# Conditional Probability in R

Becky Tang

## Indexing for conditional probabilities

It may be useful, especially when obtaining conditional problems, to subset or filter the outputs from a simulation by a certain condition. We can do this by indexing.

In general, indexing is obtained using square-bracket notation. If I want to obtain the value in the `i`-th position of a vector `v`, I would type `v[i]`.

We can put a logical condition in the brackets to say "give me the indices where the logical condition is TRUE".

Let's simulate for the problem we've seen before: "Mr. Smith has two children. It is known that the eldest is a girl. What is the probability that both children are girls?"

We will start with a small number of simulations just for illustration:

```r
set.seed(310)
outcomes <- c("GG", "GB", "BG", "BB")
B <- 10
r <- replicate(B, sample(outcomes, size = 1))
# the following code does the same in this case:
r <- sample(outcomes, size = B, replace = T)
r
```

```
##  [1] "GG" "GB" "BG" "GB" "GG" "BG" "GB" "GB" "BB" "GG"
```

```r
r %in% c("GG", "BG")
```

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

```r
# subset r to grab the replicates where we do have eldest is girl
r[r %in% c("GG", "BG")]
```

```
## [1] "GG" "BG" "GG" "BG" "GG"
```

Then, I can obtain conditional probabilities by calculating a mean of 0s and 1s (i.e. a proportion) on this subset of the simulations:

```r
mean(r[r %in% c("GG", "BG")] == "GG")
```

```
## [1] 0.6
```

Now let's increase the simulations to make sure we approximate the probability well:

```r
B <- 10000
r <- sample(outcomes, size = B, replace = T)
# roughly one half
mean(r[r %in% c("GG", "BG")] == "GG")
```

```
## [1] 0.5009126
```

We can use the same simulations to estimate the following: "Mr. Jones has two children. It is known that at least one is a girl. What is the probability that both children are girls?"

```r
mean(r[r %in% c("GG", "BG", "GB")] == "GG")
```

```
## [1] 0.3287635
```

## Loops

Often, we will need to perform a certain calculation or simulation several times. If each iteration of the expression is independent and identical to the previous iteration, we can use `replicate`.

Sometimes, however, we might have a more nuanced stopping condition when repeating/iterating through statements, or the calculation itself depends on which iteration we are on.

In these cases, if the know how many iterations we'd like to run, we should implement a `for()` loop.

### `for` Loops

Suppose we want to perform the factorial calculation manually, rather than using the `factorial()` function. I can calculate 5! via the following code.

We must use the `for()` function, then define a counter variable that keeps track of the iteration. In this case, it is `i`. Then we say `in 1:5` to convey that we'd like the repeat the code within the curly braces 5 times, where each time, `i` takes the value 1, then 2, . . . , 5. Notice that we explicitly use `i` in the code itself!

```r
my_prod <- 1
for(i in 1:5){
    my_prod <- my_prod * i
    print(paste("Iteration ", i, ": ", my_prod))
}
```

```
## [1] "Iteration  1 :  1"
## [1] "Iteration  2 :  2"
## [1] "Iteration  3 :  6"
## [1] "Iteration  4 :  24"
## [1] "Iteration  5 :  120"
```

```r
my_prod
```

```
## [1] 120
```