An Introductory Tutorial:
Learning R for Quantitative Thinking in the Life Sciences

Scott C Merrill

September 19th, 2012

Chapter 4

Clearing R's memory.
I frequently am running pretty big chunks of code and forget which variables have been assigned values, or am re-running a portion of the code. Without starting with a blank slate you can run into problems because R will remember. This is especially important if you are running through loops and using counts in your code. I always start new scripts (or new sections of scripts that are unrelated to previous material) with the following statement, which clears R's memory.

```
rm(list = ls(all = TRUE))
```

Assigning values and the equal sign
There are a few ways to assign values to objects in R. We have been extensively using the equal sign but this can lead to confusion. Many users prefer the combination symbol "<-":

```
Tent <- matrix(nrow = 4, ncol =8, data = 0)
```

Additionally you could use the assign function

```
assign (Tent,matrix(nrow =4,ncol=8, data = 0))
```

Either of these ways of assigning can help prevent confusion with the logical equals sign. That is,

```
X=10 # assigns X to be equal to the value 10
X==10 # asks R if X is equal to 10 (and would return a true or false depending upon the logical answer)
```

TRUE and FALSE in R
In R, TRUE and FALSE values have actual values. That is, one could sum a row of logical values. Using Hobb's A matrix, compile:

```
> sum(A[,1]> 1997)
```

I have found this to be of spectacular value when I was going through a set of 7.5 million cells looking to determine the calendar date where degree days past a insect developmental threshold.

More with For loops from Hobbs

6.1.2 For loops controlled by sequence vectors

The behavior of for loops is determined by a sequence. Above, I made things simple by using 1:10 to describe a sequence of 10 integers starting at 1 and ending at 10. However, for loops can be controlled by vectors that define a sequence. This requires that we understand the seq() function. Enter the following code in the console or from a script window:

```
s1 = seq(from = 0, to = .10, by = .02)
s1

#A shorthand version of the same statement is

s1 = seq(0,.10,.02)
```

This illustrates how you can form sequences of numbers over any range (with endpoints defined with from and to) with any increment (defined with by). The second version is shorthand for the first—if you can remember the order, then you can dispense with the from =, to =, and by = . If you use these, it is easy to forget the =, which will produce an error. Sequences can be useful for many things, but their primary use in this course will be to control for loops. So, the statement:

```
for (j in s1){
print (j)
}
```

will iterate over the values contained in s1, giving values in sequence to a variable named j, which can be visualized on your console because of the print(j) statement. The print(j) statement asks R to print out the value or object in the ().

6.1.3 Nested loops

Sometimes we want to iterate over two indices rather than one. This can be accomplished using nested for loops. For example, write this script in the program window and compile it:

```
a = matrix(0,nrow=5, ncol=5)
a

for (i in 1:5){
        for(j in 1:5){
                a[j,i] = j*i
        } #end of j for
} #end of i for
a
```

Nested loops work this way. The outer loop goes slowly, the inner loop, quickly. So in the above example, in the first trip through the loop i = 1 then j = 1,2,3,4,5. Then i = 2 and j = 1,2,3,4,5 and so on. In this example the results of a calculation are stored in the array a where rows are indexed by j and columns are indexed by i.

Exercise: a model experiment using nested for loops

We will often want to look at how changing a parameter value alters the trajectory of a model's predictions over time. This means that we want to iterate over different parameter values for a model, and for each value, store the vector of values of the state variable over time. This exercise will be your first experience doing that. Building on the discrete time model of exponential population growth developed above, run an experiment where you vary the values of lambda from 1 to 1.6 in steps of 0.1. Display the results together as a family of curves on a single graph. You are going to need some hints to do this. Here they are. This will seem hard at first, but after you have done this once, it will be much easier the next time.

1. Make a vector (using the seq() function) of values of lambda from 1 to 1.6 in steps of 0.1.

2. Make a matrix with 10 rows and a number of columns equal to the length of your lambda vector. Initialize all of columns in row 1 of your array to hold the value 10. (Remember how you can do this in one, simple step using a comma to index your matrix). Call your matrix N.

3. Create nested loops. The outer one controls the value of lambda and the column where you want the results of the simulation for each value of t. The inner loops controls t. Within both loops is the calculation of the population size based on the value of lambda indexed by the outer loop and time indexed by the inner loop. Store the result in your 2 dimensional array. Each column should hold the values of N calculated at the different time steps.

4. At the end of your simulation, plot the results using matplot(N, type = 'b') where N is the name you gave your matrix and type = 'b' says you want results plotted with lines and symbols. More about this command later.

The If Then Else statement (This is from SCM – so no answers will be at the back of the book).

Occasionally, you may want to compile a statement or series of statements only if some condition exists. For example, you may only want to only allow your trees to produce viable offspring after a fire (e.g., reproduction linked to serotinous cones). You could ask, has a fire occurred in this location, if so then compile the reproduction probability code. Like the For loop, the if then statement starts with a clause and then continues after an {

Compile:

X = seq(1:10)

```
if (X==5) {
        print(X)
}
```

A looping if then exercise:

If a forest system, you are interested in pine forest health and stand density. Prescribed fires occur in plots 4, 5 and 6 on year 1 and 1, 2 and 3 on year 4. When a tree reaches three years of age it has increased resistance to fire. Mature trees survive fires, saplings do not. Plots with mature trees receive one sprout per mature tree the year after a fire. Given the following start to the system, forecast the number of trees and sprouts living in year 10. Second year saplings become mature in year 3.

```
# Setting up the initial conditions for the array
Idaho =array(0, dim= c(6,5,10))

# Initial tree density for each class
Idaho[,1,1] = c(3,1,4,0,2,1) # mature trees in plots in year 1
Idaho[,2,1]=c(1,0,1,0,0,1) # 1 year old saplings in plots in year 1
Idaho[,3,1]=c(1,0,0,1,0,1) # 2 year old saplings in plots in year 1
Idaho[,4,1]=c(0,0,0,0,0,0) # sprouts in plots in year 1

# sites that burned in year 1 receive a 1 in the fire column (column 5).
Idaho[,5,1]=c(0,0,0,1,1,1) # fire in plots in year 1

for (time in 2:10) {
        if ( fire == 1) {
        …
        }
        Idaho[,1,time]…
```

Plot the sum of each field plot's trees (all ages & yes, I should have used a different word than plot for given its double meaning in a single sentence) over time as lines all on a single graphic display.

Rerun this exercise but make fires after year 1 random with one site burning per year. Hint:

```
> sample(1:6,1)
```