Quantitative Wildlife Ecology: Lab Exercise 1

A brief introduction to R

The objective of this lab is to get yourselves familiarized with the software program R. We will be using R extensively for the remainder of this course, and there is a good possibility that you will use it frequently in the future. Why use R? For one, R is widely used in the academic community. It is popular for its ability to perform advanced statistical analyses and produce publication quality figures. You need to code to use R; this makes it a bit more difficult to use than some other statistical programs. But on the other hand, it also makes R more flexible and able to perform highly advanced and user-defined analyses. Importantly, R is open-source (i.e., free), and works with Windows, Macs and Linux systems. You can download R from http://cran.r-project.org/.

A brief history of R

The R language is based on the S programming language, which was developed in the early 1980's. S (which later became S-plus) is a commercial product, also popular among statisticians. Ross Ihaka and Robert Gentleman developed a reduced version of S for teaching purposes and they chose the preceding letter to name it…R. In 1995, following the uprising of the open source movement, R was released and its source code was opened to the community who played an important role modifying and incorporating new features to the program. That is the way R is maintained today. Even though it has 20 core members (mostly computer scientists and statisticians) that maintain the main features of the software, users like you and me are responsible for maintaining and developing the libraries that perform a wide variety of mathematical and statistical analysis.

Some hints for future use

ALWAYS save your code. To facilitate this, we encourage the use of RStudio (download from (http://www.rstudio.com/). You must download the most current version of R before running RStudio. In RStudio, open a new script file and save it in a convenient location on your computer **BEFORE** starting on your assignment. When submitting your assignment, be sure to include your code.

ANNOTATE your code. This means that at each stage you need to provide sufficient commentary on your code such that it is self-explanatory to both yourself (when you view your code at a later stage) and others. We will discuss how to annotate your code shortly.

There are excellent resources for R available online. Apart from asking your TAs, also explore the R help files, and the resources available online (simply Google errors

you obtain or commands you want to use and you will be surprised at how much information there is online). You can also obtain help on any command in R by including a '?' symbol before the specified word (such as ?plot).

Note that R is **CASE SENSITIVE**. Make sure you check the names of variables, column names etc. that you use: often errors in our code pertain to small mistakes we make in these names.

Calculations in R follow the order of operations. So make sure you use parentheses appropriately and frequently.

Today, we will:

(a) familiarize ourselves with the RStudio interface
(b) explore objects in R. R is an object oriented programming language. This basically refers to a programming style based on "objects": in R, these objects include variables, vectors, matrices, data frames and lists.
(c) use some built-in functions
(d) read and work with some data on amphibians in Puerto Rico.

1. **RStudio Interface**: Open RStudio. The first thing you should do, as mentioned earlier is to open a new script file, and save it in a convenient location. Notice that there are now four windows within RStudio. One (top left) is your script file. The one below (bottom left) is the actual R window: this is where the code you write gets 'run', or implemented. On the right, we have one window (top right) to show us the objects we have saved while coding. The bottom right window is for help files, packages (you will learn more about these later), and plots.

Let's start with the very basic. R can be used as a calculator, for example:

```
> 2+2
> log(16)
> exp(2.77)
```

Note: we will henceforth indicate all code with the '>' symbol as you see above. You will notice that this is the prompt in your R window (bottom left).(You can run code by pressing the "run" button in the upper right corner of the script window, by pressing Ctrl+r, or Ctrl+Enter).

2. **Annotating code**: Something **SUPER** important is to annotate your code for easy reference. So we will talk about this first. This is so that when you view your code later (perhaps just before your finals), you can remember what your code was intended to do, what it does, and why you have included it in your work. Include comments in your code by prefixing it with a "#" symbol. You will notice that this part of the code now

turns green in your script window.

```
# As an example of inserting comments into code
# I've put this block of text, commenting is easier to NOT do
# than to do
# but invaluable when you go back to re-use code you made
later
# on
```

3. **Variables**: Well, that was easy. Now let us start working with the objects we spoke of earlier. The simplest of these are variables. You can store information by assigning it to a variable. For instance, if you want to assign the value '4' to a variable, say 'x', we can do this by typing:

```
> x<-4
```

By using the <- symbol you told R that whatever is in the right hand side of the symbol, is going to be stored in the variable x. This symbol is equivalent to the '=' symbol and the two can be used interchangeably.

To view the variable x, type:

```
> x
```

Also notice that the variable has now appeared in your 'objects' window (top right).

You can perform calculations with stored variables. For example,

```
> x^2
> x+2
> x+x+x
```

Also try:

```
> a = 5
> x + a
```

4. **Vectors**: You may want to include more than one value within an object. For example you may want to save the numbers 1 to 6 within a single object. For this, we will use vectors.

```
# To collate multiple values, we will use "c(…)" as shown
# below.
> x.vec<-c(1,2,3,4,5,6)
> x.vec
```

The same result as above can be obtained by using:

```
> x.vec<- 1:6
```

Calculations can be made using these vectors.

```
> x.vec*10
> x.vec*10-x
# Note the importance of parenthesis and the order of
operations.
> x.vec*(10-x)
```

5. **Matrices**: An important data structure in ecology, widely used in population and landscape ecology, is a matrix. Matrices are two-dimensional arrays of numbers consisting of rows and columns. As an example of matrix usage in population ecology, a population projection matrix is one made of age-specific mortality and reproductive rates. This matrix is then used to provide various population parameters and for population projections. As another example, a matrix can be used as a spatially explicit representation of a landscape.

There are alternative ways to form a matrix. For one, we can form an empty matrix and then populate it.

```
# First, create an empty 3x3 matrix.
> mat1<-matrix(nrow=3,ncol=3)
> mat1
```

Note that the arguments `nrow` and `ncol` are used to represent the number of rows and columns in the matrix, respectively. Keep in mind: rows before columns.

```
# Second, populate this matrix.
# For e.g., to make the entry in the 2nd row and 1st column 35,
> mat1[2,1]=35 # here, we say mat[row number, column number]
> mat1
```

Now try changing the entry in the 1st row and 2nd column to 53,

```
> mat1[1,2]=53
> mat1
```

If you only want to view the first row of the matrix, you can leave the column entry blank:

```
> mat1[1,]
```

We can also form a matrix based on certain values we define, or have already saved.

```
# One example,
> mat2 = matrix(1:6, nrow=2, ncol=3)
```

```
# Or…
> x = c(0,1,3,5,19,8)
> mat3 = matrix(x, nrow=3, ncol=2)
```

View each of these matrices to see what they contain.

6. **Data-frames**: Data-frames are the most common data structure in R. They also consist of multiple rows and columns, but each column can contain numbers, strings (text), or even factors. The only constraint is that all the columns need to be of the same length.

Let us now create a data-frame to mimic some data that we may work with in the future. Let us assume that we have measured the body mass of 6 squirrels. To create the data-frame, let us first make vectors that correspond to body mass and sex of squirrels.

```
> id = 1:6
> sex=c("m","f","f","m","f","f")
> bodymass=c(45,58,60,47,31,57)
# Now combining these vectors to create a data frame.
> squirrels=data.frame(ID = id,
                       Sex = sex,
                       Bodymass = bodymass)
> squirrels
# We now have a data frame with three columns.
# Note that we have specified the names of the columns above
# as ID, Sex and Bodymass.
```

One of the most important characteristics of data frames is their flexibility to subset. For example, if I want the data for body mass, we use the `$` symbol to make the subset.

```
> squirrels$Bodymass
```

If you want a subset of the body mass data for only females,

```
> squirrels$Bodymass[squirrels$Sex=="f"]
```

Note that the quotes (“”) are important because it is the way R knows that you are referring to the value (text) stored within a variable and not the variable itself.
Notice also that we are using “==” or double equal-to symbols. We use these within a conditional context, i.e., above we are asking for these values that fulfill the condition that the column Sex is ‘f’.

You can also use more than one criterion to subset data frames. For example, if you want the body mass data only for those males that are bigger than 35 grams:

```
> squirrels$Bodymass[squirrels$Sex=="m" & squirrels$Bodymass >
35]
```

Note that we are using the "`&`" symbol to represent "and". Thus, we are looking at the column 'Bodymass' of the data frame 'squirrels' where the entry in column 'Sex' is equal to 'm' AND the corresponding entry in column 'Bodymass' is greater than 35. If, on the other hand, you want a subset of Bodymass for females OR squirrels greater than 45 grams then you would use the "`|`"symbol (located one row above the return/enter button on your keyboard) instead of &.

```
> squirrels$Bodymass[squirrels$Sex=="f" | squirrels$Bodymass > 45]
```

Another object type in R is a list, which is even more flexible that data.frames. Look these up in R help files for more information.

We now know how to create and use 4 basic objects in R: variables, vectors, matrices and data-frames. Let us now look at some other procedures we can operate on these objects through built-in functions in R.

7. **Built-in functions**: R has several functions, or built-in procedures that have already been defined to make calculations. For example, the function `sum()` will calculate the summation of the values or variables included inside the parentheses.

Remember we set x.vec as the numbers 1 through 6. Type `sum(x.vec)` to see what you obtain.

Each function within R uses what are termed 'arguments'. An argument is just a necessary or optional input for the specified function. The argument for the function 'sum' is the set of values that are to be summed.

Let us calculate the mean of x.vec. Recall the formula for calculating the mean of multiple values:

$$\bar{x} = \frac{\sum x_i}{n}$$

where x_i is each value inside the vector x and n is the total number of items in x.

There are two ways to do this:

```
> sum(x.vec)
> length(x.vec)
> sum(x.vec)/length(x.vec)
> x.mean<-sum(x.vec)/length(x.vec)
```

Note that we are also using the function `length()` which counts the number of items

in a vector.

You can also use the already built-in function `mean()`.

```
> mean(x.vec)
```

Calculate the standard deviation of x.vec using the following equation and compare your result using the function `sd()`. (Hint: use `sqrt()` to calculate the square root).

$$SD = \sqrt{\frac{\sum((x_i - \bar{x})^2)}{n-1}}$$

```
> sqrt(sum((x.vec-x.mean)^2)/(length(x.vec)-1))
# Note that you can use the x.mean you created
# or use mean(x.vec) instead.
# Remember to put the parentheses in the right places.
```

So far we have seen functions that require only one argument (inputs for the function, as mentioned above). Let us move on to functions that require more than one argument. For example, to create a sequence of numbers starting from 0 to 1000, but including only every 25th number, we can use the function `seq(from,to,by)`

```
> seq(from = 0, to = 1000, by =25)
```

Use `?seq` to obtain help on the function; you will also see some examples in the help file.

Note: It is not necessary to include the name of each argument (above these names are 'from', 'to' and 'by'); but this is a good practice so that you do not have to remember the order of arguments that the function requires. To better understand the usefulness of this try the following four options and see how they differ:

a. seq(0,50,5)
b. seq(from=0, to=50, by=5)
c. seq(0,5,50)
d. seq(from=0, by=5, to=50)

You may want to create a vector with only one value repeated multiple times. For example, you need a vector of size 20 with only ones. You can use `rep(x, times)`

```
> rep(1,20) # meaning you repeat the number '1', 20 times
```

If you already have a sequence of interest, for example,
`s.seq<-c(0,26,37)` you can use the rep function to repeat it as you need. Let's say I need one 0, thirty repeats of 26 and ten repeats of 37,

```
> rep(s.seq,c(1,30,10))
```

Some more useful functions:

To get more information on other useful functions, explore the resources available within the R help pages and online. Look for operations you would want to perform on data frames or vectors and search for these. For starters, you could find out what the function is to find the maximum value of a set of values, to find out the number of values stored within a vector (its count), etc. For practice, calculate standard descriptive statistics including mean, minimum, maximum, standard deviation, variance and number of animals weighed on the body mass of squirrels.

So did you find the function to obtain a maximum value from a set of values? We will use this to obtain data on the squirrel with the largest body mass,

```
> max(squirrels$Bodymass) # gives the maximum body mass
> squirrels[squirrels$Bodymass == max(squirrels$Bodymass),]
```

Remember to include the comma mark in the end….do you recall why we need this? What we are saying is that we want to identify and subset the row where the body mass is the highest but, we want to look at all columns for that row.

Now calculate the mean and standard deviation of females and males body mass separately.

```
> mean(squirrels$Bodymass[squirrels$Sex=="m"])
> mean(squirrels$Bodymass[squirrels$Sex=="f"])
> sd(squirrels$Bodymass[squirrels$Sex=="m"])
> sd(squirrels$Bodymass[squirrels$Sex=="f"])
```

One useful function for data management `is.na()`, which can be used to turn all 'NA' values in your data frame or matrix into whatever you need. In this case we are going to use it to turn NA's into 0.

```
> mat1
> mat1[is.na(mat1)]=0
> mat1
```

Remember from where we introduced data frames that the condition included within the square brackets '`[]`' subsets the matrix.

Working with data

Now let us get to the exciting part: learning how to input and work with data. R can read data in many formats. The two most common are tab-delimited text and comma separated (i.e., .txt and .csv) files.

Now that we are working with files and will soon (in the next lab) create output files or plots, we need to set our working directory. From the next lab onwards, remember to include this step at the very beginning of your script. The easiest way to do this for the first time is to go to (at the top of R Studio) Session → Set Working Directory option and then choose folder. Navigate to the folder will you will be working on this project (such as in your jump drive where you saved the data). When you do this, you will see a command appear in your R window (bottom left). Copy and past this in your script file back into your R studio window on the top to make it easy to save for future use. It should look something like this:

```
> setwd('H:\\Q Eco 2013\\Labs 2013\\Lab 2 R Intro')
```

The command in R is `setwd`, and the argument inside is the address of the location (folder) where you will store all required files. There is also a trick on the video tutorial so you can see how to find this folder name.

We will now work with some simulated data on snout-vent length of two species of amphibians living in the mountains of Puerto Rico: *Eleutherodactylus coqui* (ELCO) and *Eleutherodactylus portoricensis* (ELPO). The data also includes the elevation at which these individuals were measured. Open up the file 'Amphibians.xlsx' in Excel and take a look at the data. Now save the file as a tab delimited file (.txt), by using the "Save As" option (locate "Text (tab delimited)" in the drop down box for "Save as type").

Now within R, type:

```
> amphibians=read.table("Amphibians.txt",header=TRUE)
# Here, we are assigning the contents of the text file we have
# just saved to amphibians (a data frame),
# and we are specifying
# that there are headers to the data – these headers will be
# included as column names.
```

A nice trick to visualize a data frame is to use the function `head()`, which prints only the first six rows of the data frame.

```
> head(amphibians)
```

Lab Assignment 1

(Remember to turn in your R code as well as complete the tables and assignment below by filling in the answers on the Lab 1 quiz on CANVAS)

Using the amphibian data you have just inputting into R, obtain the following information on the two species of frogs.

1. Calculate the count of snout-to-vent length (SVL), mean SVL, and associated standard

deviations for each species (ELPO, and ELCO) (6 points)

Species	**Count**	**Mean**	**Standard deviation**
ELPO			
ELCO			

2. Now calculate count of SVL, the mean of SVL, and standard deviation for both species, but only for those individuals captured at locations where elevation is greater than 500 m. (6 points)

Species	**Count**	**Mean**	**Standard deviation**
ELPO			
ELCO			

3. You just found out that the data on elevation is incorrect as the survey instrument was not properly calibrated, and that all elevation measures should be 1.14 times the value recorded. Fix this error and recalculate the mean SVL and standard deviation for individuals captured above 500 m elevation (for each species). (*Hint*: create a new variable with the corrected elevation: 1.14 x old elevation). (4 points)

Species	**Mean**	**Standard deviation**
ELPO		
ELCO		

4. Upload your R code <u>with</u> annotations explaining what each part of the code does (2 points).