a one forth chance of each side.

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
```

```r
# 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] 3.2 4.2 3.4 4.2 3.2 4.0
```

```r
## [1] 4.6 4.4 3.4 2.8 3.6 3.6
# compute the mean of the 1000 experiments
mean(rolls.avgs)
```

```
## [1] 3.4912
```

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).

```r
n=1000
dice=c(1,2,3,4)
rolls.avgs = vector(length=n)

for(simnum in 1:n) {
  roll = sample(dice, 10, replace=TRUE)
  rolls.avgs[simnum] = mean(roll)
}
mean(rolls.avgs)
```

```
## [1] 2.5168
```

```r
sd(rolls.avgs)
```

```
## [1] 0.3519682
```

Questions:

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

The mean does appear closer to the value we would expect, which makes sense because it had more iterations to allow the mean to get closer.

- How do you interpret the standard deviation in this problem?

If we were to roll a 4 sided dice 10 times, over 60% of those times the mean of the rolls would be within 2.5 +- ~0.3

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 addedto 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.

```r
# we will create a table using xtable and pander
library(knitr)
library(xtable)
library(pander)
# output desired summary statistics
# formatC used so integer dice tosses do not have a decimal place in the figure!

R=replicate(4, sample(1:4, 5, replace = TRUE))


table=xtable(R, caption="Replicate 5 rolls of a tetrahedron die two times", align = "|2|r|r|r|r|")
```

```
## Warning in .alignStringToVector(value): Nonstandard alignments in align string
```

```r
names(table) <- c('Replicate 1', 'Replicate 2', 'Replicate 3', 'Replicate 4')
pander(table)
```

Table 1: Replicate 5 rolls of a tetrahedron die two times

Replicate 1	Replicate 2	Replicate 3	Replicate 4
3	3	1	2
3	1	4	2
2	1	3	3
2	4	2	1
2	2	2	2

Question:

What do you observe across the replicates?

It looks like the replicate is displaying in a table the value of each roll in a sample of 5, 4 times

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.

```r
S = 1000 # number of simulation experiments performed
# store results in a (1000-dimensional) vector called rolls.avgs
```

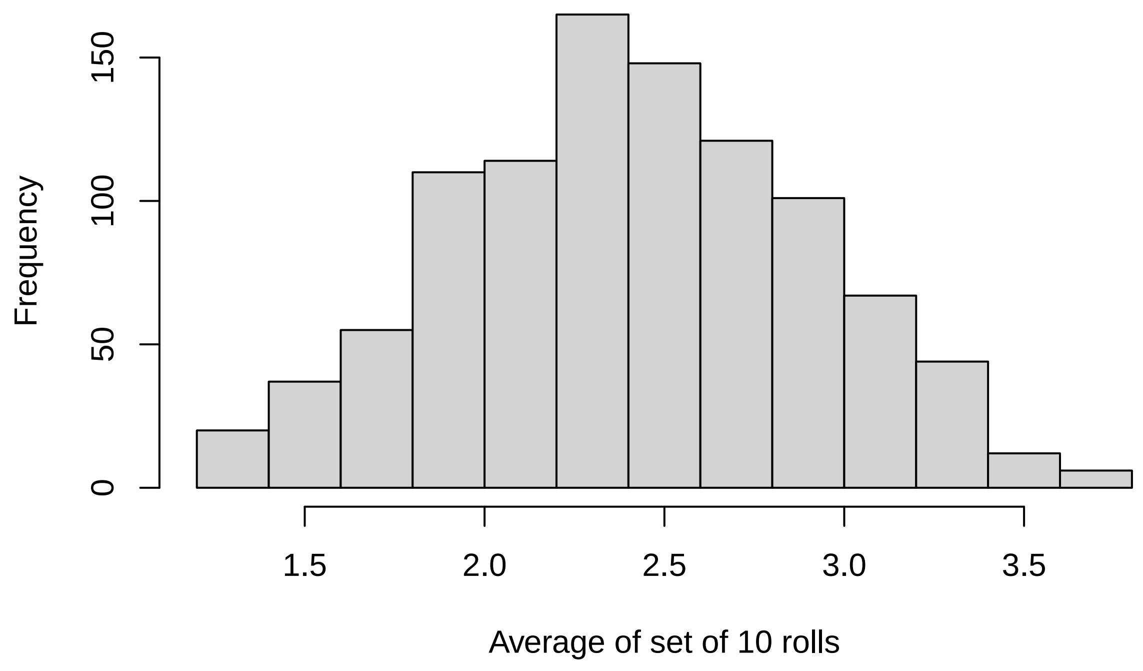
```r
rolls.avgs = vector(length=S)# setting up the variable rolls.avgs to store the average roll for each ex
# this for-loop steps the variable simnum through the sequence 1 to 1000,
# repeating 1000 times the die rolling tasks inside the curly brackets {...}.

for(simnum in 1:S){
# Use our die rolling code from Task 1!
x = 1:4 # 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] 2.8 2.2 2.0 2.4 2.4 2.6
```

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

```
## [1] 2.5092
```

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



The problem

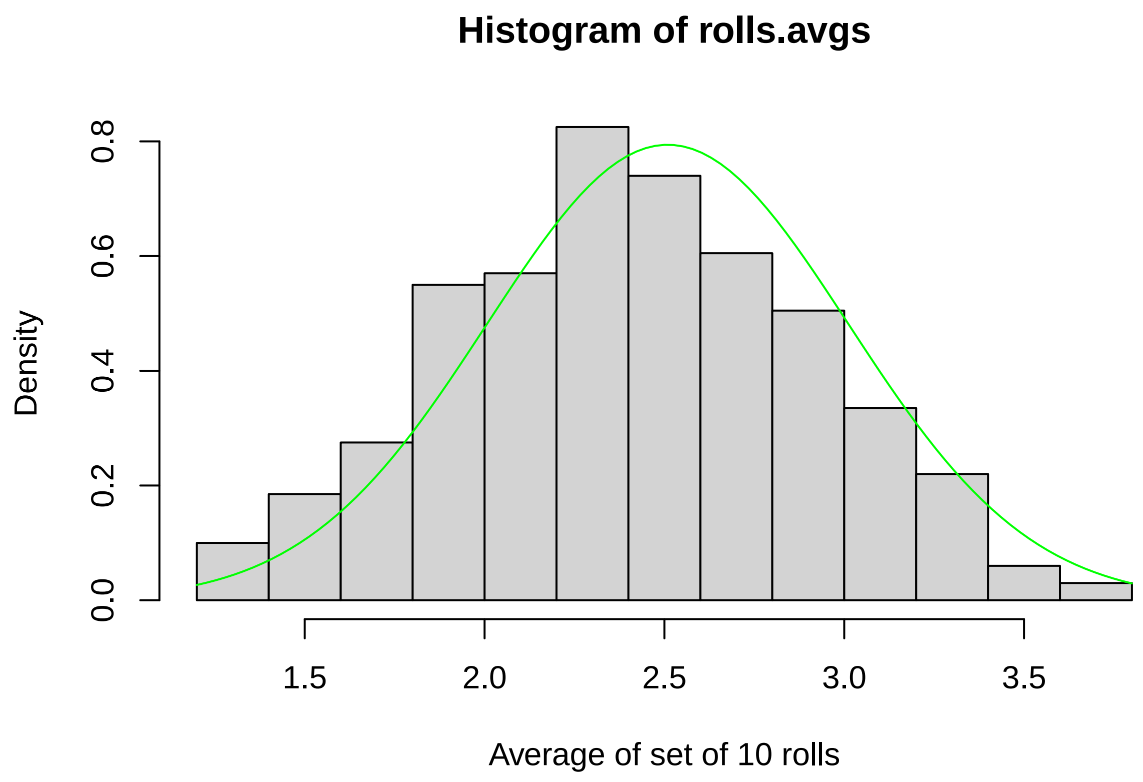
Let us add a normal approximation (bell curve) to the histogram. We will cover the normal distribution later in the course. But hopefully you recall it from your Statistics course! To add a density curve to the plot, need to change the y-axis to a 'density' scale. This is done by setting the parameter prob = TRUE.

The curve function addes 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 approximatio
```

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

```
hist(rolls.avgs, prob=T, xlab="Average of set of 10 rolls")
curve(dnorm(x,mean=mean(rolls.avgs), sd=sd(rolls.avgs)), add=TRUE, col="green")
```



Questions:

- Interpret the histogram–shape, skew, spread, center. *The histogram seems to be normally distributed and centered around 2.5, as expected. The spread also seems to be as expected, as the histogram isn't super wide or super narrow*
- Does this follow what you would expect to see?

Given a fair 4-sided dice, this is what we would expect to see

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.

```r
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] 1
```

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!

```r
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
}
six
```

```
## [1] 829
```

The problems

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

```r
two = 0
dice=1:4
rolls = sample(dice, 5, replace=T)
sum(rolls==2)
```

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

Questions:

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

Pretty consistently, I get 0, 1, or 2

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

With fair dice, occasionally getting 0 or 2+ rolls can be expected, and considering I've gotten 1 more than any in 5 rolls, it does seem fair

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

```r
two = 0
n = 1000
dice = 1:4

for(simnum in 1:n){
  roll = sample(dice, 5, replace = TRUE)
  two = two + (sum(roll==2)>0)
}
two
```

```
## [1] 768
```

Questions:

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

The value is close to the truth

- How can we modify the simulation experiement to get a value even close to the truth?

We can get even closer by creating an even larger sample size or by adding more repetitions to our experiment

Task 6: Finalize your output for submitting to Canvas

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.