Problem Set 1

Deterministic Growth

Part A. (Re)Learning R

A goal for this course is to introduce you to the methods of simulating (and later analysing) ecological models. I'm going to assume you're starting from step 0. These days computers play a key role in both of these methods, so your first assignment will be to start learning how to code in R. A number of you may already be familiar with R in the context of statistical analysis, but what we'll eventually be doing may be a little different from what you're used to, so may require some new ways of thinking about R. But let's first spend some time (re)familiarizing ourselves with the environment. That is, let's start by interacting with R using the "R Console".

Open R and try typing `x = 2` before you hit enter. Nothing will appear to be happening, but if you now type `x` you will see `[1] 2`, which is R's way of saying that the value of `x` is 2. Now, `x` has only one value so you can see that the program is being a little compulsive in telling you that you're seeing the first (i.e. the `[1]` entry), but the basic idea is that you have successfully assigned the value 2 to the variable `x`. Now try typing `t = c(1:20)`, hit enter, and then type `t` and hit enter, as before. This time you will see that you have assigned the numbers 1 through 20 to the vector `t`. In fact, that's what the function `c` does, it combines (concatenates) values into a vector. To find out more about the function `c`, type `?c` (i.e. a question-mark following by the letter *c*), and hit enter again. A new window will come up that describes `c`. The explanation is a bit stilted, but that's because the authors are trying very hard to be concise and consistent. The main point is that you can ask R for help with any function using the question mark. For many functions you can see their actual code by typing the name of the function all by itself. (Try typing `apply` into the command prompt, for example.) Often you will see functions written in text as `c()` where the empty parentheses are used to indicate that input variables (i.e. parameters) are required.

Notice that it is possible to assign a variable to a letter already used to denote a function, as in `c = 3`. This can create a lot of confusion so is best avoided. Also notice that while using the `=` command works, it's actually more typical to use a left-arrow `<-` (a less-than symbol followed by a minus symbol) to assign variables. That's because the equal symbol also comes into play in logic statements (e.g., try typing: `a <- 2; b <- 2; a == b;`), as well as in other more nuanced functions down the road. It's therefore best to use the left-arrow symbol in all but the simplest of commands. [As an aside: `2 -> a` works as well!]

Resources & R Studio

To start becoming familiar with the resources available to you in trying to fix coding problems (which I guarantee will be a never-ending part of your coding experience), I want you to start using the resources available to you. Besides Chapter 2 of *The R Book* (posted on our course website), there are many online resources (e.g., the "R manual") at the R Project website. (On the R Project website, to get to the manual, click on Manuals on the left and then choose *An Introduction to R*. The reference cards can also be useful and are posted on our course website.)

If you have never interacted with R before, using it might be somewhat of a frustrating challenge at first. Don't worry though, the learning curve is not as steep as it seems. Learning R is like

learning any other language (seriously), but with more resources for help. The more you practice it and the more you're forced to try implement new and different intentions, the easier it will become. Think of this class as the equivalent of being dropped off in a foreign country. The rest after that is simply learning additional tricks and shortcuts, and becoming increasingly familiar with all the tools and packages (bundled sets of functions) that other people have written.

There are a number of ways to code in `R`. As you saw above, you can easily type commands directly into the command-line. For longer scripts this gets tedious and inefficient. In fact, the beauty of scripts is that you can save pages and pages of code and run it without having to type it in line by line the next time you want to run the same set of commands. There are several ways of writing scripts and interacting with `R` more generally, but these days the easiest is the `R Studio` interface to `R`. I'm pretty sure most use `R Studio` these days. (Many don't even realize that `R` and `R Studio` are not the same thing!) So I'll assume you're using `R Studio` from here on out.

Part B: Simulating deterministic geometric growth

What we want you to do next is to make a plot of a population undergoing geometric growth according to the model

$$N(t+1) = \lambda N(t)\,. \tag{1}$$

Here $N(t)$ is the population size[1] at time t, and λ is the net number of offspring per individual per time-step. In class we will solve this equation to show that $N(t)$ can be reformulated as a function of any given starting size $N(0)$. However, for the purposes of learning the basics of `R`, I instead want you to solve for each value of $N(t)$ in an iterative fashion using eqn. 1.

Open a new `R` script within `R Studio` and save it to a designated folder on your hard-drive. Let's call it `Test.R` and save it to a location like
`/Users/marknovak/TheoreticalEcology/ProblemSets/Test.R`.
Once you've saved your script, there are several options for running it, but the easiest and most specific are:

- Method 1: Simply copy and paste your code from the script editing window into the command prompt. (Hit the `Enter` key if nothing seems to happen.)
- Method 2: Highlight (select) the lines of code you'd like to run and hit `Ctrl+Enter`. (You can change this short-key under the `Tools` menu.)

To give you a head-start on writing your geometric growth code, here are the first four lines of the script that you need to write:

```
lambda <- 2           # this line sets the variable "lambda" equal to 2
TimeSteps <- 10       # sets the variable "TimeSteps" equal to 10
t <- c(1:TimeSteps)   # creates a vector from 1 to the number of timesteps
N <- numeric()        # creates an empty numeric vector
```

Hopefully the meaning of the first 3 lines in this code are fairly obvious by now. Note that you could also use `t <- seq(1,TimeSteps,1)` or even just `t <- 1:TimeSteps` for the 3rd line. In the 4th line, `N = numeric()` means that you are telling `R` to make a new empty vector named `N` and that its contents are going to be numbers (1,2,3, ...) rather than characters ("a", "b", "c", ...). Creating empty vectors (or, later on, 2-dimensional matrices and n-dimensional arrays) allows you

[1] equivalently written as N_t

to store information more quickly and easily by filling the spaces in as you go, rather than having to concatenate old and new results. Notice also that I've used the pound symbol `#` (aka hashtag) to annotate my code. `R` will ignore any text that comes between the `#` and the end of the line. Believe me, it is very useful to annotate your code as fully as you can. You'll be amazed at how often seemingly obvious code is totally impenetrable to you a year (or even a week) later.

Part B.2

Finally, your assignment... Extend your code to iterate eqn. 1 and to plot $log_e(N(t))$ versus time for `t` going from 1 to the total number of time steps. Set your starting population size to be 1 individual and $\lambda = 2.0$.

Hint #1 is that you need to use a `for` loop, which you can read about in your assigned readings in *The R Book* or in the `R` manual referred to above.
Hint #2 is that, to put a value into `N`, you can write, for example, `N[1] = 1`. (Yes, you will indeed need to include that line into your code.)
Hint #3 is that while `TimeSteps = 10` the resulting `N` vector will be of a different length. Why? You'll need to think through this to figure out your plotting.

To figure out how to plot $log_e(N)$ versus time, ask `R` about `plot()` by typing `?plot` at the command prompt. (A second way to obtain logarithmic axes is described in the `par` command which allows you to pre-specify how your plot will look before you use `plot()`.)

Now repeat the above and use either the function `points()` or the function `lines()` to add the results of a second simulation where $\lambda = 1.8$ to your existing plot.

Part B.2

How does the plot of $log_e(N)$ vs. time compare to the plot of N vs. time? What does plotting log_e population size as a function of time indicate about the population's growth rate? Confirm this by plotting the relationship between the population's growth rate (i.e. the *relative* change in N between successive time steps) and N or $log_e(N)$.
Hint #4: You may need to use the function `length()` and a shortcut command described on the R Reference Card under the heading *Slicing and extracting data* to do this calculation in one line of code.
Use the function `abline(h=lambda, lty=2, col='red')` to check your answer.

Part B.3

Suppose a different population had a starting population size of 1000 individuals but had $\lambda = 0.8$.
(a) How long will it take the population to go extinct (reach a population size of zero)?
(b) How many time-steps does it take the population to reach quasi-extinction if we define quasi-extinction as attaining 1% of the population's starting population size?
Hint #5: Since the time-steps we're using for our simulations are in units of 1, you can use a sub-setting method to do this. For a vector `x <- rev(seq(0.1, 100, 0.1))` you can could use `qe <- length( x[ x > x[1]/100 ] )`.
Backup your answer with a plot using `abline()` to indicate the time-step of quasi-extinction.

If you're already an `R` guru and got done with the assignment with time to spare...

- **Help out a fellow classmate.**
- **Think about how to perform the simulation a completely different way. Can you do it by writing a function and using one of the `apply` family of functions?**
- **See if you can do it all in Mathematica too! (A solution will be posted.)**