BIOU9PC: Population and Community Ecology

Lab Practical week 1, part A: Exponential population growth

Modified from Luc Bussiere's introductory practical to SCIU7SR

***15 September 2016***

**Objective**

In today’s practical, we will cover some of the basics of the R programming language, and attempt some simple operations including the importing and manipulation of data. One of the most important learning outcomes will be an ability to resourcefully solve problems. Because R is a programming language (rather than a piece of commercial statistical software), its syntax is not always immediately accessible, but its language is ruthlessly logical. We will give some hints on how to solve problems, and will repeatedly expect you to be resourceful in solving problems as you progress through the module.

If you have not already done so, it may help you to read through some of the basics of the R programming environment, so that when I describe objects, vectors, and functions below, you will have a sense for what I am writing about.

**Some useful commands**

Below is a list of some (but not all) of the useful commands you will probably need to run at some stage of today’s practical. Most of these are names of functions, sometimes with suggested usage, sometimes not. I will try to highlight the growing repertoire of commands you acquire as the module advances. If you need help on usage, pass ?commandname to the console as explained in the help section below.

getwd

names

order

read.csv (or read.table)

rm(list=ls())

setwd

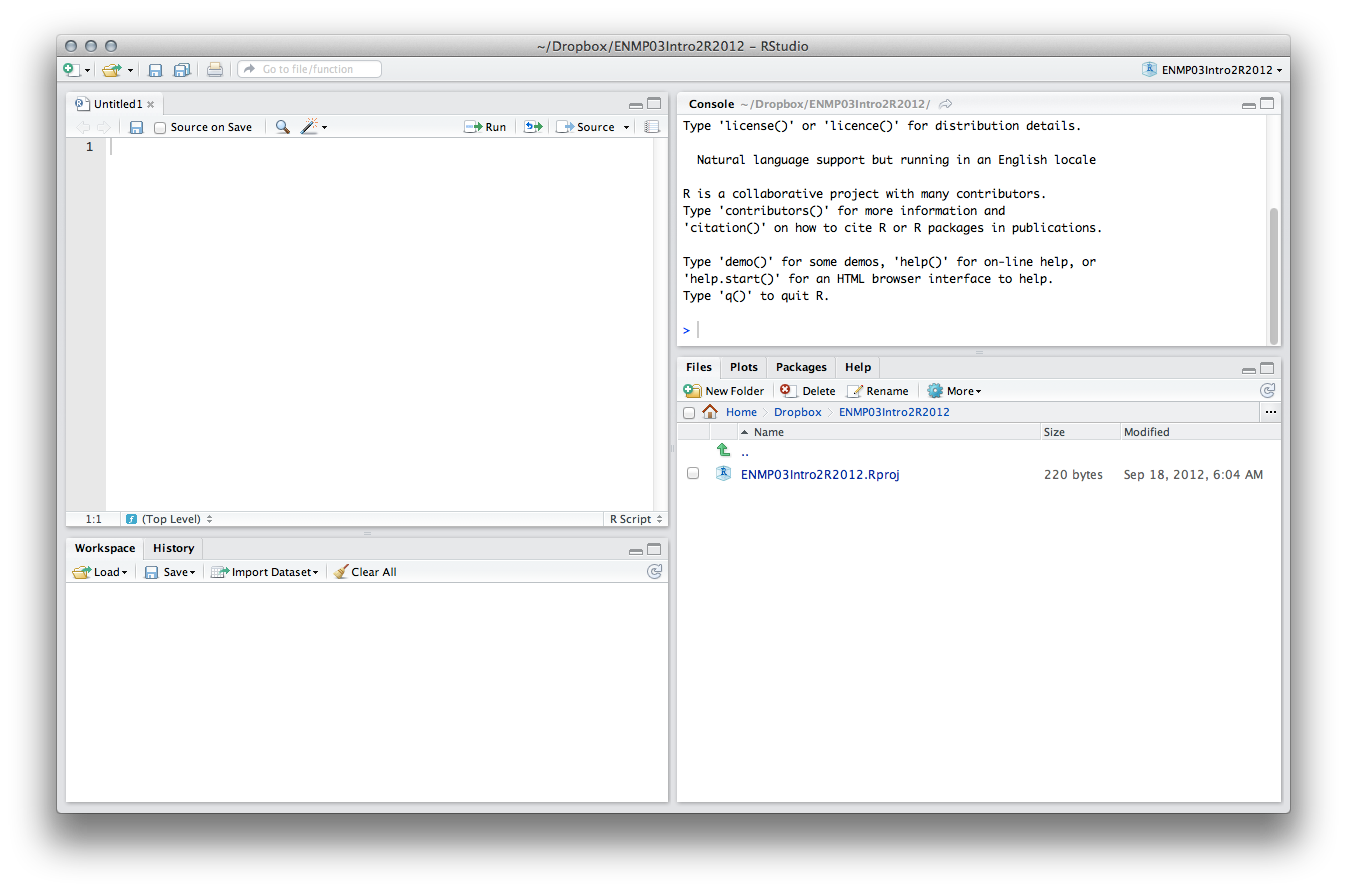
str

? and ??

**Working in RStudio**

Begin by loading the RStudio package. RStudio is a client that operates R for you, and provides several advantages over coding in the native R environment that you will discover as we proceed through the lab work.

One of the convenient features of RStudio is that it helps you organize your projects by grouping saved files together in their own directory. Your saved files will probably include data files, R scripts, any plots you generate, and possibly your workspace. Let’s start a new R project using the Project pulldown menu in the upper right hand corner of the R interface. Select “New Project”, and then use the browser to select or create a folder where you want to store the files. Choose this carefully because later you will need to know where to dump data files belonging to this project. Give the project a descriptive name (e.g., BIOU9PC.Prac1, or something snappier if you prefer), and save it. Normally, you will store all files associated with a project together. What you consider a project is up to you, but it might be one lab practical.

The RStudio interface normally has four panels, but right now you may only see three of them, because our new project has no R script associated with it. Let’s fix this using the File menu (New - > R Script). Now that you have all four panels, you may wish to arrange them so that your layout matches my own preferred layout, as illustrated below. This is achieved by adjusting preferences or options, depending on your operating system. 

In this layout, the “source” panel is top left, and the “console” panel is top right. Think of the source panel as the “keyboard” through which you will always communicate with R. It is possible to enter commands directly into the console, ***but I strongly discourage you from doing this*** most of the time, as you cannot easily save the console (nor do you normally want to). The console is instead the “display screen” for the non-graphical results of your instructions. Similarly, it is possible to execute some functions using pulldown menus, but because these are never stored in the script, I will also discourage that.

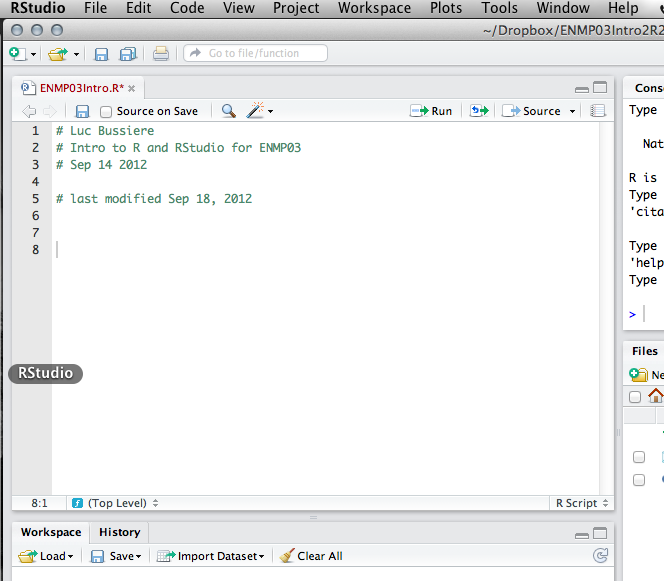
The workspace and history panels in the lower left allow you to quickly monitor the objects in your current workspace as well as the history of commands you have passed to the console.

The last panel has four tabs: “Files”, “Plots”, “Packages”, and “Help”. The “Files” tab allows you to browse your computer for files, and helps you change the working directory if necessary (although if you consistently work in projects, the working directory will always be the one you nominated or created when you set up your project). We will use it shortly. The “Plots” panel is where graphical output from your commands will be displayed. The “Packages” panel will be used to add R add-ons called packages when you need access to specialist code (we will cover this later). Finally, the “Help” panel will display R’s standardized (and probably at first rather puzzling) help pages. More on that later.

Let’s begin by focusing on the source panel. The script is the permanent record of the work you do, so before we go any further, we should try saving it. There is more than one way to do this, but the easiest is to click the floppy disk icon in the upper left of the source panel. Name your script something descriptive. It will be saved into your project folder, and you may notice that your “Files” panel now shows the two files you have created within that folder: one project file (something principally useful for you when working on this project in RStudio) and one script file (a portable set of instructions for the R console that you can share with anyone in the world (e.g., me!), regardless of what platform or client they use to operate in R). When I ask you to share your script with me, I want the file with the .R suffix, as that’s the one containing the instructions. Please don’t send me .Rproj files.

**Annotations**

Because the R script is a permanent record of your instructions, it is essential that you annotate this list very well, so that when you try to run the script again two months from now (having received requests for revision from your supervisor or a scientific reviewer), your future self will know exactly what each section of code is for. Furthermore, as you become an adept R user, you will want to share bits of code with collaborators and future students (or with your instructors in this course), and they will definitely need help to navigate your script. R has a very simple annotation methodology: anything following a hash key (#) will not be treated as code by the R console. This means you can add lines of annotations (or alternatively short annotations at the end of lines) by preceding them with a hash key. Let’s try this now by adding a few lines to the top of the script that describe the R script. I suggest your name, a description of the file, the date the script was first created, and perhaps the date it was last modified (see below). You can add empty lines anywhere in the R Script to improve its clarity or separate sections, and these will never affect the code. If you can’t find the hash key, try alt-3 (or right alt-3). Alternatively, you can select multiple lines of code and use command-shift-C (control-shift-C) to annotate (or un- annotate) blocks of text (C is for “comment”, to help you remember).

You may notice that the name of your R script has now turned red, and is followed by an asterisk. This is RStudio telling you that you have made alterations to the script file since your last save. If you are happy with the changes, save your script (using the floppy disk icon), and you will see the filename go back to black. 

**Clearing the workspace**

Because R is a programming language, one of the habits we’ll need to get into is to adhere to good coding practice that will help your machine to run effectively. It is good practice to clear your workspace before beginning work on a project, so I tend to always add the following lines to the top of my scripts, just after the information on dates. (Note, from now on I will often add bits of code directly to the text instead of always taking screenshots.)

# clear R of all objects

rm(list=ls())

Note the first line, which is annotation (and therefore not executed if it is passed to the console). The second line contains two functions nested within one another and the arguments used by the functions. We’ll cover this in due course, but for now just trust me and enter the code as written. This line clears R’s workspace to make sure you aren’t operating with any clutter in the memory.

Once you have it entered, select all of the script text, and hit command-Enter (control-Enter). You should see that the text you have selected has been passed from the script to the console, and executed. Since most of the text was annotation, R hasn’t done anything with that. The only command was to clear the workspace, which was probably already clear for you anyway. But it’s good practice to make sure!

**Getting objects into R**

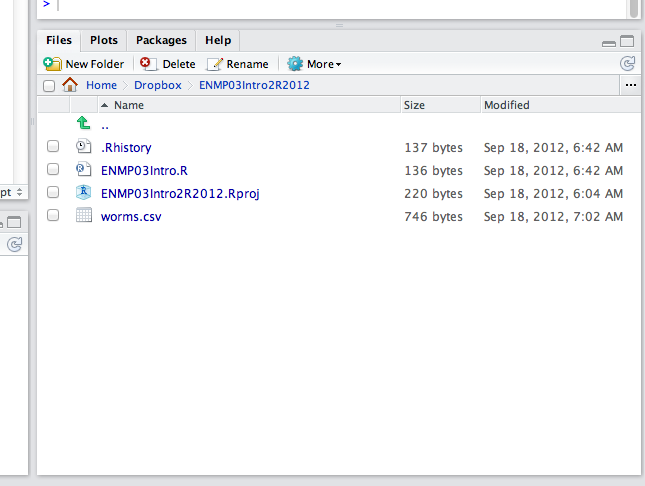
Although it is possible to enter data manually into R, you will usually import your data from files created in other packages such as MS Excel. R doesn’t read .xls files, however, so you need to convert the files to .txt or .csv before importing them. (NB I will usually use .csv files, but in some contexts it may be preferable to use tab-delimited text files, e.g., if you are collaborating with colleagues who use others languages who sometimes the comma is used as a decimal point.)

In order to import .csv data into an object called a data frame, we will use the read.csv function. In preparation, make sure to move the data file you want to work on (worms.csv, which is posted in the practicals folder on Succeed) to the folder you created for your project. If you have done this correctly, it should appear in the list of files displayed by default in the Files panel.

The read.csv file is designed specifically for csv data, so using it is relatively simple for this particular file:

# import the data; note that read.csv only works for .csv files WORMS<-read.csv("./worms.csv") (From now on I will leave the annotations up to you, but make sure you continue to annotate!)

Note the arrow that follows WORMS in that command. It is called the “**assignment operator**”, which is the operator that assigns value to an object. Remember this! It’s important to realize that R only remembers the most recent assignment value, so if you make a mistake, you can simply correct it and re-execute the code to change the value of the object.



6

Once you have passed these lines from the script to the console, you should notice a new object appear in your workspace called WORMS. You can inspect the data through the console. One option is to “call” the object itself.

# examine the object in the console

WORMS

Note: It is also possible to inspect the data by clicking on the object name in the source panel, which will open a new window that hopefully resembles a spreadsheet. But beware! Clicking on objects does not only display them; it may also try to edit them, which can have disastrous consequences for the reproducibility of your analyses, because you will be changing your data! I strongly urge you to always examine and modify objects using text commands recorded in the R script rather than the mouse.

R is case-sensitive, so if you use uppercase letters to name an object, it won’t be able to find it when written in lowercase letters. Try this to see what kind of error message is generated. Note that error messages have information within them! This will often help you decipher what exactly went wrong. From now on, we expect you to carefully read error messages, so you can try to discover what is wrong with your code on your own before asking a classmate or instructor.

You will often find that your data file is too big to conveniently examine within the console, so you may instead wish to have the console show only the first six lines of the data frame. This is achieved using the following syntax:

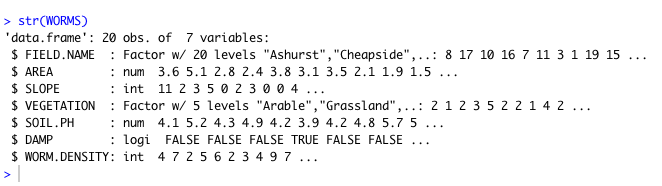
head(WORMS) Try out the following commands as well

names(WORMS)

summary(WORMS)

str(WORMS)

The last function, str(), is a very useful command for examining the structure of your data frame. For example, it is always worth looking at the structure of your imported data to check whether everything has been imported correctly. It tells you how many observations and variables there are, and it also gives you information on what kinds of ‘vectors’ you have in the data frame.



It is possible to extract a single vector from within an object by specifying its address:

WORMS$AREA

The dollar sign tells R to look within the first object, WORMS, for the vector AREA.

In addition to examining the object, you can easily perform operations on it, using operators or functions. Try the following:

mean(WORMS$AREA)

sd(WORMS$DENSITY)

The last command will generate a cryptic “NA” response. Why? What do you think NA means? Examine the data frame and the command carefully and see if you can find the error in my code. Correct the error and see if you can get a good measure of the standard deviation for worm density.

Now try the following command:

MEAN.SOIL.PH<-mean(WORMS$SOIL.PH)

Note that in the previous two commands, the output is generated and then forgotten by R. In the third, you have assigned the value to a new object (which is available for subsequent manipulation if you need it), but it is not displayed. How would you examine this new object? Do this now.

**Examining and manipulating data**

You will often want to examine a specific part of a data frame or vector, rather than the whole thing. You must first realize that all rows and columns of your data frame are numbered, and that individual cells in the data frame can be “called” by referring to their x and y coordinates using square brackets, [,]. Omitting a coordinate means you want to look at all of the records. The number preceding the comma always refers to rows, and the number after the comma always refers to columns. So if you want to examine the AREA for Nursery Field, you need to translate this into coordinates. Nursery field is in the third row, and AREA is the second column. (You can check this by viewing the whole object WORMS again.)

WORMS[3,2] If you want to examine the whole vector for SLOPE, you could either name the vector WORMS$SLOPE

Or you can use square brackets and omit the row information

WORMS[,3] How would you request rows 5-15 of the data frame (hint – use “:” to specify a range of rows or columns)? How would you store this information for later use?

You can also include operators within square brackets to specify conditions that entries must meet to be included. Which rows are selected using the following code?

WORMS[WORMS$AREA>3,]

WORMS[WORMS$AREA>3 & WORMS$SLOPE<3,]

You can also sort the data according to one of the vectors, using the command order. The following code orders columns 1:6 by AREA (column 2). Note that I have ‘nested’ a function, order, within the call. R is smart enough to deal with this.

WORMS[order(WORMS[,2]),1:6]

You can nest more than one function at a time. The command rev() reverses the order of a vector. Try creating a new appropriately named object that contains rows sorted in decreasing order of SOIL.PH.

We have already used one kind of operator, >, to specify conditions that must be met. R has a long list of these that may be useful (<, <=, >, >=, == for exact equality and != for inequality). Note that the ‘identity’ operator is two equals signs together, ==, and that ‘unequal to’ is an exclamation point followed by an equals sign, !=. A single equal sign is usually equivalent to the assignment operator <- but because that can get very confusing I like to stick to the <- and avoid using = in my own coding.

Store the following subsets in well-named objects. Note that you don’t necessarily need to do any of these in a single step! Store objects that break down the task if it helps.

All records where DAMP is TRUE

All columns with Meadow VEGETATION

All columns in Grasslands, sorted by DENSITY

The plots where density < 5.5, sorted by SLOPE

The AREA and SOIL.PH for all plots not in Grassland

**Graphing**

One of R’s greatest strengths lies in its ability to create precise, publication-quality graphs quickly and easily. The basic command is plot(). You need to tell R what to put along the X (horizontal) and Y (vertical) axes. R will make a good guess about the rest. However, you can override any of R’s default settings. For help on the many possibilities using plot, use ?plot.default

plot(x = WORMS$SOIL.PH, y = WORMS$WORM.DENSITY)

The names of arguments are optional, as long as they are supplied in the correct order, so this creates the identical graph:

plot(WORMS$ SOIL.PH, WORMS$WORM.DENSITY)

Graphics can be modified in many ways. For example, to make all the points blue, use

plot(WORMS$SOIL.PH, WORMS$WORM.DENSITY, col = 'blue')

To add custom x and y- axis labels, use

plot(WORMS$SOIL.PH, WORMS$WORM.DENSITY, col = 'blue', xlab = 'Soil pH', ylab = "Worm density")

To make the points vary by the size of each site, use

plot(WORMS$SOIL.PH, WORMS$WORM.DENSITY, cex = WORMS$AREA)

**Getting help**

There are many sources for help on using R. We can only teach you a small fraction of what you will need, so you must be sufficiently resourceful and confident to find and use help from various sources. R includes its own help documentation. If you know the name of the function for which you want help, just execute a command with a question mark preceding the function:

?mean

The format of R’s help documentation takes some getting used to. It always includes a description of the object, a template for how to use the object, a description of the optional arguments, more details on usage, authorship information, and finally some example bits of code (which can be copied directly into your console if you want to see the function illustrated). The last section (Examples) is often the most useful.

Often you may not know the name of a function, so using two consecutive question marks allows you to search help files for a term. Suppose you didn’t know that the function for standard deviation was sd, you might search for standard deviation (you can’t include spaces in your search, so you must replace the space with a fullstop).

??standard.deviation

The result of your search (if there are any) will be a list of help pages containing the phrase specified. The name of each function begins with the package containing the function (e.g., the sd() function is from the ‘stats’ package. We’ll cover packages in more detail later in the module.

Can you find out what the functions are for calculating the median, mode, maximum and minimum of a vector? What about the square root, or the natural log, or log base 10? Try these functions out on some of the vectors in your data frame. You may need to be resourceful in your search....

In addition to the help files, R functions are often but not always illustrated with examples (for short functions) and demonstrations (for more involved procedures). Use the example() or demo() functions to run these.

example(sd)

demo(lm.glm)

There are of course many resources for help outside the R environment that you may prefer to rely on. Your textbooks, instructors, and classmates may all be able to help, but you should also consult online resources including CRAN (the comprehensive R archive network), from which you probably downloaded R, and which has many manuals and links to other resources. Google may also help with some queries, as its ability to search specific error messages can often rescue an otherwise insoluble situation. Be persistent and resourceful, and patient in deciphering what seems like (and is!) a rather foreign language at the start of your learning curve.

Once you are satisfied with your ability to search for help, make sure you have saved your R script. If you want, you can also save the workspace, which means that next time R will open this project with all objects intact (rather than needing you to run the script to generate the objects again).