Tutorial 5: R programming basics

Tom Miller

While R is perhaps best known as a statistical tool for analyzing data or for making graphs, it is also really useful as a simple programming language and compiler. Using R for programming is not "machine efficient" and the programs can be a tad clunky and slow. However, for most users, it is very friendly and easy programming environment. And, because the programs can then incorporate other R functions, they can often be incredibly useful for analyzing and visualizing data.

The first thing to realize is the R really is just a bunch of smaller programs that have already been written in to one package for you to use. As an example, we can enter a column of numbers by typing:

```
> bob=c(3,5,2,6,4,6)
```

We could then ask for the average number by typing:

```
> mean(bob)
```

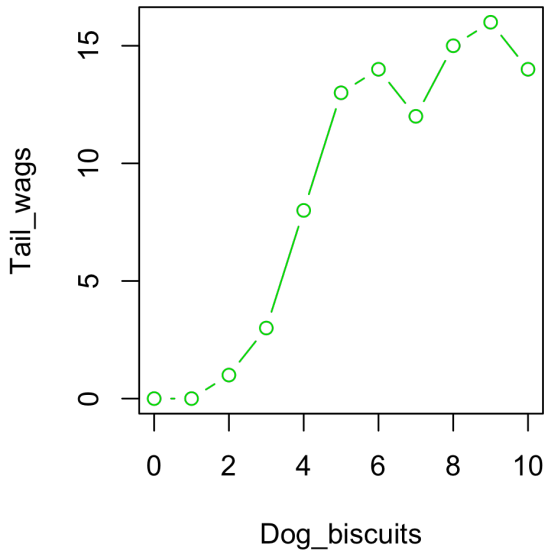
And the computer should return:

```
[1]  4.33333
```

In this case, R already has program written to determine averages. By typing "mean" you called up that program, which simply adds these numbers and divides by the sample size. Because this program can be called by a single name, it is referred to as a **function**.

R comes with a huge number of built-in functions, especially to do graphics and statistics . . . .or you can use it to write your own programs and functions, which can incorporate programs already built into R. Here is an example of other functions, that draw an XY plot for us.

```
>Dog_biscuits=c(0,1,2,3,4,5,6,7,8,9,10)
>Tail_wags = c(0,0,1,3,8,13,14,12,15,16,14)
>tw=mean(Tail_wags)
>plot(Tail_wags~Dog_biscuits,typ="b",col=3)
```



Using Text Editors and ".R" Files in R

Before we even start writing programs in R, you need to learn another R feature -- using a text editor window (if you haven't learned this already). By using a text editor, you can wrote whole groups of commands and have the computer run them separately or all together. Further, text editors allow you to save your program for later use (or just to admire and show your friends).

To understand text editors, you need to be aware that there are three different types of windows that are used by R: console, graphics, and text editor windows. You are already familiar with the first two. The window where you enter line commands is the **R Console**. When you used the "plot" command, it opened a new window, which is the graphics window. With a Mac, graphics windows are called "**Quartz**". On a PC, this window is, somewhat more logically, the "**graphics**" window.

Text editor windows are just simple text editors that are smart enough to interact with R.

-- To create such a window on Macs, go up to "File" and open "New Document". To execute (sic) any of your commands, either highlight the command(s) or put the cursor anywhere on that line. Then, press and hold "A" while also pressing the return key.
-- On a PC, go to "File" and open "New script". To execute commands, either highlight the command(s) or put the cursor anywhere on that line and push the button in upper corner of the main R window for "Run line or selection."

Creating a new document (or script) opens a simple text editor in R. You can then enter multiple lines of commands that are not executed until you are ready. And, instead of executing commands one by one, you can execute them all at once or any set of them together. You can also save the file (usually as a "____.R" file) and rerun these commands at a later time.

So, we could write the commands shown above to plot tail wags against biscuits as four separate lines in the R Console window OR, we could first write these commands in a text window (note the little prompts are gone from the front of each line):

```
Dog_biscuits=c(0,1,2,3,4,5,6,7,8,9,10)
Tail_wags = c(0,0,1,3,8,13,14,12,15,16,14)
tw=mean(Tail_wags)
plot(Tail_wags~Dog_biscuits,typ="b",col=3)
```

Then, have the computer run this four lines all at once. Further, we could then save this file as "TailWags.R" to use or add to at a later date.

From now on, we will assume that you are using a text editor, as it makes it much easier to write and edit programs, as well as to save programs for later use. Text editors also allow us to include tabs and blank lines, which often make programs easier to read and understand.

A Simple Program of Population Growth

Ecologists are interested in population sizes and so often make programs that simulate populations growing or interacting through time. A simple example can be used to show how "loops" are used – this is a basic tool of programming.

Assume that we start off with one bacterial cell and it doubles during a given time interval. We could do this in R by just keeping track of population size N:

```
N =1
N = 2*N
print (N)
```

Each previous N gets replaced by the new population size N. Now, just repeat this, with both bacterial cells dividing:

```
N = 2*N
print (N)
```

We could do this over and over, but it gets a little boring, so let's create a "for loop" to do this "for" us. **Loops** are very important for programming, so it is very important that you understand their structure and use.

```
N = 1
for (i in 1:10)  {
        N = 2*N
        print (N)
        }
```

You can see that the "for loop" simply repeats the actions in the curly brackets as many times as you wish: in this case, 10. Note the use of tabs to set the bit of code within brackets off to the right a bit. You should do this -- it makes your code much easier to read later.

Note that we have the "i" value, which actually increases from 1:10 each time we go through the loop. Let's put it to work to put our results in a column, with each row determined by "i":

```
N = rep(0,10)           #this creates an column or "vector" of 10 zeros.
N[1]=1                  #now we make the first number in the column equal to 1
for (i in 2:10)  {
        N[i] = 2*N[i-1]
        print (N[i])
        }
```

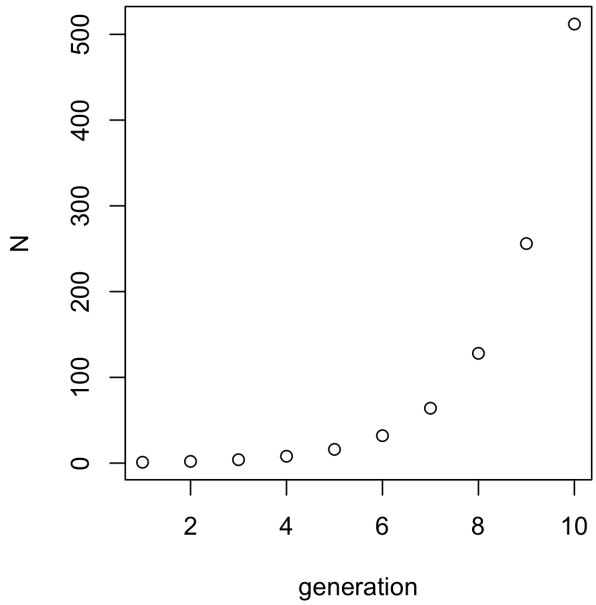
R knows to ignore everything typed on a line after it encounters a "#" symbol, which is convenient for adding comments to code. Don't be afraid (or too lazy) to do this -- again, it can really help to later figure out what you were trying to do.

All of our results are now in a column, which we can see by just typing "N"

```
[1]  1  2  4  8  16  32  64  128  256  512
```

Even better, we can now plot our results to see what is happening to our bacteria through time. But, we haven't really kept track of time per se, although it is equal to i each case in our loop. So, let's create another vector "generation" and set it equal to i each pass through the loop, then plot population size (N) against generation:

```
N = rep(0,10)
generation=rep(0,10)
N[1]=1
generation[1]=1
for (i in 2:10)  {
        N[i] = 2*N[i-1]
        generation[i]=i
        print (N[i])
        }
plot(N~generation)
```



If you can write this on your own, you will have written your first program. It simulates bacteria growing "exponentially" through time, doubling every generation. The bacteria are following the general equation:

$$N_{t+1} = N_t * 2$$

Now, we could begin to make this program a bit more flexible and, therefore, useful for other situations by adding some generic **variables.** First, we might want to run this for more than just 10 generations, so let's make the number of generations a variable in the program:

```
num_gen=10
N = rep(0,num_gen)
generation=rep(0,num_gen)
N[1]=1
```

```
for (i in 2:num_gen)  {
        N[i] = 2*N[i-1]
        generation[i]=i
        print (N[i])
        }
plot(N~generation)
```

We could also create a constant for the population growth rate, which is currently 2.0, to make our program a bit more flexible to use.

Making Functions from Your Program

Hey, remember back up at the beginning of this tutorial, we discussed the difference between a simple program and a function. In R, a program is just any group of commands that you wish to run as a set to achieve some output. A function is a program in R that can be run using a single command, such as "mean" or "plot". Such functions might be convenient for you to make if, for example, you want to run the same program over and over, perhaps just changing one or two variables.

We can do that with our program by using the "function" program. For example, might want to understand the effect of the growth rate on shape of the growth curve. We can make a new function called "grow":

```
grow = function(growth.rate){ #"growth.rate" will be a parameter for the function grow
        num_gen=10
        N = rep(0,num_gen)
        generation=rep(0,num_gen)
        N[1]=1
        for (i in 2:num_gen)  {
                N[i] = growth.rate*N[i-1]   #we replace "2" with "growth.rate"
                generation[i]=i
                }
        plot(N~generation, typ="l")
        }
```

The above program creates a new function called grow. Running the program doesn't produce any output. However, now run each of these lines, individually:

```
grow(1.5)

grow(2)

grow(8)
```

How does your graph change? Now, explain:

```
grow(1)
```

Exercises

1. Make sure you can write the example program above on your own. Using the text editor, create this program that simulates bacteria growing "exponentially" through time, doubling every generation. The bacteria are following the general equation:

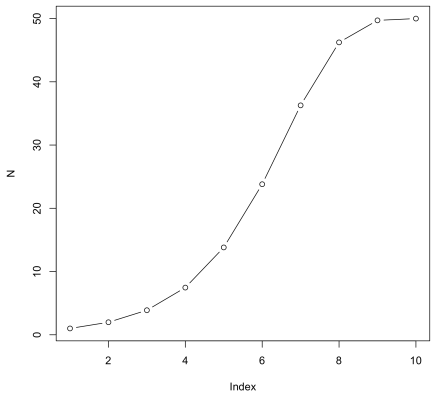
$$N_{t+1} = N_t * 2$$

   Make the program generalizable to use for different periods of time and growth rates. Write and save this as "exponential.R" so that you can run it again in the future.

2. Write a second program for a different kind of growth, the logistic growth pattern. Here,

$$N_{t+1} = N_t + \left[growth.rate * N_t * \left(\frac{100 - N_t}{100} \right) \right]$$

   Write and save this as "logistic.R" so that you can run it again in the future. Now make the program generalizable for different time periods, growth rates, and "carrying capacities" (the size at which the population levels off, which in the above equation is 100). You should get something like:



3. Let's try something a bit more complicated that will test both your programming and graphics skills. If we have two species that use the same resource (but with different rates and efficiencies), then we can model their population growth using two linked equations:

$$N1_{t+1} = N1_t + \left[growth.rate * N1_t * \left(\frac{K1 - N1_t - \alpha_{12} N2_t}{K1} \right) \right]$$

$$N2_{t+1} = N2_t + \left[growth.rate * N2_t * \left(\frac{K2 - N2_t - \alpha_{21} N1_t}{K2} \right) \right]$$

The K's are the carrying capacities, which can be different for different species (e.g., a grassland may support more rabbits than deer). The a's are "competition coefficients" that translate the effect of one species on another into units of the second species. Don't

worry about this for now, but try numbers in the range of 0.5 to 1.5, which would indicate species that are fairly similar competitors.

See if you can write a program that grows each species alone (no competition with the other species, then the two species together and then compares the results in graphs. This will probably require **split.screen** or **mfrow**, as well as **plot**, **lines**, and **text**. It would be easiest if you first made a function for a program of two species competing, with the input variables being initial densities. It should produce something like: