Functions and the for loop

T. Eiting, M. Rosario, C.-Y. Kuo

10.19.2012

1 Introduction

1.1 Welcome

This document provides the basics of understanding functions, and will give you enough information to start writing your own functions! Up until this point, we've made heavy of writing our R codes into scripts and then running them line by line. In practice, you might find that you often group 2 or more lines of code together because they build off of each other. For example, a line of code in your R script may perform some calculation, the next line of code may add your calculation to an existing data.frame, and the third line of code may produce a plot. If you are running many iterations of these 3 lines (for multiple species or treatment groups for example) it makes sense to group these lines together, and functions give us an excellent way of organizing and modifying our snipets of code.

1.2 What is a function

A function is basically comprised of 4 main parts, the name of the function (functionName), inputs to the function (input1,input2,input3), the code, and the output.

```
# creating a function
functionName <- function(input1, input2, input3) {
    # code goes here
    return(output)
}
```

The benefit or organizing our code this way is twofold. First of all, because our code is enclosed in the curly brackets, they "belong" to our function and are explicitly linked and organized in a way that groups the code together. This means that it is much more difficult for our code to get lost or undergo unexpected interations with other parts of the code.

```
# using a function we've created
functionName(a, b, c)
```

Second, as seen in the code above, using the function effectively the details of the function away from the user for ease of use (though the code within the function is available at anytime and can still be edited or replaced). In the above function, we've substituted variables a, b, and c for input1, input2, and input3 of our function. These ideas will become more aparrent as we build and use our own functions, so let's get started!

2 Building simple functions

2.1 Summing two numbers

For our first function, to get some practice in thinking in terms of functions, we'll create a function that takes any two numbers as inputs and then returns the sum of the two numbers. We'll call this function Sum2Num() and we'll call our inputs in1 and in2.

```
sum2num <- function(in1, in2) {
    output <- in1 + in2
    return(output)
}
```

First thing to note is that in creating this function we used a combination of parenthases and curly brackets. In this case, we have to keep in mind that parenthases () are used to enclose inputs, and curly brackets are used to enclose code. Another thing you may have noticed is that after typing in your function into R, nothing happened. This is because when you write a function, you don't actually run the function that you just created. To actually make use of your function, use the following syntax:

```
sum2num(1, 5)

## [1] 6

sum2num(-194, 45)

## [1] -149

sum2num(23 + 2 * 5, 5^2)

## [1] 58
```

2.2 Standard error function

Earlier in SOURCE, we discussed that there was no built in function for standard error. However, given what you now know about functions and the equation for standard error (standard deviation of valeues over square root of the number of values), you'll be able to write your own standard error function.

```
se <- function(x) {
    output <- sd(x)/sqrt(length(x))
    return(output)
}
```

3 The basics of the for loop

3.1 What is a for loop?

When we write functions and group lines of code together, we often would like to apply those lines of code to many different inputs. Whenever you find yourself applying the same code to different piece of data, you can use a "for loop" to iterate through the data! We can create a temporary variable (which we usually call "i" which stands for iteration), run a block of code with the variable of i, increment the value of i, and repeat until we have run through our code the desired amount of times. Again, this is easier to understand in practice, so let's make a loop.

3.2 Printing out a sequence of numbers from 1 to n

Though we can use more simple code to do this, this following function is a good example of the basics of a for loop.

```
showMeTheSequence <- function(n) {

    for (i in 1:n) {
```

```
        print(i)
    }
}

showMeTheSequence(10)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

3.3 Generating and plotting a random walk

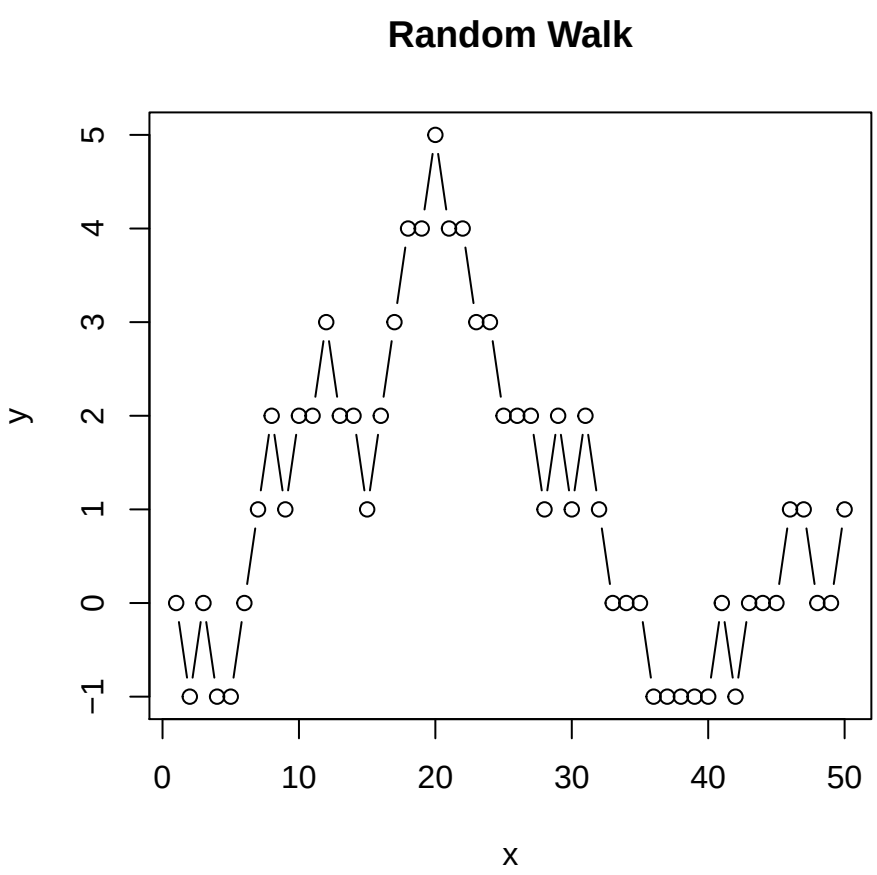
Functions and loops form the basis of many of the Monte Carlo solutions we'll be employing later in the semester to solve some really fun problems. The last missing piece is the ability to generate random numbers. We need the ability to generate random numbers if we want to create a function of a random walk model. Typically, our preferred method of generating random numbers is the runif() function, but in this case, we'll show you how to use the sample() function to create a script that not only generates the data for a random walk, but plots it as well.

```
ranWalker <- function(timeSteps) {
    # create x values for our random walk
    x <- 1:timeSteps
    # create a y vector to store our random walk values into
    y <- rep(0, timeSteps)

    # start the loop at 2 since the first time step is at 0
    for (i in 2:timeSteps) {
        y[i] <- y[i - 1] + sample(x = -1:1, size = 1)
    }

    plot(x = x, y = y, main = "Random Walk", type = "b")  #we specifiy type='b' to plot
both the points and the lines
}

ranWalker(50)
```


Random Walk
y
x
−1
0
1
2
3
4
5
0
10
20
30
40
50