Evolutionary Behavioural Ecology: practical on Modeling Evolution

Bram Kuijper

2021-09-23

Contents

Chapter 1

Practicals on evolutionary models using R

During the following two practicals, we will try to implement some basic evolutionary models in R. The first practical will focus on the evolution of aggressive vs evasive behaviour. The second practical will focus on the evolution of cooperation and conflict. I am certainly no R expert myself, so this is as much a learning process for you as it is for me.

1.1 Learning objectives

After this practical, you…

- should be able to use R to implement short programs
- understand how we can develop simple population genetics models in R
- should be able to explain why models necessarily include many limiting assumptions
- should be able to inspect your model and draw conclusions from it.

1.2 Opening the R environment

To provide you with a standardized environment during this practical session, you will have to log into to the University of Exeter's Rstudio server. Of course, you are welcome to try to do the same using your installation of R on your own computer, but we might run into problems regarding the installation of some packages, hence we ask you to stick to the rstudio server during the practical.

Chapter 2

A brief R programming crash course

To get started with making our own evolutionary model, we need to make a simple program in R that tracks an evolving population.

To this end, you need to know something about the ingredients that simple programs use, namely variables, vectors/sequences and for-loops. Some of this may have already featured in your statistics or introduction to data science modules, but at this stage, getting some repetition about the sometimes bamboozling inner workings of R certainly does not hurt. Even the most field-oriented among you will use R extensively during your research projects, so the more exposure you can get to this environment, the better.

Try to type along with the examples below and see whether they work. Give me a shout if something looks odd.

2.1 Variables

A variable provides us with a named storage of data, allowing you to retrieve that bit of data when you want it. A variable in R can contain any type of data (e.g., a single number, a string of text, a vector of multiple values, etc).

We assign data to a variable using a `=` or `<-` operator (both can be used interchangeably) For example:

```r
# create a variable named var.1 and assign a value to it using the <- operator
var.1 <- 25000

# create a variable named var.2 and assign a vector, c(), of numbers to it using the = operator
var.2 = c(0,1,9.5,200)
```

```
# a bit of text, surrounded by single '' or double "" quotes
some.text <- "Swanpool beach"
```

2.1.1 Naming variables

Variables can have any name, but cannot start with a number and can only contain dot `.` and underscore `_` characters as special characters.

```
# this fails
2times_song_release_year <- 1999
## Error: <text>:2:2: unexpected symbol
## 1: # this fails
## 2: 2times_song_release_year
##      ^
```

```
# this is fine
release_year_2times <- 1999
```

```
# this fails too
release%year <- 1999
## Error: <text>:2:8: unexpected input
## 1: # this fails too
## 2: release%year <- 1999
##            ^
```

2.1.2 Changing the value stored by variables

You can easily change values of variables, simply by assigning them a new one! The new value can be of a completely different type.

```
# print the value of a variable, showing its current value
print(var.1)
## [1] 25000
# change the value
var.1 <- "some text instead!"

# print the new value
print(var.1)
## [1] "some text instead!"
```

You can always check the type of your variable using the `typeof()` function:

```
typeof(var.1)
## [1] "character"
typeof(var.2)
## [1] "double"
```

As you see, `typeof` does not tell you `var.2` is a vector of multiple values, but just focuses on the type of the values contained in them (in this case a `double`, which is another word for floating point value. If you want to find out if something is a vector of values, you have to look at the length of the variable (see below).

2.2 Vectors and element selection

As you have seen in the example of `var.2` above, variables can point to collections of multiple values (vectors). To create a vector, we use the `c()` function. We can subsequently get values of (groups of) individual elements by using the `[[]]` operator:

```
# 1. create a vector of character elements using c()
animals <- c("Opossum","Dog","Pallas's leaf warbler")

# 2. select a single element using the double square brackets [[]]
print(animals[[3]])
## [1] "Pallas's leaf warbler"
# 3. obtain a selection of
# multiple values using [] and
# a vector of elements you want to select
print(animals[c(2,3)])
## [1] "Dog"                   "Pallas's leaf warbler"
# 4. assign a new value to an existing element
animals[[1]] <- "Click beetle"
print(animals)
## [1] "Click beetle"          "Dog"                   "Pallas's leaf warbler"
```

2.2.1 Exercises

- What happens to the vector `animals` if you assign the name of your favorite animal (in quotes `""`, because text) to element `40`?
- Assign the number 0.5 to element `10` of the `animals` vector. Try to multiple this value by 10, by typing

```
animals[[10]]*10
```

what happens? Use `typeof(animals[[10]])` to see what is going on.

2.3 Information about your vector

You can obtain information about your vector by using a range of different functions, such as

- `sum()`: sums all values (only works with numbers obvz)

- `unique()` lists all unique values
- `sort()` sorts values. and there are lots more.

Most often, however, you will use the `length()` function to obtain its size:

```
length(x=animals)
## [1] 40
```

2.3.1 Exercise

Using the standard `iris` dataset (available within in `R` by typing `iris` on the command line), find out the number of unique values for the column `Sepal.Width`. If you see `iris` but don't know how you can access the column, try to search for it on the web.

2.4 Sequences

Vectors are also used to contain sequences of numbers, which can either be generated with the from-to operator `:` or with the more explicit `seq()` function:

```
# sequence from 1 to 10
some.seq <- 1:10

# sequence from 10 to 1
reverse.seq <- 10:1

# same, but using the sequence in some.seq and the rev() 'reverse' function:
reverse.seq.2 <- rev(some.seq)

# sequence from 10 to 1, using the seq() function, with a negative increment
another.seq <- seq(from=10,to=1,by=-1)

# the seq() function is great if you want to take bigger steps:
steps.of.5 <- seq(from=3,to=29,by=5)
```

We can also make vectors using repetitions of values, through `rep()`:

```
# a vector of 300 zeros
only.zeros <- rep(x=0,times=300)

# a vector of 0,1 values, repeated until a length of 13 is achieved
zero.one <- rep(x=c(0,1),length.out=13)
```

2.5 The important bit: `for` loops

The point about programming is that you want to repeat actions multiple times. To this end, R offers you two different looping constructs: the `while()` loop and the `for` loop.

In this practical, we focus on the `for` loop as it is more commonly used than `while()`. Best to begin by an example:

```
# you need some vector of something
# otherwise there will
# be nothing to loop over
# Here, I use a vector of numbers from 5 up to (& including) 10
some.vec <- 5:10

# component 1: the for statement within
# parentheses ()
for (iterator in some.vec)
{
    # component 2: the for body within curly braces {}
    # in which tasks
    # are performed
    # repeatedly for each value of iterator
    print(iterator)

    # I am also printing something else
    # for the sake of illustration
    print("Some text.")
}
## [1] 5
## [1] "Some text."
## [1] 6
## [1] "Some text."
## [1] 7
## [1] "Some text."
## [1] 8
## [1] "Some text."
## [1] 9
## [1] "Some text."
## [1] 10
## [1] "Some text."
```

From the printed output, you see that the `iterator` gets a new value every time the loop goes round. The message `"Some text"` is also repeatedly printed, but this value does not change.

2.5.1 Dissecting the for loop

The loop starts with the keyword `for`, followed by parentheses `()`. Within those parentheses, we always start by declaring a new variable, which – in this case – I have called `iterator`, but you could have given it *any* valid name, like `index` or `boring.practical`.

Each time the `for` loop goes round, the `iterator` variable gets assigned a new value. Which value? An element from a vector, which we have called here `some.vec`. The `in` keyword in between `iterator` and `some.vec` performs the assigning of these consecutive elements from `some.vec` to `iterator`.

Hence, the first time the loop goes round, `iterator` receives the value of `5`, as this is the first element of `some.vec`. Subsequently, the `for` loop enters the bit in between curly braces `{}`. The code in this part will be executed every time `iterator` gets a new value. In this case, the code is `print(iterator)`, meaning it prints the *current* value of the `iterator` variable (which is `5`) is printed to the screen. After this the subsequent `print()` statement is executed, which prints `"Some text"` to the screen.

Next, the loop goes round again. Now the value of `iterator` will be 6. Again, `6` is printed by the `print()` statements in the `{}`-part and this goes on until all elements from `some.vec` have been assigned to `iterator`. Hence, the last value printed will be `10`, after which the `for` loop exits.

2.5.2 Another example

Just to throw in another example that is slightly different, let us calculate the product of elements of two vectors and sum those products:

```r
# generate two vectors with numbers;
# vectors have the same length
vec.1 <- c(0.3,0.9,1.2,0.1,3.5)
vec.2 <- c(5,8,12,3,7)

# check whether the length is indeed the same
stopifnot(length(vec.1) == length(vec.2))

sum <- 0

# loop not over the elements of the list
# but over the element indices
# (we can do so by creating a list from
# 1 to the length of one of the vectors,
# i.e., 1,2,3,...)
for (idx in 1:length(vec.1))
{
    # use idx to obtain elements of both vectors
    # and add them to the total
    sum <- sum +
        vec.1[[idx]] * vec.2[[idx]]
}
print(sum)
## [1] 47.9
```

2.5.3 Exercises

- Can you think of why we don't we write `for (idx in vec.1)` as in the previous example?
- Could you change the code to store the sum that you obtain in each iteration in a list? I.e., that list should have the values

```
print(cumul.list)
```

```
## [1]  1.5  8.7 23.1 23.4 47.9
```

OK now that we have some tools under our belts, let's get going with evolution!!

Chapter 3

An evolutionary model of the hawk-dove game

In your previous studies you may have come across a simple evolutionary model, called the Hawk-Dove game. This influential model aims to sketch out how aggressive versus more restrained behaviours evolve when individuals compete with each other over resources.

Here we analyze a deterministic version of this model, by focusing on a population consisting of i) aggressive hawks (denoted by H) who always fight and do not retreat; and ii) timid Doves (denoted by D), who may perform some threat display, but eventually will always retreat if the other individual starts a fight.

3.1 Payoff matrices

We can summarize these interactions in a so-called payoff matrix:

$$\begin{array}{lll} & H & D \\ H \text{ encounters...} & \frac{v-c}{2} & v \\ D \text{ encounters...} & 0 & \frac{v}{2} \end{array} \qquad \begin{array}{r} (3.1) \\ (3.2) \\ (3.3) \end{array}$$

Here we are going to develop our own hawk dove model in R and investigate how it plays out, using what we call a *deterministic* model where

A key element of this model is that the fitness payoff of a strategy one plays depends on the Until the 1970s, most evolutionary models considered traits

Chapter 4

Fitness of a hawk

For the sake of simplicity, we assume that being a hawk or dove is the result of alleles at a single haploid locus. This is crazy of course, hardly any animal traits are haploid! Moreover, even fewer are chiefly the result of variation at a single locus only.

This is what typical models are about: sure, the assumptions are wrong, but deep insights in evolutionary biology typically come from overly simplistic models as these are at least understandable. The insights of the Hawk-Dove game would have been much more poorly understood if

Let p be the frequency of hawks in the population, while $1-p$ is the frequency of doves in the population. For the sake of simplicity, we are going to assume that

4.1 Exercise

- Can hawks invade in a full dove population?
- Can doves invade in a full dove population?

Chapter 5

Parts

You can add parts to organize one or more book chapters together. Parts can be inserted at the top of an .Rmd file, before the first-level chapter heading in that same file.

Add a numbered part: `# (PART) Act one {-}` (followed by `# A chapter`)

Add an unnumbered part: `# (PART\*) Act one {-}` (followed by `# A chapter`)

Add an appendix as a special kind of un-numbered part: `# (APPENDIX) Other stuff {-}` (followed by `# A chapter`). Chapters in an appendix are prepended with letters instead of numbers.

Chapter 6

Footnotes and citations

6.1 Footnotes

Footnotes are put inside the square brackets after a caret `^[]`. Like this one [1].

6.2 Citations

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package [Xie, 2021] (check out the last code chunk in index.Rmd to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** [Xie, 2015] (this citation was added manually in an external file book.bib). Note that the `.bib` files need to be listed in the index.Rmd with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: https://rstudio.github.io/visual-markdown-editing/#/citations

[1]This is a footnote.

Chapter 7

Blocks

7.1 Equations

Here is an equation.

$$f\left(k\right) = \binom{n}{k} p^k\left(1-p\right)^{n-k} \tag{7.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (7.1).

7.2 Theorems and proofs

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 7.1.

Theorem 7.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* ***other*** *two sides, we have*

$$a^2 + b^2 = c^2$$

Read more here https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html.

7.3 Callout blocks

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

Chapter 8

Sharing your book

8.1 Publishing

HTML books can be published online, see: https://bookdown.org/yihui/bookd own/publishing.html

8.2 404 pages

By default, users will be directed to a 404 page if they try to access a webpage that cannot be found. If you'd like to customize your 404 page instead of using the default, you may add either a `_404.Rmd` or `_404.md` file to your project root and use code and/or Markdown syntax.

8.3 Metadata for sharing

Bookdown HTML books will provide HTML metadata for social sharing on platforms like Twitter, Facebook, and LinkedIn, using information you provide in the `index.Rmd` YAML. To setup, set the `url` for your book and the path to your `cover-image` file. Your book's `title` and `description` are also used.

This `gitbook` uses the same social sharing data across all chapters in your book-all links shared will look the same.

Specify your book's source repository on GitHub using the `edit` key under the configuration options in the `_output.yml` file, which allows users to suggest an edit by linking to a chapter's source file.

Read more about the features of this output format here:

https://pkgs.rstudio.com/bookdown/reference/gitbook.html

Or use:

```
?bookdown::gitbook
```

Bibliography

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.org/knitr/. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2021. URL https://CRAN.R-project.org/package=bookdown. R package version 0.24.