Introduction to Computational BioStatistics with R: flow control

Erik Spence

SciNet HPC Consortium

29 September 2020

Today's slides

To find today's slides, go to the "Introduction to Computational BioStatistics with R" page, on the right, under Lectures, "Loops".

`https://support.scinet.utoronto.ca/education`

About today's class

Today's class will explore the wild wild world of:

- Loops,
- Conditionals,
- Factors,
- Tables.

Loops

Loops are a feature of all programming languages. What are loops?

- A loop is a programming structure that allows for the same chunk of code to be run over and over again.
- Each time the chunk of code is run, one or more variables change value, allowing for slightly different behaviour each 'iteration' of the loop.
- This allows for more efficient writing of code, since you don't need to copy-and-paste a chunk of code many times, if you need to run it many times.
- As with most coding structures, loops can be inside other loops ('nested').

R loops

R has three types of loops. The first is the "for" loop:

- "For" each element in the list or vector, assign the element value to the "loop variable" ("i", in this case).
- Then perform the code inside the code block.
- Code blocks are indicated using { and }.
- Repeat the code block for each value of the list or vector.

```
>
> b <- c("Hello", "World", "From",
+    "A", "Vector")
>
> b
[1] "Hello" "World" "From" "A" "Vector"
>
> for (i in b) {
+    cat(i, "\n")
+ }
Hello
World
From
A
Vector
>
```

R loops, continued

The for loop is used whenever you know ahead of time how many times you want to run the code block. In this example, we want to run the code block once for each element in the list.

Note that the loop variable, my.loop.var in this case, can be named anything.

```
>
> for (my.loop.var in list('cow', 1, F, 'pants')) {
+
+    cat(my.loop.var, "\n")
+
+ }
cow
1
FALSE
pants
>
```

R loops, continued more

The second kind of loop is the "while" loop:

- The while loop will continue for as long as the "test condition" is TRUE.
- Be careful not to create infinite loops.
- Type "Ctrl-C" to kill your infinite loop, if you get into one.

The while loop is used whenever you don't know ahead of time how many times you want to run the code block.

The third loop is the "repeat" loop. We will not cover it here.

```
>
> i <- 1
> while (i < 4) {
+     cat(i, "\n")
+     i <- i + 1
+ }
1
2
3
>
> # Don't do this!
> while (TRUE) cat("hello", "\n")
"hello"
"hello"
⋮
>
```

Conditionals

R has the usual types of conditionals. The most commonly used is the "if" statement:

- If the condition is TRUE, then the commands in the code block are executed.
- If the condition is FALSE, then the code block is skipped.
- Notice that, if one code block is inside another code block, the entirety of the second code block is indented.

```
>
> ("pants" == "blue")
[1] FALSE
>
> if ("pants" == "blue") {
+     cat("Yay for pants!\n")
+ }
>
> for (i in c(1, 2, 3)) {
+     if (i < 2) {
+         cat(i, "\n")
+     }
+ }
1
>
```

Boolean operators

The conditionals usually contain Boolean operators:

- You probably know what "<" and ">" are.
- The "==" is the equivalence test ("is this equal to this?").
- The "!=" is the "not equal to" test.
- The "&" symbol is the "AND" operator.
- The "|" symbol is the "OR" operator.
- If you have multiple tests in the same line, separate them with brackets for clarity.
- Boolean operators show up in if statements and while loops.

```
>
> 2 < 3
[1] TRUE
>
> "arms" != "legs"
[1] TRUE
>
> (2 < 3) & ("arms" != "legs")
[1] TRUE
>
> (2 < 3) & ("arms" == "legs")
[1] FALSE
>
> (2 < 3) | ("arms" == "legs")
[1] TRUE
>
```

Conditionals, continued

You can add some optional structure to your 'if' statement, such as including an 'else':

- If the condition is TRUE, then the commands in the first code block are executed.
- If the condition is FALSE, then the code block associated with the 'else' is executed.
- The 'else' statement is optional.
- If you include it, the 'else' statement must be immediately after the 'if' code block's }.

```
> if ("pants" == "blue") {
+    cat("Yay for pants!\n")
+ } else {
+    cat("Boo for pants!\n")
+ }
Boo for pants!
```

```
>
```

```
> for (i in c(1, 2, 3)) {
+    if (i < 2) {
+       cat(i, "\n")
+    } else {
+       cat("Too big!\n")
+    }
+ }
1
Too big!
Too big!
```

Conditionals, continued more

Note that the part inside the parentheses of a while loop or an if statement does not need to be a conditional. It can be anything that returns a boolean, including functions.

```
>
> if (TRUE) {
+     cat("I am TRUE!\n")
+ }
I am TRUE!
>
> is.character(1)
[1] FALSE
>
> if (is.character(1)) {
+     cat("I am a character!\n")
+ } else {
+     cat("I am NOT a character!\n")
+ }
I am NOT a character!
>
```

Conditionals, continued even more

But there's more!

- The "if" statement can also contain the usual "else if" option seen in other languages.
- This allows you to combine several conditionals into a single giant if statement.
- As soon as a positive conditional is encountered the program runs the associated code block, and then jumps to the end of the if statement.
- If no positive conditional is encountered,
  - if there is an else statement, the else statement code block is executed.
  - if there is no else statement (since it is optional), then nothing is done.

```
> for (i in list(1, 2, 3, 4, 5)) {
+     if (i < 2) {
+         cat(i, "\n")
+     } else if (i == 3) {
+         cat("Go 3!\n")
+     } else if (i > 3) {
+         cat(i, "\n")
+     } else {
+         cat("no good!\n")
+     }
+ }
1
no good!
Go 3!
4
5
>
```

Use vectors instead of loops!

Whenver possible, operate on whole vectors ('vectorization') rather than looping and operating one element at a time.

- Your computer has built-in abilities which speed up vectorized calculations.
- The difference, especially on large amounts of data, can be enormous.

```
> a <- 3:7
> b <- 6:10
> e <- rep(0,5)
>
> # do this!
> d <- a * b
>
> # don't do this!
> for (i in 1:5) {
+     e[i] <- a[i] * b[i]
+ }
>
> d
[1] 18 28 40 54 70
> e
[1] 18 28 40 54 70
>
```

Factors

R has other data types you may run into. One of them is 'factors':

- 'factors' are categorical variables, and as such take on discrete values.
- OrchardSprays is a built-in dataset.
- The 'levels' of a factor are the possible, integer, values the variable can take.

```
>
> OrchardSprays$treatment
[1] D E B H G F C A C B H D E A F G F H A E D C G B
[25] H A E C F G B D E D G A C B H F A C F G B D E H
[49] B G C F A H D E G F D B H E A C
Levels: A B C D E F G H
>
> str(OrchardSprays$treatment)
Factor w/ 8 levels "A", "B", "C", "D", ...:  4 5 2 ...
>
> factor(c("Agree", "Agree", "Disagree", "Indifferent"))
[1] Agree Agree Disagree Indifferent
Levels: Agree Disagree Indifferent
>
```

Tables

Tables are used to summarize results:

- Give the 'table' command a vector, or a factor, and it will summarize the frequency of values.
- You can use the "names" function to get the various column values.
- Individual entries can be accessed using the double square brackets.

```
> table(OrchardSprays$treatment)
 A  B  C  D  E  F  G  H
 8  8  8  8  8  8  8  8
>
> my.table <- table(c("A", "A", "B", "A", "B", "B",
+ "C", "A", "C"))
>
> my.table
 A  B  C
 4  3  2
>
> names(my.table)
[1] "A" "B" "C"
>
> my.table[[3]]
[1] 2
>
```

Tables, continued

Tables can also be used to do frequency analyses on multi-dimentional data.

```
>
> a <- c("Sometimes", "Never", "Never", "Always",
+ "Always", "Sometimes", "Sometimes", "Never")
>
> b <- c("Maybe", "Maybe", "Yes", "Maybe",
+        "Maybe", "No", "Yes", "No")
>
> table(a, b)
             b
a             Maybe No Yes
   Always         2  0   0
   Never          1  1   1
   Sometimes      1  1   1
>
```