Intro to R and Simulations

Becky Tang

02-18-2026

R is a calculator

```
1 + 2
```

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

```
3*4
```

```
## [1] 12
```

```
5^2
```

```
## [1] 25
```

```
sqrt(9)
```

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

```
# calculates e^2
exp(2)
```

```
## [1] 7.389056
```

```
# this evaluates the binomial coefficient "5 choose 3"
choose(5, 3)
```

```
## [1] 10
```

R respects PEMDAS:

```
1 + 1/2
```

```
## [1] 1.5
```

```
(1+1)/2
```

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

R objects

Booleans

Booleans/logicals can be formed with (in)equality symbols:

```
# Is 1 greater than 2?
1 > 2
```

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

```r
# Is 2 greater than or equal to -1?
2 >= -1
```

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

```r
# Does 3 equal 4? Note that evaluate equality, we need double equals signs
3 == 4
```

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

Variables

Storing/saving values into variables is achieved by using the syntax `<variable name> <- <value>`. Note that the variable name cannot begin with a number! The result of a stored variable is not displayed to the user. Type out the variable name after storing to explicitly show the values.

```r
a <- 1
a
```

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

Vectors

Create vectors of values (i.e. a list of values) using the concatenate `c()` function:

```r
v <- c(1, 4, 16, 9)
v
```

```
## [1]  1  4 16  9
```

The syntax `a:b` creates a sequence of integers from `a` to `b`.

```r
c <- 1:4
c
```

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

R is vectorized language, so it operates element-wise in vectors. Also, under the hood, booleans/logicals are treated as 1/0 for TRUE/FALSE

```r
c + v
```

```
## [1]  2  6 19 13
```

```r
v >= 3
```

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

```r
# remember: FALSE is like 0, and TRUE is like 1
sum(v >= 3)
```

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

Functions

A lot of the commands we have been working with are **functions**. You can tell a command is a function if it has parentheses (e.g. `sum()` and `c()`). A function usually takes in some argument(s) as input and returns something back as output. We can write custom functions using the function `function()`. We need to specify the name of the function, what it expects as input, as well as what to return back to the user.

Birthday problem, generalized

Suppose a group of k people each choose a number randomly with replacement from a list of n distinct numbers. If $k \leq n$, the probability of at least one match in the group is 1 minus the probability of no matches:

$$1 - \frac{n(n-1)(n-2)\cdots(n-k+1)}{n^k} = 1 - \frac{n!}{(n-k)!n^k} = 1 - \frac{n!k!}{k!(n-k)!n^k} = 1 - \binom{n}{k}\frac{k!}{n^k}$$

We can write this a function in R to obtain this probability for arbitrary n and k:

```
prob_match <- function(n, k){
  1 - choose(n, k) * factorial(k)/(n^k)
}
```

Note that the function name is prob_match, and the arguments/inputs are called `n` and `k`. The output is the probability of at least one match for the specified values of `n` and `k`. We can now use this function! For example, the probability of at least one match among 23 people choosing a number between 1 and 365 is:

```
prob_match(365, 23)
```

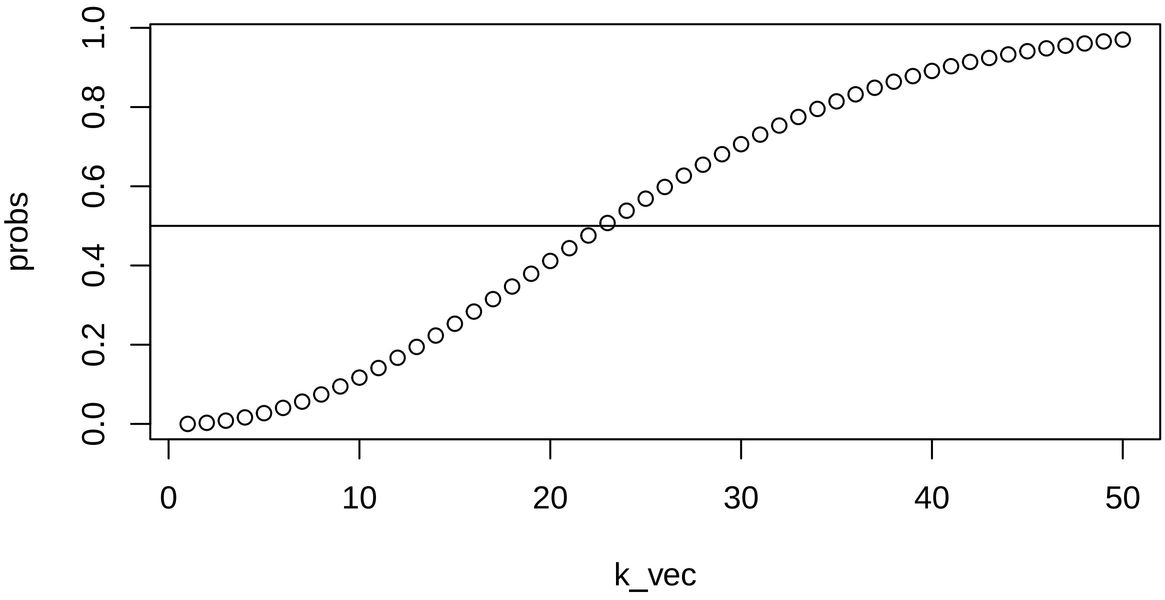
```
## [1] 0.5072972
```

Basic plotting

We can plot one variable against another (e.g. input and output of a function) using the `plot()` function. The first argument is a vector of values for the x-axis variable (e.g. input), and the second argument is a vector for the y-axis variable (e.g. output).

For example, we will plot probabilities of each least one birthday match in a room full of $k = 1, \ldots, 50$ people:

```
k_vec <- 1:50
probs <- prob_match(365, k_vec)
plot(k_vec, probs)
abline(h = 0.5) # adds horizontal line at 0.5
```



Sampling and simulating

The `sample()` function generates random values from a pool of values contained in a specific vector. You can twist many knobs to get sampling with or without replacement, uneven probabilities, and different sized samples.

```
vals <- 1:10
# sampling five values from vals without replacement.
# By default, all values are equally likely.
sample(vals, size = 5)
```

```
## [1] 6 1 8 9 4
```

```
# Defaults sampling without replacement the same size as input
# i.e. just randomly shuffles
sample(vals)
```

```
##  [1]  4 10  9  8  1  6  7  2  3  5
```

```
# Sampling five values from vals with replacement
# Remark: replace = F is default behavior
sample(vals, size = 5, replace = T)
```

```
## [1] 3 4 7 5 2
```

Approximating probabilities

In the following, we simulate flipping a fair coin once:

```
sides <- c("H", "T")
sample(sides, size = 1)
```

```
## [1] "H"
```

We can approximate probabilities by repeatedly simulating an experiment, counting up how many times a favorable outcome occurred, and dividing by the total number of experiments we did. This is easily done using the `replicate()` function:

```
# Replicate flipping a coin 100 times
flips <- replicate(1000, sample(sides, size = 1))

# Count up number of times we saw Heads, divided by total number of simulations
sum(flips == "H")/1000
```

```
## [1] 0.503
```