Lecture 2
A First Glimpse of R and of Population Growth Models

Dr. Ido Filin

`ifilin@univ.haifa.ac.il`

29 October 2012

Outline

Why R?

- R is a software mainly designed for statistical analysis and high-quality (publication-quality) graphics.

- R is **free software**.
  - Source code is freely available for all, and all can contribute to development of R.
  - Free of charge to download and use – a donation is encouraged, either to the R Foundation or to the FSF – the Free Software Foundation – both are non-profit organizations.
  - Free of any irrelevant commercial interests – as academia should be.

- Written and constantly expanded **by scientists for scientists**.

- Packages/expansions for any type of statistical analysis, as well as discipline-specific packages (for bioinformatics, social sciences, physics, ecology, etc.).

Why R?

- Freely available distributions for Linux, Mac OS X, Windows.

- Just go to http://www.r-project.org

- Downloads, manuals, expansion packages etc.

- In R we can do mathematical modeling, data analysis, statistical tests, and produce publication-quality figures – all using the same tool.

Basic calculations

1. > 0 [Enter] ⟶ [1] 0
2. > 100 [Enter] ⟶ [1] 100
3. > 1e+2 [Enter] ⟶ [1] 100
4. > 2.5e+3 [Enter] ⟶ [1] 2500
5. > 5.43e-1 [Enter] ⟶ [1] 0.543
   (the e-notation stands for multiplying by powers of 10)

6. > 1 + 2 [Enter] ⟶ [1] 3
7. > 1.5 + 2.5 [Enter] ⟶ [1] 4
8. > -1.5 + 3.9 [Enter] ⟶ [1] 2.4
9. > 1e+2 - 1e+1 + 1e+0 [Enter] ⟶ [1] 91
10. > 1 * 2 * 3 * 4 [Enter] ⟶ [1] 24
11. > 1 * 2 * 3 / 4 [Enter] ⟶ [1] 1.5
    (use [↑] [↓] to scroll through previous commands and edit them)

Basic calculations

12 > 1 * 2 * ⟦Enter⟧ ⟶ + (and waiting for further typing)
(if R detects that a command is incomplete it will request you to complete the command by changing the prompt sign from **>** to **+**)

13 + 3 / ⟦Enter⟧ ⟶ + (again, because the command is not yet complete)

14 + 4 ⟦Enter⟧ ⟶ [1] 1.5

15 Press ⟦Esc⟧ or ⟦Ctrl⟧ + C (depending on operating system) to cancel the whole calculation at any time, while in + prompt-mode.

16 > 9 ^ 2 ⟦Enter⟧ ⟶ [1] 81

17 > 9 ^ 0.5 ⟦Enter⟧ ⟶ [1] 3

18 > 2 ^ 3 + 5 ^ 4 ⟦Enter⟧ ⟶ [1] 633

19 > 4 ^ (-1) ⟦Enter⟧ ⟶ [1] 0.25

Basic calculations

19 > 5 + 3 * 4 ^ 2   ⌯Enter⌯ ⟶ [1] 53

20 > (5 + 3) * 4 ^ 2   ⌯Enter⌯ ⟶ [1] 128

21 > ((5 + 3) * 4) ^ 2   ⌯Enter⌯ ⟶ [1] 1024

R observes the regular order of operations and how it is modified with use of parentheses.

Basic use of variables

In a model we require state variables $\longrightarrow$ R variables.

- If we define a variable named `foo`, we can always refer to the name `foo` in order to get the current value it holds.
- The value of `foo` may change over time (while our R program is running), but we can always recall its current value by referring to the name `foo`.
- A valid variable name can be any combination of alphanumeric signs, that does not begin with a number (digit).
- Valid names: `x y z N foo myDensityVar OranGe tOmATo N0 N1 N2 Pop9Size2012`
- Invalid names: `1myVar 2012popSize Pop?Size var()name pop-size pop;size`
- R is **case-sensitive**: `foo Foo fOo PopSize popsize` are all different variables.

Basic use of variables

The most basic thing you can do with a variable is assign a value to it.

1. > x = 3 [Enter] (Note that assignment commands do not display result on the screen)

2. > x [Enter] ⟶ [1] 3

3. > print(x) [Enter] ⟶ [1] 3 (this is the preferred way to display a value of a variable)

   An alternative way to assign values

4. > x <- 9 [Enter]

5. > 20 -> x [Enter]
   This is the preferred way to write assignments in R.

   You can assign to a variable the result of any expression.

6. > (4 + 5) * 2 ^ 3 -> x [Enter]

7. > y <- x + 10 [Enter]

Basic use of variables

Your first population growth model.

1 > `N <- 10` Enter (A initial population size of 10)

2 > `N + 1 -> N ; print(N)` Enter ⟶ [1] 11

- Note that we assign the result of the expression `N + 1` back to `N` itself.
- This is how we update the current value of a variable based on its previous value.
- Also, note that we can write several commands in a single command-line, by separating the commands with a semi-colon.

3 Use ↑ to display and run the last command-line again and again. Observe how the value of `N` increases.

4 Start again with `N=10`, but this time change the command-line such that population size would double itself each time-step.

Vectors of values

Often we want to observe a series of values – a series of population sizes over time.

In such cases we need to assign a series of values into a variable

In R, such variables are called **vectors**.

1 > N <- numeric(5) ; print(N) ⟦Enter⟧ ⟶ [1] 0 0 0 0 0
(Assigning a blank vector of zeros)

2 > N <- c(10, -1.37, pi, 2012) ; print(N) ⟦Enter⟧
⟶ [1] -1.000000 -1.370000 3.141593 2012.000000
(Assigning a custom vector using the c(. . .) notation)

3 popSize <- c(0, 2, N, c(100, 1000, 10000)) ;
print(popSize) ⟦Enter⟧
(You can embed vectors within other vectors using c(. . .) notation)

Vectors of values

A third way to assign vectors is using **sequences**, with the colon sign.

④ >   N <- 1 :   10 ; print(N)   ⟦Enter⟧

⑤ >   N <- 2.43 :   8 ; print(N)   ⟦Enter⟧

⑥ >   N <- 20.5 :   -19.5 ; print(N)   ⟦Enter⟧

⑦ >   startVal <- 17 ; endVal <- -46 ;
   N <- startVal :   endVal ; print(N)   ⟦Enter⟧

Can you guess now what is the purpose of the bracketed numbers on the left side of printed values?

Vectors of values

Often you need to access or modify specific elements with the vectors.

We access vector elements using **brackets**;

8 >   N <- 10:20 print(N[5])   `Enter`

9 >   print(N[11])   `Enter`

10 >   print(N[100])   `Enter` $\longrightarrow$ [1] NA

NA = **N**ot **A**vailable. A value used as a placeholder for missing data or otherwise undefined values.

11 >   N[3] <- 1000 ; print(N)   `Enter`

12 >   N[4] <- N[3] + 100 ; print(N)   `Enter`

13 >   N[5:8] <- N[4] * 2 ; print(N)   `Enter`

14 >   N[9:11] <- N[9:11] * 10 ; print(N)   `Enter`

15 >   N[9:11] <- N[6:11] * 10 ; print(N)   `Enter`   **Error**: the two sub-vectors are not the same size.

Vectors of values

A more sophisticated population growth model.

16 >   `N <- numeric(5) ; print(N)`   [Enter]

17 >   `N[1] <- 1 ; print(N)`   [Enter]
Initial population size is 1.

18 >   `N[2] <- 2 * N[1] ; print(N)`   [Enter]
Population doubles each time-step.

19 >   `N[3] <- 2 * N[2] ; print(N)`   [Enter]

20 >   `2 * N[3] -> N[4] ; print(N)`   [Enter]

21 >   `N[5] <- 2 * N[4] ; print(N)`   [Enter]

Plot a graph of population size over time:

22 >   `Time <- 0:4 ; plot( Time , N )`   [Enter]

R - scripts

We do not want to start writing each model from scratch
every week.
We would like to save what we have done so far, and build
upon it next time.
R-scripts are simple text files that contain a list of R
commands. Basically they are computer programs.

1. Open New Script under the File menu.
2. This would create a blank window.

   First, we need to name and save the script file.

3. Click on the blank script window.
4. Then choose Save as. . . from the File menu.
5. Go to the directory you wish to save the file in, or create
   a new directory for your R-scripts and then choose it.
6. Once inside the directory, name the file hello.r and click
   on the save button.

R - scripts

7. In the blank script window type print("Hello World!")

8. Now save the file using Save option form File menu or pressing ⌨Ctrl + S.

9. Go back to the console / command window.

10. Type getwd() ⌨Enter
    This gives you the current working directory of R.
    You need change the working directory to the directory which contains the hello.r file.

11. In the File menu choose Change dir.

12. Then change the directory to the one in which you saved the hello.r file.

13. Verify that the working directory has changed by using the getwd() command again.

R - scripts

14 To run the hello.r program type
   `source("hello.r")` ⟦Enter⟧

15 Another option is to choose the script window again and choose Run all from the Edit menu.

16 A third option is to run only some selected text within the script file or run only the line in which the cursor is currently on.
   This is achieved by selecting some text with the mouse or moving the cursor to the line you want to run, and then choosing "Run line or selection" from the Edit menu, or just pressing ⟦Ctrl⟧ + R.

17 After you have run hello.r and observed the output in the command / console window, close the hello.r script window.

R - scripts

18 Create a new script and save it with the name FirstPopModel.r

19 Write the following command lines into the script file:

1 `Ninitial <- 10`

2 `lambda <- 2`

3 `N <- numeric(10)`

4 `Time <- 0:9`

5 `N[1] <- Ninitial`

6 `N[2] <- lambda * N[1]`

7 `N[3] <- lambda * N[2]`

8 `N[4] <- lambda * N[3]`

9 . . . (i.e., repeat the previous style of commands, each time incrementing the indexes)

10 `N[10] <- lambda * N[9]`

11 `print(N)`

12 `plot( Time, N )`

20 Save the new commands that you added to the script file by pressing [Ctrl] + S

R - scripts

21 Now run the script file FirstPopModel.r
Make sure you are in the right directory by switching back to the console window and using the `getwd()` command and Change dir from File menu if needed.
1mm

22 Observe the output in the console window and the graphical output.

23 Congratulations! You have now written and run your first population growth model in R.

Parameters vs. Variables

The recursion relation for the discrete-time growth model we just wrote is

$$N_{t+1} = \lambda \, N_t$$

- Obviously, N is the **variable** in this model
  1. Its value changes over time.
  2. We can attach a time index ($1, 2, \ldots, t, t+1$, etc.) to each value .

- The value of λ, however, is fixed – does not change over time.

- λ is a **parameter** in this model – it participates in determining the dynamics of N, but its own value does not change over time.

Parameters vs. Variables

$$N_{t+1} = \lambda \ N_t$$

- The value of λ may be different in different populations or species or in different circumstances (different climate, temperature, pH, nutrient level, etc.).
- But for a single occurrence of the phenomenon (population growth) it is fixed.
- We can have as many parameters or variables as needed.
- For example, in the R-script we just wrote a second parameter is `Ninitial`, the initial population size.
- We usually write the parameters at the beginning of the R-script file, so we can use them later and easily locate and change their values.
- Try to change the values of `lambda` and `Ninitial` and observe how the output of the R program changes.

Final word

A good programmer is a lazy programmer.

- We wrote an R-script, so not to write the entire model again and again each time we want to run it.

- We defined the parameter `lambda`, so we won't have to go over and change all the lines that calculate `N`, each time we want a different growth rate for the population.

- During the course, we will learn additional ways to make modeling in R easier, and the R-code more compact, so we can do a lot with just few lines of code.

Outline