Handout 06: Basic Simulations

January 17, 2017

THE FOCUS OF THIS COURSE is on symbolic mathematics, however I think doing probability without any computational simulation is a wasted opportunity. Today I will demonstrate how to do simulations in the R programming language. Later in the course I will have you do simluations on your own.

R is a free and open-source programming language available for Mac, Windows, and Linux. I will include instructions for downloading R on your own laptop on Box in the notes directory.[1] R and RStudio are also available in the computer labs in Jepson.

[1] The notes come from Math209 course, so there may be some references that are not applicable to you but the bulk of the material will be useful.

A very simple example

The R language is quite extensive and we will be using only a very small portion of it in MATH329. I will teach it to you by way of example. Variables in R can be assigned using the `<-` symbol. Two functions we will use to make vectors (variables with n-elements) are `:` and `seq`. The first makes a run of integers between a start- and end-point and the second repeates a given value a certain numbers of times. For example:

```
x <- 5:10
x
```

```
## [1]  5  6  7  8  9 10
```

```
y <- rep(3, 10)
y
```

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

To access and re-assign a particular element of a vector we use square brackets:

```
x[3]
```

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

```
x[3] <- 10
x
```

```
## [1]  5  6 10  8  9 10
```

Note that R starts indicies at one instead of zero. We will make use of the `sample` function, which picks a fixed number of elements from a set with or without replacement:

```
sample(1:6, 10, replace = TRUE)
```

```
##  [1] 6 5 2 1 2 1 3 4 1 1
```

```
sample(1:6, 10, replace = TRUE)
```

```
##  [1] 3 1 3 5 5 4 2 2 6 1
```

```
sample(1:6, 10, replace = TRUE)
```

```
##  [1] 3 3 1 1 6 6 4 2 4 4
```

Finally, we will use a *for-loop* to replicate a given set of commands a fixed number of times. For example, the previous code snippet could have been done this way instead:

```
for (i in 1:3) {
  sample(1:6, 10, replace = TRUE)
}
```

Using little more than these tools we will be able to run a large number of probability experiments.

Sum of a pair of dice

Let's use R to figure out how often a pair of dice come up with a sum less than 4:

```
N <- 100
dice_sum <- rep(0, N)
for (i in 1:N) {
  temp <- sample(1:6, 2, replace = TRUE)
  dice_sum[i] <- sum(temp)
}

dice_sum
```

```
##   [1]  6  2  8  6  7  6 10  7  6  8  8  5  7
##  [14]  8  2 10  4  6  5  8  6  9  7  7  2  5
##  [27] 11 10  7 10  4  3  7  3  9  8 11 11  8
##  [40]  7  5  7  4 11  7  4  7  9  6  2 10  5
##  [53]  7  7  9  4  6 12  8  8  7  8 11  6  8
##  [66]  6  5  8  7  3  4  5  7  5  6  5 11  4
##  [79] 12  7  8  8  8  8  7 11 12  6  8  4  7
##  [92]  7 10  7  2  4  3  3  9  9
```

```
mean(dice_sum < 4)
```

```
## [1] 0.1
```

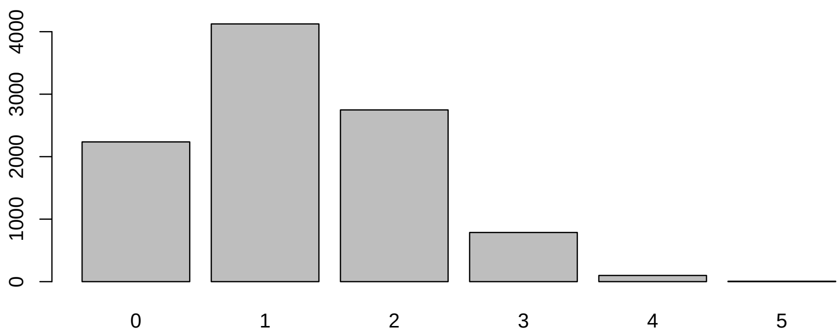
Taking the `mean` of a logical statement makes R print out what proportion of the simulations have the result hold.

Number of spades in a hand of cards

Now, what is the probability that a random hand of cards contains a given number of spades? We can simulate all of the possibilities with R, and print out the results using `barplot`.

```
N <- 10000
spade_sum <- rep(0, N)
for (i in 1:N) {
  temp <- sample(rep(1:4, 13), 5, replace = FALSE)
  spade_sum[i] <- sum(temp == 1)
}
```

```
barplot(table(spade_sum))
```



This gives the entire distribution of the random output, something that will be the focus of the class after the first exam.

Probability of a Three-of-a-Kind

A slightly more involved example figures out the probability of getting a three-of-a-kind:

```
N <- 10000
three_of_kind <- rep(0, N)
for (i in 1:N) {
  temp <- sample(rep(1:13, 4), 5, replace = FALSE)
  if (max(table(temp)) == 3 & min(table(temp)) == 1) three_of_kind[i] <- 1
}
```

```
mean(three_of_kind)
```

```
## [1] 0.021
```

Probability of a Full House

Or, we can compute the probability of a full house:

```
N <- 10000
full_house <- rep(0, N)
for (i in 1:N) {
  temp <- sample(rep(1:13, 4), 5, replace = FALSE)
  if (max(table(temp)) == 3 & min(table(temp)) == 2) full_house[i] <- 1
}

mean(full_house)
```

```
## [1] 7e-04
```

Flipping two coins

What is the probability that two coins both come up heads? Here we represent heads as a 1. Notice that there is a faster way to do this, but this will be the best for generalizing to a more complex example:

```
N <- 10000
outcome_true <- rep(0, N)
for (i in 1:N) {
  first_coin <- sample(0:1, 1, replace = FALSE)
  second_coin <- sample(0:1, 1, replace = FALSE)

  if (first_coin == 1 & second_coin == 1) outcome_true[i] <- 1
}

mean(outcome_true)
```

```
## [1] 0.2513
```

The result closely matches the 25% probability we would compute analytically.

Flipping Until Heads

Let's flip coins until a heads show up. What is the average number of flips required?

```
N <- 10000

num_flips <- rep(0, N)
for (i in 1:N) {
  this_coin <- sample(0:1, 1, replace = FALSE)
  flips <- 1

  while(num_flips[i] == 0) {
    if (this_coin == 1) {
      num_flips[i] <- flips
    } else {
      this_coin <- sample(0:1, 1, replace = FALSE)
      flips <- flips + 1
    }
  }
}

mean(num_flips)
```

```
## [1] 1.9979
```

Two Consecutive Heads (HH)

Not that you know how long it takes on average to flip to get a heads, let's see how long it takes to get two consecutive heads:

```
N <- 10000
n <- 100

num_flips <- rep(0, N)
for (i in 1:N) {
  last_coin <- sample(0:1, 1, replace = FALSE)
  this_coin <- sample(0:1, 1, replace = FALSE)
  flips <- 2

  while (num_flips[i] == 0) {
    if (this_coin == 1 & last_coin == 1) {
      num_flips[i] <- flips
    } else {
      last_coin <- this_coin
      this_coin <- sample(0:1, 1, replace = FALSE)
      flips <- flips + 1
    }
  }
}

mean(num_flips)
```

```
## [1] 6.0353
```

Tails Followed by Heads (TH)

Finally, let's end with something particularly interesting. The probability of getting a tails followed by a heads:

```
N <- 10000
n <- 100

num_flips <- rep(0, N)
for (i in 1:N) {
  last_coin <- sample(0:1, 1, replace = FALSE)
  this_coin <- sample(0:1, 1, replace = FALSE)
  flips <- 2

  while (num_flips[i] == 0) {
    if (this_coin == 1 & last_coin == 0) {
      num_flips[i] <- flips
    } else {
      last_coin <- this_coin
      this_coin <- sample(0:1, 1, replace = FALSE)
      flips <- flips + 1
    }
  }
}

mean(num_flips)
```

```
## [1] 3.9645
```

What is particularly interesting about this?