Introduction to Monte Carlo simulation

Introduction to R

R is a programming language. There are often multiple ways to accomplish a given task, so you may see different code here than in the textbook. The textbook code is a good reference, but during the semester, don't forget your class notes and examples too!

Note that R is case-sensitive. That means capitalization and spelling count! The `#` in a line of code is a commenting convention. R ignores anything after the `#` as a comment, but it will still show up in the line of code.

There will be R problems in homework, plus occasionally regular textbook homework problems will require some R calculations (instead of using a calculator or tedious calculation).

Basic built-in functions

```
x <- c(3,5,6,8) # creates a vector with 4 entries `c` stands for concatenate
print(x)
```

```
## [1] 3 5 6 8
```

```
sum(x) # adds up the entries in x
```

```
## [1] 22
```

```
sum(x)/length(x) # finds the average by summing and dividing by the number of entries
```

```
## [1] 5.5
```

```
mean(x) # finds the average value (just like the previous line of code)
```

```
## [1] 5.5
```

```
x == 6 # == means "is"
```

```
## [1] FALSE FALSE  TRUE FALSE
```

Functions useful for simulation

One of the primary reasons we will be using R and RStudio is to run simulations that illustrate and reinforce probability concepts. Later, it will also be useful for working with probability distributions.

The sample command

Suppose we want to generate a random number between 0 and 10 (integers only). There are many ways to do this, but we'll start with learning the sample command.

```
0:5
```

```
## [1] 0 1 2 3 4 5
```

```
sample(0:5, size = 1)
```

```
## [1] 3
```

Question How would you adjust the `sample` command to get 3 random values instead of one? What is the meaning of the `replace` option?

```
## write code here
```

Rolling a die To simulate rolling a standard 6-sided die:

```
sample(1:6, size = 1) # roll once
```

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

```
sample(1:6, size = 2, replace = TRUE) # roll twice
```

```
## [1] 3 3
```

```
sum(sample(1:6, 2, replace = TRUE)) # roll twice and sum the results
```

```
## [1] 8
```

Tossing a coin To simulate tossing 20 fair coins:

```
sample(0:1, size = 20, replace = TRUE) # 1 corresponds to heads, 0 to tails
```

```
##  [1] 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 1
```

To simulate tossing 20 fair coins and counting the number of heads:

```
sum(sample(0:1, size = 20, replace = TRUE))
```

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

The replicate function

The replicate function lets you repeat a command a fixed number of times.

```
replicate(5, print("Hello"))
```

```
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
```

```
## [1] "Hello" "Hello" "Hello" "Hello" "Hello"
```

Question Write some code to repeat the experiment the following experiment 100 times: toss a coin 20 times and count the number of heads.

```
## write code here
```

Defining a function (to make an indicator variable)

When doing Monte Carlo simulation, we want to be able to simulate an experiment and check whether our desired event A occurred. It's useful to make a function that outputs 1 when a trial of our experiment results in event A occurring, and 0 when it does not.

The following function runs one trial of the experiment where we toss a coin 6 times and checks whether we got 3 heads.

```
trial.simulation = function() {
  # this function is like our X_k random variables
```

```
  x = sum(sample(0:1, size = 6, replace = TRUE))
  if (x == 3)
    return(1)
  else
    return(0)
}
# when we get 3 heads in 6 tosses, this function returns 1;
# otherwise it returns 0
```

Doing Monte Carlo simulation

Once we've written our `trial.simulation` function, to do Monte Carlo simulation we simply need to repeat our experiment a large number of times and average the results.

```
mean(replicate(n = 1e6, trial.simulation()))
```

```
## [1] 0.312046
```

Question How does this compare with the exact value (found using counting)?

Setting Seeds

Let's generate two more random numbers - just from 1 to 10, using separate commands.

```
sample(1:10, 1)
```

```
## [1] 7
```

```
sample(1:10, 1)
```

```
## [1] 5
```

You likely got different results. Generating random numbers is great except that we like to be able to reproduce our work. If we each use a `sample` command, we aren't assured of getting the same random number, unless we set the same *seed* for the random number generator. If you run the next code chunk (all of it), your randomly generated value should be the same always.

```
set.seed(1)
sample(1:10, 1, replace = TRUE)
```

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

```
set.seed(1)
sample(1:10, 1, replace = TRUE)
```

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

It may sound odd to be able to generate the same random numbers every time, but this is a way to make your work reproducible. You won't always need to set a seed, but if you find you are getting unexplainable results, and want to ask for help, try comparing with a friend who has set the same seed.

Getting help

What if you don't know what a command does, and there is no instruction about it? Help for R functions can be accessed by typing `?functionname` at the console. For example, if I wanted help for the sample function, I would type `?sample` in the console.

Exercises

Problem 1

Let's write code for a Monte Carlo simulation that estimates the probability of getting exactly 1 four in 5 rolls of a die?

Question What is the exact probability (found using counting)? This will help us check whether our simulation is correct.

Monte Carlo simulation

Write a trial simulation function and then use the `replicate` and `mean` functions to get an approximation of the desired probability. Try doing 100,000 trials of the simulation. Compare with the exact probability you found.

```
# write code here
```

Problem 2

Suppose we roll a pair of dice and sum the results. What is the probability of getting a sum of at least 9? Write a Monte Carlo simulation to verify your answer.

```
# write code here
```