Introduction to Computational BioStatistics with R: introduction to R

Erik Spence

SciNet HPC Consortium

15 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, "Intro to R".

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

Today's class

Today we will visit the following topics:

- Introduction to R.
- Getting R started.
- Primitive data types.
- Container types.

The point of today's class is to introduce you to R, and review the major data types. Please stop me if you have a question.

Computer programs

Let's start at the very very beginning: what is a computer program?

- A computer program is a set of instructions which tell the computer what to do.
- Generally speaking, for scientific computing, you define variables, which contain your data, and perform operations on those variables to do the calculations which you need.
- You also define your own personal functions, which you will then re-use over and over with different parameters (different arguments).
- There are about a bazillion programming languages out there, each with their own strengths and weaknesses.

As you know, we will begin by using R as the programming language in this class.

R history

R has been around for a while, and is well-developed:

- Introduced in 1996 as an evolution of the S language.
- R is designed for exploring and analysing data.
- Home page: `http://www.r-project.org`.
- Community packages are stored at CRAN - Comprehensive R Archive Network.
- Bioinformatics packages are also stored at `http://bioconductor.org`.
- A new full version of R is released each year. We're currently on release 4.0.2.

About R

Some important things to know about R:

- R is a scripting language (like the Linux shell), meaning an interpreter executes commands one line at a time (not a compiled language).
- R can be used interactively, with or without an IDE (RStudio).
- R can also be used non-interactively, run through scripts.
- R has a large repository of community packages.
- R is all about data analysis: it is not a general purpose language.
  - Several important features (numerics, visualization) are baked into the language, not add-ons.
  - Not as useful outside of number crunching.
- R is designed with interactive data exploration in mind.
  - Lots of surprising things "just work" interactively.
  - But this design can make it a little difficult to debug large non-interactive programs.

Starting R

Start R now. This means opening a terminal and typing "R" (Macs), or double-clicking on the R symbol (Windows).

Raise your hand if you don't think it's working. You are welcome to follow along by entering the commands on the slides, and playing with the output.

Alternatively, there are several graphical R interfaces available. These are handy, but we generally don't recommend them as in some cases they have serious drawbacks. However, you are welcome to use them if you like.

R data types: numeric

Once you start your session you will get an interactive prompt:

- This is a different prompt from the "shell" prompt. This is the "R" prompt.
- As you enter commands the interpreter interprets them.
- The "<-" symbol is the assignment operator, thus creating a variable.
- As we saw with the Linux shell, variables hold values that we want to use later.
- You should read "a <- 1" as "a is assigned the value 1".
- If you just type the name of the variable, and hit Enter, the value of the variable will be printed.
- The "[1]" is the index of the answer.

```
> a <- 1
> b <- 1.73
>
> a
[1] 1
>
> b
[1] 1.73
>
> a + b
[1] 2.73
>
> d <- a - b
>
> d
[1] -0.73
>
```

Assigning values, an aside

As said on the previous slide:

- You should read ”a <- 1” as ”a is assigned the value 1”.
- Whatever is on the right-hand side of the ”<-” is evaluated first.
- If there is a variable on the right-hand side of the ”<-”, the value of the variable is put into the calculation, before the value is assigned to the left-hand side.
- In this case we are over-writing the previous value of the ’a’ variable with a new value.

If you don’t understand what’s going on on this slide, please raise your hand.

```
>
> a <- 1
>
> a
[1] 1
>
> a <- a + 1
>
> a
[1] 2
>
> a <- (a / 2) + 3
>
> a
[1] 4
>
```

R has functions

Like all languages, R has built-in functions:

- The "typeof" function "returns" the 'type' of the "argument".
- The "argument" of the function is the thing in the brackets.
- The "return value" of the function is the value which the function "gives back" when it's finished running.
- If a returned value is not assigned to a variable R will just print it.
- If the function takes multiple arguments, they are separated by commas.

```
>
> a
[1] 4
>
> typeof(a)
[1] "double"
>
> b <- typeof(a)
>
> b
[1] "double"
>
> is.numeric(a)
[1] TRUE
>
```

Function return values

Function return values are either assigned to a variable, or printed!

- If a returned value is not assigned to a variable R will just print it.
- This is true whether you are running functions on the R command line, or in a script.
- If your scripts start randomly printing things out, it's likely that you're calling functions which return a value, and the value is not getting assigned to a variable.
- Note that functions are not required to return something.

```
>
> a
[1] 4
>
> typeof(a)
[1] "double"
>
> b <- typeof(a)
>
> b
[1] "double"
>
> is.numeric(a)
[1] TRUE
>
```

Help!

But what if you don't know how to use the function, or don't know the optional arguments? You can use the help function, which is also accessed using '?':

```
>
> help(sum)
:
>
> ? sum
:
>
```

Press 'q' to exit the help page (on a Linux system).

Types of programming

The type of programming you are doing is called "interactive programming." As mentioned already, R uses an interpreter to interpret the commands you enter. As such, it's possible to interact with R directly.

R can also be run by writing a "script". This is a set of commands, contained in a text file, which the interpreter reads and executes, line by line, as if you were typing the commands in at the R command line. We will revisit this in a later class; these are the types of programs you will submit for your homework.

There is a third type of programming, whereby the "script" is "compiled" into an executable program. R is not a compiled language, it's an interpreted language, and as such is not capable of being run this way.

R data types: strings

R has the usual non-numeric types as well:

- Values in quotes are called "strings" (collections of characters).
- R accepts single or double quotes.
- "paste" converts the inputs to strings (if it's not already), concatenates them, and returns them.
- "cat" prints the arguments to the screen. It also needs a newline character (\n).
- Notice that these are built-in functions.

```
> e <- "hello"
> g <- 'world'
>
> mystring <- paste(e,g)
>
> print(mystring)
[1] "hello world"
>
> cat(e, g, '\n')
hello world
>
> typeof(e)
[1] "character"
>
> e + g
Error in e + g :  non-numeric argument to
binary operator
```

Strings versus variable names

Don't get confused between variables and strings:

- If a variable is assigned the value of a string, it is assigned the value of a string. The string has nothing to do with any similarly-named variables.
- When a variable is assigned another variable, it's assigned the variable's value.

```
>
> a <- 1
>
> b <- "a"
>
> b
[1] "a"
>
> d <- a
>
> d
[1] 1
>
```

R data types: booleans

R has the usual non-numeric types as well:

- TRUE and FALSE are called "boolean" values (sometimes called "logical").
- The "!" is the NOT operator. It returns the opposite of the argument.
- If you're feeling clever, you can do math on booleans.

```
> f <- FALSE
>
> f
[1] FALSE
>
> !f
[1] TRUE
>
> typeof(f)
[1] "logical"
>
> is.logical(f)
[1] TRUE
>
> TRUE + TRUE
[1] 2
>
```

R data types, continued

We've seen R's primitive types:

- "numeric": floating types (double precision)
- logicals (booleans, meaning TRUE/FALSE)
- character strings

Some notes about some R features:

- In R the idiomatic assignment operator is "<-".
- logical literals are shouty (TRUE/FALSE, or T/F).
- Variables can have periods in their names.
- Comments are started with the # symbol (as with Python and bash).

Note that R is case sensitive ("A" is not the same as "a").

Dynamic typing

R uses dynamic variable typing.

- This means that you can re-use a variable over and over again.
- Compiled languages do not have this feature.
- This is one of the reasons why interpreted languages are much slower than compiled languages.
- It's possible to re-define already-existing functions which come with R. Don't.

```
>
> a <- 1
>
> a
[1] 1
>
> a <- "pants"
>
> a
[1] "pants"
>
> a <- T
>
> a
[1] TRUE
>
```

R lists

Lists are the most basic "container" data type in R:

- The "list" function will generate a list from the inputs.
- Lists can be of mixed type.
- "pi" is a builtin variable.
- "str" stands for "structure". It gives a description of the argument.

```
>
> l <- list(a, b, e, f, g, pi)
>
> str(l)
List of 6
$ : logi TRUE
$ : chr "a"
$ : chr "hello"
$ : logi FALSE
$ : chr "world"
$ : num 3.14
>
> is.list(l)
[1] TRUE
>
```

R lists, continued

The values of lists can be of various types, including other lists.

- Accessing individual items in a list is done with [[ ]].
- The number in the double square brackets is called the "index".
- Indexing starts at 1, as with most scientific computing languages.
- In R, the "start:finish" notation returns a sequence running from start to finish, inclusive.

```
>
> l[[6]]
[1] 3.141593
>
> l[1:3]
[[1]]
[1] TRUE

[[2]]
[1] "a"

[[3]]
[1] "hello"
>
```

R lists, continued

The values of lists can be of various types - including lists. Note:

- Lists can also be created with the 'c' command ('combine').
- Indexing individual items in a list is done with [[ ]].
- Indexing starts at 1, as with most scientific computing languages.
- Slicing is done with [start:finish], and the last item is included.
- Note that [[-1]] will return an error message.
- What does slicing with a negative number do - eg, l[-1]?

R named lists

- Named lists allow you to access elements by name, rather than by index.
- If you don't finish your line in R, but hit enter, it will display the "+" symbol, indicating that it's waiting for more input.
- You can access pieces of a named list with the "$".
- The "names" function returns the names of a named list, data frame, etc.

```
>
> named.list <- list(value = 5,
+ word = "text", number = 7.3)
> str(named.list)
List of 3
 $ value : num 5
 $ word  : chr "text"
 $ number: num 7.3
>
> named.list$value
[1] 5
> named.list[["number"]]
[1] 7.3
> names(named.list)
[1] "value"   "word"   "number"
>
```

Enough to get started

There's obviously a lot more to learn about using R. In particular, there are 2 major container types, vectors and data frames, which we'll cover next class. Nonetheless, this is enough functionality to get you started.