R for Simulations

R has many built in functions for generating random values which can make it useful to simulate processes. Simulations can be useful for “checking your work” for problems that have closed analytical solutions or getting intuition before writing a proof. And sometimes a closed form analytical solution might not exist so simulations are the only way to estimate an underlying distribution.

Often the challenge with simulations is really figuring out how to model real world processes with numbers and variables. Many times we need to re-parameterize the problem to make it easier to do with a computer. This may require creating explicit variables for things that are unobserved in real life.

It's also important to note that the "randomness" is R is only pseudo-random. The default random number generator depends on a "seed" which is some initial starting value (number) where random numbers grow from. Numbers generated from the same seed will always be the same. But each time you generate a random number, you update and change the seed. If you need to get consistent results (especially with someone else or on a different computer), use the `set.seed()` function before you draw any random numbers. But you generally only do this once (if at all) at the top of your script. R will otherwise "randomly" set the seed each time you start a new session (and the seed is usually based on the current time and the process ID). There is no need to use `set.seed()` with any of these exercises, but if you want to compare your answers with others, you should both set your seed to the same value, like `set.seed(17)` or `set.seed(525600)` or whatever you like.

Often we are interested in "expected values" of random values or "probability" of events occuring. Through simulation, we are not calculating these values directly. Instead we create a sample with the given constraints. From the sample, we can calculate a mean value to estimate the expected value, or we can calculate an observed frequency to estimate the probability.

Useful functions for simulations

If you want to draw numbers from a "well-known" distribution, there are plenty of functions to help with this. For example if you wanted a 10 random numbers from a standard normal distribution, you can call `rnorm(10)`. Most of the random number generating functions start with an "r". Here are some other useful ones

Normal Distribution	`rnorm(n, mu=0, sd=1)`
Uniform Distribution	`runif(n, min=0, max=1)`

Binomial Distribution	`rbinom(n, size, prob)`
Exponential Distribution	`rexp(n, rate=1)`
Poisson	`rpois(n, lambda)`

And there are many more. For a complete list, see the "?Distributions" help page. The first parameter to all of these are the number of values you want returned. In general, it's better (slightly faster) to call these functions once and get all the numbers you need rather than repeatedly calling them. That is, if you need 10 random normal numbers, it's better to call `rnorm(10)` rather than calling `rnorm(1)` 10 times.

Another useful function is the `sample()` function. This function takes a fixed set of values and can draw random values from the set with or without replacement. It can also be used to just shuffle a vector of values. Here are a few examples of the sample function in use

- Choose 100 male/female values (we feed in the two values we with to sample from, and note that we need to include replace=TRUE because we want to choose more samples than are in the vector we supplied): `sample(c("M","F"), 100, replace=TRUE)`
- Choose 100 male/females but we want about 75% to be female (the order of the probabilities should match the order of the values in the original vector): `sample(c("M","F"), 100, replace=TRUE, prob=c(.25, .75))`
- Randomly choose 3 out of the first 10 letters of the alphabet (since we didn't allow replacement, we will not get any duplicate values here): `sample(letters[1:10], 3)`
- Shuffle the values 1 to 10: `sample(1:10)`

The `replicate()` function can help you repeat an expression multiple times and capture the responses. The expression should have some randomness in it otherwise you will just get the same value back multiple times. For example, let's use the `sample()` function to draw a collection of M/F values where the probability of F should be .75. But let's see how often we actually get more males when we only draw 10 values. We can raw 1000 samples with the following command :

```
experiment <- replicate(1000, {
  mydraw <- sample(c("M","F"), 10, replace=TRUE,
     prob=c(.25, .75))
  sum(mydraw=="M")>sum(mydraw=="F")
})
mean(experiment)
```

You should get ~0.012 or so. Since we wanted to run more than one command each time, notice that these commands are wrapped in braces in the call to replicate(). At the end we are returning boolean values that are true when the number of M's is greater than the number of F's. We take the mean of those boolean values to find the frequency of how often that happened.

Exercises

1. Let's say X is a random normal variable with mean 0 and standard deviation 1. If you take the cosine of that random variable, is it true that E[cos(X)] == cos(E[X])?
2. Let’s say you have a line segment and you randomly cut it at two points. What is the probability that you can make a triangle from the three pieces? (HINT: remember the triangle inequality)
3. If you have three positive real numbers that sum to 10, what is the expected value of their product?
4. Let’s say you have a row of 5 chairs. If you ask everyone to get up and randomly shuffle their positions, how many do you expect to be in the same seat on average. How does this change with different numbers of chairs.
5. If you choose 3 points at random in a circle what is the expected value of the area of the triangle formed by the 3 points? (HINT: to choose points in a circle, consider using polar coordinates)
6. Say you have an infinite “grid” with parallels lines exactly 1" apart (like a sheet of very-wide-ruled paper). Now say you drop a needle that’s 1" long on this grid. What is the probability that this needle will cross one of the lines? (This problem is referred to as Buffon's Needle)

Answers

1) It's not true. E[cos(x)] = 1/sqrt(e) ≅ 0.6065
2) 1/4 = 0.25
3) 50/3 ≅ 16.6667. If you get 13.76 or 22.83, you're not sampling from the sample space uniformly.
4) E[# in same chair] = 1 (no matter n)
5) 0.2477
6) 2/π ≅ 0.6366