Week 4 - Simulation

Sally Longmore

31/01/2020

One of the great advantages of using a statistical programming langurage like R is it's vast collection of tools for simulating random numbers.

This lesson assumes familiarity with a few common probability distributions, but these topics will obly be discussed with respect to random number generation. Even if you have no prior experience with these concepts, you should be able to complete the lesson and understand the main ideas.

The first function we will use to generate random numbers is sample(). Use ?sample to pull up the documentation.

Let's simulate rolling a four six-sided dice: sample(1:6, 4, replace = TRUE)

```
sample(1:6, 4, replace = TRUE)
```

```
## [1] 6 6 5 5
```

Now repeat the command to see how your results differes. (The probability of rolling the exact same result is (1/6)^4 = 0.00077, which is pretty small!)

```
sample(1:6, 4, replace = TRUE)
```

```
## [1] 4 1 4 5
```

sample(1:6, 4, replace = TRUE) instructs R to randomly select four numbers between 1 and 6, with replacement. Sampling with replacement simply means that each number is "replaced" after it is selected, so that the same number can show up more than once. That is what we want here, since what you roll on one die should not affect what you roll on any of the other dices.

Now sample 10 numbers between 1 and 20 WITHOUT replacment. To sample without replacement, simply leave off the **replace** argument.

```
sample(1:20, 10)
```

```
##  [1] 12 14 15 16 13 11  3 18 17 20
```

Since the last command sampled without replacement, no number appears more than once in the output.

LETTERS is a predefined variable in R containing a vector of all 26 letters of the English alphabet.

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

The sample() function can also be used to permute, or rearrange, the lements of a vector. For example trype sample(LETTERS) to permute all 26 letters of the English alphabet.

```
sample(LETTERS)
```

```
##  [1] "D" "Q" "Z" "X" "L" "I" "C" "K" "W" "V" "J" "E" "S" "U" "A" "H" "N" "R" "B"
## [20] "Y" "T" "F" "O" "G" "M" "P"
```

This is identical to taking a sample of size 26 from LETTERS, without replacement. When the **size** argument to sample() is not specified, R takes a sample equal in size to the vector from which you are smapling.

Now suppose we want to simplate 100 flips of an unfair two-sided coint. This particular coin has a 0.3 probability of landing on 'tails' and a 0.7 probability of landing on 'heads'.

Let the value 0 represent tails and the value 1 represent heads. Use sample() to draw a sample of sive 100 from the vector c(0, 1), with replacement. Since the coin is unfair, we must attach specific probabilities to the values 0 (tails) and 1 (heads) with a fourth argument prob = c(0.3, 0.7). Assign the result to a new variable called flips.

```
flips <- sample(c(0, 1), 100, replace = TRUE, prob = c(0.3, 0.7))
flips
```

```
##   [1] 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1
##  [38] 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 1 1
##  [75] 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1
```

Sice we set the probability of landing on heads on any given flip to be 0.7, we'd expect approximately 80 of our coin flips to ave the value 1. Count the actual number of 1s contained in flips using the sum() function.

```
sum(flips)
```

```
## [1] 75
```

A coin flip is a binary outcome (0 or 1) and we are performing 100 independent trials (coin flips), so we can use rbinom() to simulate a binomial random variable. Pull up the documentation for rbinom() using ?rbinom.

Each probability distribution in R has an R*** function for for **random**, a d*** function for **density**, a p*** function for **probability** and a q*** function for **quantile**. We are most interested in the r*** functions in this lesson, but I encourage you to explore others on your own.

A binomial random variable represents the number of **successes** (heads) in a given number of idependent **trials** (coin flips). Therefore we can generate a single random variable that represents the number of heads in 100 flips of our unfair coin using rbinom(1, size = 100, prob = 0.7). Note that you only specify the probability of **success** (heads) and NOT the probaility of **failure** (tails).

```
rbinom(1, size = 100, prob = 0.7)
```

```
## [1] 78
```

Equivalently, if we want to see all of the 0s and 1s, we can request 100 observations, each of size 1, with success probability of 0.7. Give it a try, assigning the result to a new variable called flips2.

```
flips2 <- rbinom(n = 100, size = 1, prob = 0.7)
flips2
```

```
##   [1] 1 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1
##  [38] 1 1 1 0 1 1 1 0 0 1 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1
##  [75] 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 1
```

Now use sum() to count the number of 1s (heads) in flips2. It should be close to 70.

```
sum(flips)
```

```
## [1] 75
```

Similar to rbinom(), we can use R to simulate random numbers from many other probability distributions. Pull up the documentation for rnorm() now.

The standard normal distribution has a mean 0 and standard deviation 1. As you can see under the **Usage** section in the documentation, the default values for **mean** and **sd** arguments to rnorm() are 0 and 1 respectively. This rnorm(10) will generate 10 random numbers from a standard normal distribution.

```r
rnorm(10)
```

```
##  [1] -0.19357571  0.28852567 -0.43208541  2.38186722 -0.47253405 -0.03576173
##  [7] -0.18076484  0.51729882  0.58036013  2.02937639
```

Now do the same, except with a mean of 100 and a standard deviation of 25.

```r
rnorm(10, mean = 100, sd = 25)
```

```
##  [1] 102.13246 103.49109  81.11060 118.87827  79.26813  82.26695 138.87861
##  [8]  54.16490  89.70813  70.51644
```

Finally, what if we want to simulate 100 **groups** of random numbers, each containing 5 values generated from a Poisson distribution with mean 10? Let's start with one group of 5 numbers, then I'll show you how to repeat the operation 100 times in convenient and compact way.

Generate 5 random values from a Poisson distribution with mean 10. Check out documentation for rpois() if you need help.

```r
rpois(5, 10)
```

```
## [1]  7 10 11  6 12
```

Now use replicate(100, rpois(5, 10)) to perform this operation 100 times. Store the result in a new variable called my_pois.

```r
my_pois <- replicate(100, rpois(5, 10))
my_pois
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]   14   11   11   11    6    9   15    7   17    11    13     8     6     8
## [2,]    9   18    8    7    5   10   12    9   14    12     7    13     7     8
## [3,]   14    4   13   13   12   17    8    9    7    11    17    16     9     8
## [4,]    6    8    7    6    8   12    8    2   15    10    14    12    13     9
## [5,]    6    8   12   14    7    9   16   10   15     9    10    15     9    12
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]    10    16    11     8    19    10     9    17    11    14    10    12
## [2,]    12    11    10    11     7     8    11     9     9     7     5    11
## [3,]    11    12    16    11    10     8     7    12    10    12    15     7
## [4,]    11     8     8     9    10    10    11    12    11     6    12     8
## [5,]     4    14     6    12     7    10    14     5    18     3     8     6
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]     3     8    10     5    16    12    14    13     6     9    10    12
## [2,]     5    13    14     5    14     9     7     8     7     5    12    16
## [3,]     9    11    10    12     9    11    10    16     9    17    10    12
## [4,]    10    10    10    14     8    10     9    13     8    11    14    14
## [5,]     5    11    12    11     7    13    11    15    13    10    10     8
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,]    14    12     8     8     7    15    12    11     9     6     6    14
## [2,]     5    11    15    14     8    16    19    12    12    18     7     5
## [3,]    10    11     9    12    10     2     8     9     4     8     9     9
## [4,]     9     6    10     9    10    17    11    11     8     8     9    11
## [5,]     8    11    11     6    13    11    16    10     6     5    14    13
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
## [1,]     9    10     6    12    12    11     6    16     5    10     9    12
## [2,]    11     7     6    13    12    11    11    12     7    10    15    18
## [3,]    14     7    10    14    14     7     5    15    14     6    19    15
## [4,]    12    11    11     9     7     5    11    10    11    12     5     6
## [5,]    11     7     6    12    11    11    10    10    12    12    13     9
##      [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
## [1,]    11     8     9     4    13    16     6     8     4    12    13    10
## [2,]     8     8    12     6    10     9    14    11    13    14     9    11
## [3,]     9     7    12    11    10    12     6    11     4    14     7    18
## [4,]    12     7    14     9    12    14    10     5     7    11     9    12
## [5,]     7    15    10    11     6    16     7    10    11     6    12    10
##      [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
## [1,]    13     4     9    12     9     9    17    14     6     3     4    11
## [2,]     8    12    14     4     9    17     8     7    11    13     9     7
## [3,]    13    13     7     7    11     6    15    10    10     7    14     9
## [4,]    10    13    14     6     7     7     7     9     6    10    10     9
## [5,]     5    10    10    15    11     5    12    10    17    13    11    11
##      [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
## [1,]    10    12    14     9    10     6     8    12    20    11    12    11
## [2,]    10    13     7     9     3    10    13    12     8    12    13     9
## [3,]     7     7    11     6    10    11     8     9     9     8    11     7
## [4,]    11     8    15     7     6    12    12    12    10     3    14    13
## [5,]    14     3     8     9     8    13     9    11     9     8    11     5
##      [,99] [,100]
## [1,]    12      7
## [2,]     4      7
## [3,]     9     10
## [4,]     5     11
## [5,]     8     10
```

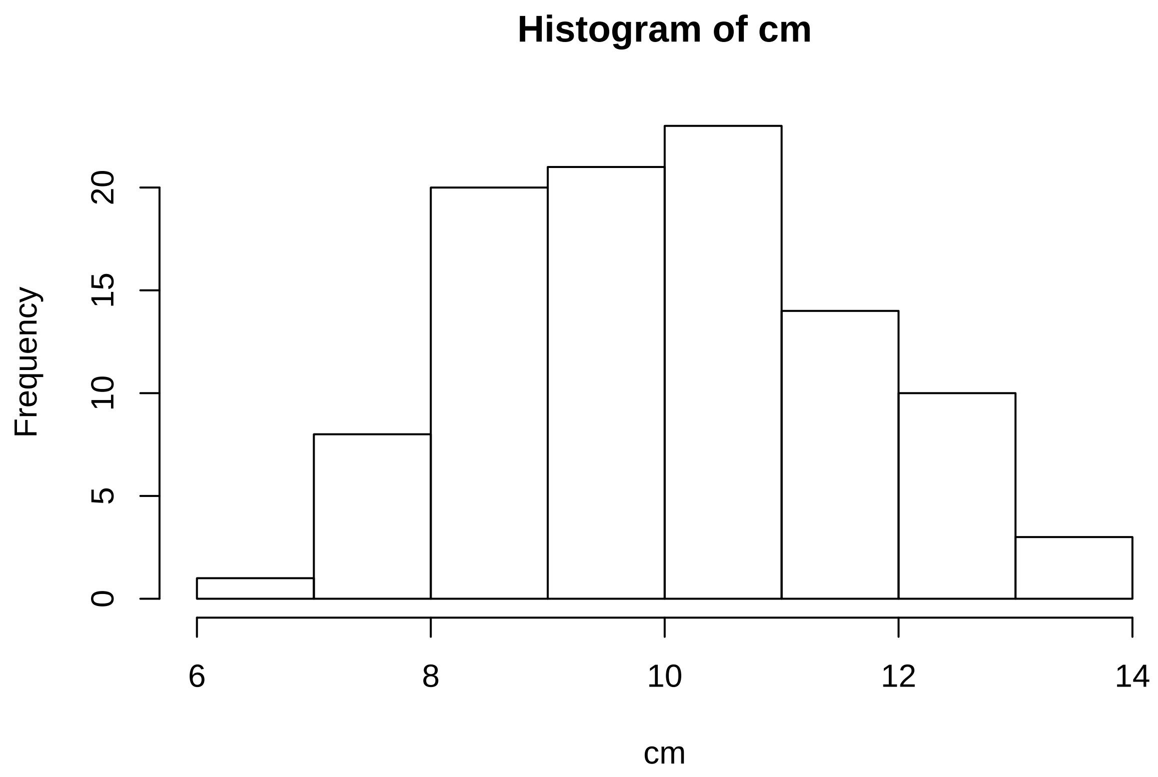
replicate() created a matrix, each column of which contains 5 random numbers generated from a Poisson distribution with a mean of 10. Now we can find the mean of each column in my_pois using the colMeans() function. Store the result ina variable called cm.

```r
cm <- colMeans(my_pois)
cm
```

```
##   [1]  9.8  9.8 10.2 10.2  7.6 11.4 11.8  7.4 13.6 10.6 12.2 12.8  8.8  9.0  9.6
##  [16] 12.2 10.2 10.2 10.6  9.2 10.4 11.0 11.8  8.4 10.0  8.8  6.4 10.6 11.2  9.4
##  [31] 10.8 11.0 10.2 13.0  8.6 10.4 11.2 12.4  9.2 10.2 10.6  9.8  9.6 12.2 13.2
##  [46] 10.6  7.8  9.0  9.0 10.4 11.4  8.4  7.8 12.0 11.2  9.0  8.6 12.6  9.8 10.0
##  [61] 12.2 12.0  9.4  9.0 11.4  8.2 10.2 13.4  8.6  9.0  7.8 11.4 10.0 12.2  9.8
##  [76] 10.4 10.8  8.8  9.4  8.8 11.8 10.0 10.0  9.2  9.6  9.4 10.4  8.6 11.0  8.0
##  [91]  7.4 10.4 10.0 11.2 11.2  8.4 12.2  9.0  7.6  9.0
```

Now let's take a look at the distribution of our column means by plotting a hustogram with hist(cm).

```r
hist(cm)
```



Looks like our column means are almost normally distributed. That's the Central Limit Theorem at work, but that's a lesson for another day.

All of the standard probability distributions are built into R, including exponential **exp()**, chi-squared **rchisq()**, gamma **rgamma(), . . . Well, you see the pattern.

Simulation is practically a field of its own, and we have only skimmed the surface of what is possible. I encourage you to explore these and other functions futher on your own.