Lab 4 – Discrete Probability Simulations

This lab contains some numbered questions. You are required to submit:

1. One pdf document (Lab 4.pdf) that contains:
   - your written answers to the question
   - the commands you used to answer the question, when asked

2. One R script (Lab 4.R) that contains all the code used to produce your answers. The script must run without errors.

Submit your files to the Lab 4 folder in Learning Hub by 11:59pm next school day.

Lab Objectives

In this lab, we are going to simulate random experiments and estimate the probability of an event E based on the results. If we simulate an experiment m times and event E occurs k times (k "successes") then we estimate the probability to be:

$$P(E) \approx \frac{k}{m}$$

We will simulate our random experiments using the built-in function **`sample()`**.

```
sample(x, size, replace = FALSE, prob = NULL)
```

Arguments

x	Either a vector of one or more elements from which to choose, or a positive integer.
size	a non-negative integer giving the number of items to choose.
replace	Should sampling be with replacement?
prob	A vector of probability weights for obtaining the elements of the vector being sampled.

For instance, the following command simulates rolling 5 dice where the dice are "loaded" in favor of getting a 6.

```
sample(c(1, 2, 3, 4, 5, 6),
       5,
       replace=TRUE,
       prob=c(0.1, 0.1, 0.1, 0.1, 0.1, 0.5))
```

Experiment 1: Flipping a Fair Coin ($n = 1$)

Our first experiment will simulate flipping one fair coin. To simulate the single coin-flip, run the following in the console:

```
> sample(c("H", "T"), 1)
```

This tells R to select 1 item from the set $\{H, T\}$. The output is either:

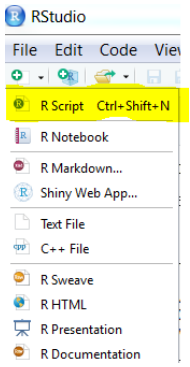
[1] "T"
or
[1] "H"

1. Record a one-line R command that achieves the same thing as running `sample(c("H", "T"), 1)` ten times in the console. Screen-capture the console outputs and add to your document.

Writing Functions in R

Our coin-flipping experiment was a single line that we could easily run in the console. However, for more complicated programs, we will want to write functions and scripts.

Open a new script in R: →



We will define a function called `FlipOnce()` to simulate a single coin flip and return either heads or tails. Type the following into the script window:

```
FlipOnce <-function(){
  result <- sample(c("H", "T"), 1)
  return(result)
}
```

Save your script as Lab04.R in an appropriate folder.

Check the "Source on Save" option. This automatically runs your code every time you save it (your function definitions are updated when you change your code and save).

Now call your command from the console:

```
> FlipOnce()
```

As before, your program will return either:

```
[1] "T"
or
[1] "H"
```

Now we will write a script to flip a coin $\boldsymbol{n}$ times and return the results.

In the same script window, define a new function **FlipCoins(n)**. This function will call **FlipOnce()** in a loop. Type the following:

```
FlipCoins <- function(n){
  coins <- character(n)     #initialize coins
  for (i.coin in 1:n){
    coins[i.coin] <- FlipOnce() #simulate one coin flip
  }
  return(coins)
}
```

Now save your script again, and call **FlipCoins(10)** in the console:

```
> FlipCoins(20)
 [1] "T" "H" "H" "T" "T" "T" "T" "H" "H" "H" "T" "H"
[13] "T" "H" "H" "H" "H" "H" "H" "T"
```

(Note: the numbers in brackets in your output indicate the index of the item at the beginning of that line. For instance, the second row begins with the 13th coin flip.)

For computing probabilities, we are interested in the number of coins that came up heads and tails, not the actual list of results. You should now define a function **ProbHeads** that determines the probability of getting "H" based on **m.trials** experiments.

```
ProbHeads <- function(m.trials){
  coins <- FlipCoins(m.trials)
  k.Heads <- sum(coins == "H")
  return( k.Heads/ m.trials )
}
```

Note the double equals sign used in the comparison operator. If you printed the result of the expression `coins == "H"` (using **m.trials**=100), you would get something like this:

```
  [1]  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
 [10] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 [19]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
 [28] FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
 [37]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
 [46]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
 [55] FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
 [64]  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
 [73]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
 [82] FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
 [91]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE
[100] FALSE
```

If an entry of **coins** is “Heads”, then the corresponding entry of **coins == "Heads"** is TRUE. The third line of **ProbHeads** counts the number of TRUE entries.

Notice that the result of **ProbHeads(100)** is different each time. If you run it five times, you will get results like this:

```
> ProbHeads(100)
[1] 0.49
> ProbHeads(100)
[1] 0.57
> ProbHeads(100)
[1] 0.49
> ProbHeads(100)
[1] 0.51
> ProbHeads(100)
[1] 0.54
```

These results are all around 50%, which is consistent with the true probability of obtaining heads on a fair coin.

Since we will be executing *many* trials, let’s first make **FlipCoins** more efficient by rewriting it without using a for-loop. For instance, the following line simulates flipping 10 coins:

```
> sample(c("H", "T"), 10, replace=TRUE)
```

2. In your script, rewrite **FlipCoins** so that it does not use loops. (This second version will get used in **ProbHeads** as long as you "source" your script.) Copy both your **`FlipCoins`** code and the output of running **ProbHeads(100)** into your document.

As noted already, when we estimate a probability based on **m** trials of a random experiment, we will get a slightly different probability every time. If we want to repeatedly performs sets of **m** trials, we can use the function **replicate**.

Example To simulate repeating 5 sets of 10 trials of flipping a coin, use:

```
> replicate(5, FlipCoins(10))
      [,1] [,2] [,3] [,4] [,5]
 [1,] "T"  "H"  "H"  "H"  "T"
 [2,] "H"  "T"  "H"  "H"  "T"
 [3,] "H"  "H"  "H"  "T"  "T"
 [4,] "T"  "H"  "T"  "H"  "T"
 [5,] "H"  "T"  "H"  "H"  "T"
 [6,] "T"  "T"  "H"  "H"  "H"
 [7,] "T"  "T"  "T"  "H"  "T"
 [8,] "H"  "H"  "H"  "H"  "T"
 [9,] "T"  "H"  "H"  "H"  "T"
[10,] "H"  "H"  "H"  "T"  "H"
```

The output is a 10x5 array. Each column represents one set of 10 coin flips.

3. Write a function called **MinAndMaxProbHeads** that calls **ProbHeads(m.trials)** a total of **r.reps** times (i.e., you are doing **r.reps** repetitions of **m.trials** trials) and returns the minimum and maximum probability of obtaining heads over the **r.reps** repetitions. (Note: R only lets you return one result per function. To return multiple results, return **c(result1, result2)**.)

   Run your function for the values of **m.trials** and **r.reps** shown in this table and record your results in the table. What happens as **m.trials** increases? What happens as **r.reps** increases?

\ r.reps m.trials	10	100	1000	10000
10	Min: Max:			
100				

1000				
10000				

Experiment 2: Rolling a Fair Die

In our simulations, we are often interested in returning a table of results of the following form:

Result						
Frequency						

For instance, in the coin flip example for **m**=100:

Result	Heads	Tails
Frequency	47	53

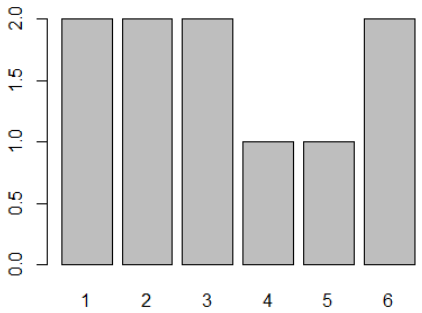
This is easy to do with R. We just save a list of the results of each trial and then use **table()** to determine the frequencies. For example, suppose we roll a 6-sided die 10 times and get the following results: 3,6,3,1,2,6,4,2,1,5. We can store these results in a list and convert them into a table, as follows:

```
> die.list<-c(3,6,3,1,2,6,4,2,1,5)
> die.table<-table(die.list)
> die.table
1 2 3 4 5 6
2 2 2 1 1 2
```

This tells us that we have rolled two 1's, two 2's, two 3's, one 4, one 5, and two 6's. We can then create a bar graph based on this data:

```
> barplot(die.table))
```

As always, you can customize your bar graph with labels.



4. Write a function **RollDice(m.trials)** that simulates rolling a dice **m.trials** times and then creates a bar plot like the one above showing the distribution of outcomes. The title of your bar plot should be "Outcome of <m> Die Rolls", where <m> is the actual value of **m** you entered. (See the **paste** function for help concatenating strings). Run your function for **m.trials** =100, 1000, and 10000. Submit all three graphs along with descriptions of their shapes. What is happening as **m.trials** increases?

Experiment 3: Rolling Multiple (n > 1) Fair Dice

Now we will consider a random experiment in which we roll $\boldsymbol{n}$ fair dice at once. For $\boldsymbol{n} = 5$, we can simulate this experiment once with the familiar command:

```
> sample( 1:6, 5, replace=TRUE)
[1] 1 4 5 3 2
```

(Note: The number 5 here represents the number of dice, not the number of trials of the experiment.)

Let's also define the random variable $X =$ the number of times 3 occurs out of **n** dice. (For the experiment shown here, we have $X = 1$.)

5. Write a function called **RollSomeDice** that simulates **m.trials** trials of the experiment in which you roll **n.dice** dice at once. Let $X =$ the number of dice showing 3.
   Your function **RollSomeDice** should output a table that shows the distribution of X as a table and as a bar plot, based on the **m.trials** trials. The title of your bar plot should be "Distribution of X (number of 3s out of <n> dice) Based on <m> Trials", where <m> and <n> are the actual values entered. Choose your axis labels appropriately.

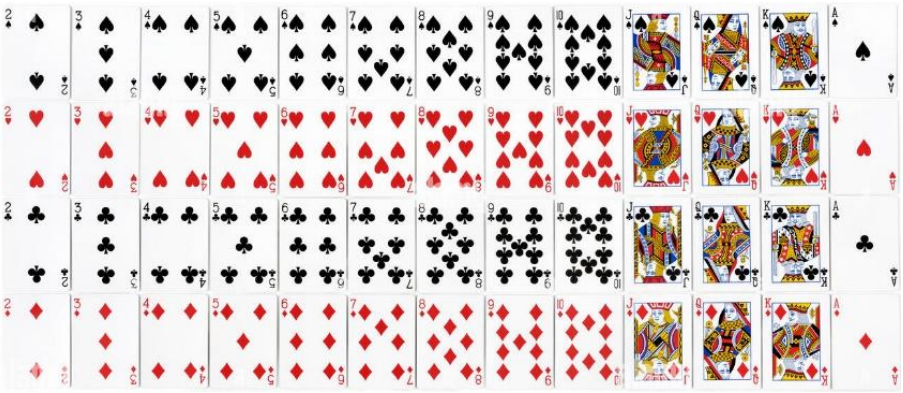
   Run your function for **m.trials** =10 000 and for each of the following:
   - **n.dice =** 1 (i.e., roll a single die)
   - **n.dice =** 2 (i.e., roll 2 dice)
   - **n.dice** = 5
   - **n.dice** = 10
   - **n.dice** = 100

   Copy-paste all five graphs into your Word document. What is happening as **n.dice** increases? Explain in a sentence or two.

Experiment 4: Drawing Cards

We modelled coin flips and die rolls as sampling with replacement. That is, we simulated the die-roll experiment by imagining a box that contained the numbers 1, 2, 3, 4, 5, and 6, drawing a number, and then putting it back and drawing again.

Some experiments are best modelled as sampling *without* replacement. Drawing cards is a common example of such an experiment. We may want to know the probability of getting 3 red cards if we draw 5 cards without replacement from a 52-card deck.



As before, we use the **sample** function, but this time we set **replace=FALSE**.

For the next two questions, let X = the number of red cards in the sample of **n** cards. (This makes X a *random variable*.)

6. Write a function **DrawCards(m.trials, n.cards, replace)** that simulates drawing **n.cards** cards from a 52-card deck. The argument **replace** indicates whether it is done *with replacement* or not. The experiment is performed a total of **m.trials** times.
   For each trial, record X = the number of red cards. (You need to decide how to model the colour of a card.)

   Your function should then create a mosaic-style histogram based on the values of X obtained. (Include **n** in the title and ensure there is one class per integer value of X.)

   Run **DrawCards** for **m=10000** and **n=1, 5, 10, 20, 50** and **replace**=TRUE.
   Run **DrawCards** for **m=10000** and **n=1, 5, 10, 20, 50** and **replace**=FALSE.

   Copy-paste all ten histograms into the following table, which you can copy into your document. Shrink them if necessary to fit them all on one page.

n	With replacement	Without replacement
1		
5		
10		
25		
50		

For each **n**, compare the two types of graphs. What do you notice as **n** increases? What accounts for the differences?