Conditional Probability in R

Becky Tang

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 interations we'd like to run, we should implement a `for()` loop.

Other times, we don't know at the outset how many times we will need to run the experiment, and so we will use `while` loops. The `while` loop will continue executing until a statement is proved wrong.

`for` Loops

Suppose we want to perform the factorial calculation manually, rather than using the `factorial()` function. I can calculate 10! 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 1:10` to convey that we'd like the repeat/iteration the code within the curly braces 1-10 times, where each time, `i` takes the value 1, then 2, . . . , and finally 10. Notice that we explicitly use `i` in the code itself!

```
my_prod <- 1
for(i in 1:10){
    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"
## [1] "Iteration  6 :  720"
## [1] "Iteration  7 :  5040"
## [1] "Iteration  8 :  40320"
## [1] "Iteration  9 :  362880"
## [1] "Iteration  10 :  3628800"
```

```
my_prod
```

```
## [1] 3628800
```

`while` loops

Other times, we don't know when an experiment is over. Recall the example from last week: we play a game where the score we receive is an integer from 1-50, all equally likely. We play one round, and call the score X. Then we keep playing until we score Y such that $Y \geq X$. All scores remain equally likely. In this case, we don't know how many times we have to play until we obtain such a Y, but I know that once I do score a number at least as large as X, I'm done playing. We found the theoretical probabiliy $P(Y = 50) \approx 0.09$.

Suppose we want to simulate this game. We use the `while` function, where the logical statement in parentheses means we will keep running this code *while* that statement is true. I will keep track of a variable `n` to sum up the number of times I play. Suppose I scored $X = 40$. Then I will keep playing while future rolls are less than 10.

```
# initialize
X <- 40
second_score <- 0
n <- 1

while(second_score < X){
  second_score <- sample(1:50, 1)
  n <- n + 1
  print(paste("Iteration ", n, ": ", second_score))
}
```

```
## [1] "Iteration  2 :  10"
## [1] "Iteration  3 :  19"
## [1] "Iteration  4 :  31"
## [1] "Iteration  5 :  28"
## [1] "Iteration  6 :  46"
```

```
Y <- second_score
Y
```

```
## [1] 46
```

Combining `replicate()` with loops to estimate probabilities

The example above performed one simulation of a game. However, we know that to estimate probabilities, we need to perform the simulation many times. We might want to write a function that codes up the simulation of the game, and use that function in `replicate()`.

For the game above, we found that $P(Y = 50) = \sum_{i=1}^{50} \frac{1}{51-i} \times \frac{1}{50} \approx 0.09$. Let's confirm this in simulation.

```
boring_game <- function(){
  # simulate the game from the first round:
  X <- sample(1:50, 1)
  second_score <- 0
  while(second_score < X){
    second_score <- sample(1:50, 1)
  }
  Y <- second_score
  Y
}
```

```r
# r holds the values Y from each one of 10000 simulations of the game
r <- replicate(10000, boring_game())
sum(r == 50)/10000
```

```
## [1] 0.0891
```

```r
# note, we can use the mean() function to streamline the probability calculation
mean(r==50)
```

```
## [1] 0.0891
```

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]`.

```r
v <- 10:15
v[2]
```

```
## [1] 11
```

We can put a logical condition in the brackets to say "give me the indices where the logical condition is TRUE". So for example, suppose we play our boring game 100 times. I can index to obtain the values from the simulations where $Y > 30$:

```r
t <- replicate(100, boring_game())
t[t > 30]
```

```
##  [1] 46 50 50 50 49 36 34 47 49 43 49 45 46 42 45 43 33 35 49 49 49 42 45 40 42
## [26] 46 33 34 47 38 50 37 50 44 49 39 48 39 50 49 35 49 31 47 47 40 46 40 47 50
## [51] 50 46 50 37 38 37 48 40 38 48 50 48 41 38 44 48 42
```

Then, I can obtain conditional probabilities by calculating proportion on this subset of the simulations. For example, the following code simulates the probability $P(Y = 50|Y > 30)$.

```r
t <- replicate(1000, boring_game())
mean(t[t > 30] == 50)
```

```
## [1] 0.129506
```

Notice that this probability is larger, because by conditioning we have removed all cases where $Y \leq 30$.