Advanced looping

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

10.26.2012

"Loopers are well paid, they lead a good life..."- Joe from trailer of Looper

1 Introduction

1.1

One of the main advantages of learning how to handle your data in R or any other programming language is the ability to automate repetitive tasks. Last week, we learned how to organize our code into simple functions in order to reduce the number of tasks, however there is another way to approach the problem of data processing automation involving loops. Although we were introduced to loops in our lesson on functions, we just barely scratched the surface of using loops to solve problems. In this weeks lesson, well learn more about how we can use loops to extend our R functionality to perform some complex, but useful, tasks.

1.2 Basics of the for loop

The idea of a loop is actually very simple: we create a temporary variable (usually named i), and run all the code in the following curly brackets changing the value of i each time until we run out of values. Here is a simple example that uses a for loop to print out the numbers 1 through 10.

```
# creating a function
for (i in 1:10) {
    print(i)
}
```

By now, the example above should be familiar. We start with i=1, we print the number that i currently equals, and we continue until i=10. This example is useful for understanding how a loop works, but now let's spend some time in exploring how the for loop can be useful in handling and processing your data.

2 Advanced uses of the for loop

2.1 Calculating centers of mass of a moving object

Suppose we have 2 sets x,y data over time from a high speed video of an ant spinning through air. These datasets are available on the gitbub website and are named antHead.txt and antAbdomen.txt. Though this animal is spinning, what we would like to do is at each time step (row of the datasets) calculate the center of mass of the ant by taking the average of the x values and the average of the y values of the head and abdomen locations. We can do this by employing the for loop in the following way:

```
# read in data
antAb <- read.table("antAb.txt", header = T)
antHead <- read.table("antHead.txt", header = T)

# check data
str(antAb)
```

```
## 'data.frame': 74 obs. of  8 variables:
##  $ Frame: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Area : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Mean : int  134 123 122 143 143 145 155 161 166 161 ...
##  $ Min  : int  134 123 122 143 143 145 155 161 166 161 ...
##  $ Max  : int  134 123 122 143 143 145 155 161 166 161 ...
##  $ X    : num  9.48 9.31 9.15 9.19 9.37 ...
##  $ Y    : num  16.8 16.9 16.8 16.5 16.2 ...
##  $ Slice: num  1.69 1.69 1.69 1.69 1.69 1.69 1.69 1.69 1.69 1.69 ...
```

```
str(antHead)
```

```
## 'data.frame': 74 obs. of  8 variables:
##  $ Frame: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Area : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Mean : int  68 158 162 163 180 176 181 179 179 179 ...
##  $ Min  : int  68 158 162 163 180 176 181 179 179 179 ...
##  $ Max  : int  68 158 162 163 180 176 181 179 179 179 ...
##  $ X    : num  9.05 9.53 9.9 10.15 10.44 ...
##  $ Y    : num  16.4 16.3 16.2 16.1 16.1 ...
##  $ Slice: num  0.021 0.021 0.021 0.021 0.021 0.021 0.021 0.021 0.021 0.021 ...
```

What we need to do is to iterate row by row and to calculate the mean of the x values of head and abdomen as well as calculate the y values of the head and abdomen. In order to do this, we can use a for loop, but we need to setup an empty output vector which we can use to save our calculations to. We will call these vectors centerX and centerY.

```
# setup empty output vectors of appropriate length
centerX <- rep(NA, length = nrow(antAb))
centerY <- centerX

for (i in 1:nrow(antAb)) {
    centerX[i] <- (antAb$X[i] + antHead$X[i])/2
    centerY[i] <- (antAb$Y[i] + antHead$Y[i])/2
}

output <- data.frame(x = centerX, y = centerY)
head(output)
```

```
##        x     y
## 1  9.264 16.60
## 2  9.424 16.61
## 3  9.527 16.49
## 4  9.666 16.31
## 5  9.905 16.19
## 6 10.107 16.12
```

2.2 Using different i's

It's important to note that your values of i don't have to be sequential integers. For example, We talked breifly last week about how we could replace our "1:10" statement instead with a sequence function. Another clever manipulation of the iterative value is using a vector of column names of a dataset in order to perform column operations. To demonstrate this idea, we'll look at the geospiza dataset from a previous lesson. What if we were interested in the range of each column? In that case, we would want to go from column to column and find the difference of the max and min values.

```r
geospiza <- read.table("geospiza.txt", header = T)
str(geospiza)
```

```
## 'data.frame': 13 obs. of  5 variables:
##  $ wingL  : num  4.4 4.35 4.22 4.26 4.24 ...
##  $ tarsusL: num  3.04 2.98 2.9 2.93 2.89 ...
##  $ culmenL: num  2.72 2.65 2.28 2.62 2.41 ...
##  $ beakD  : num  2.82 2.51 2.01 2.14 2.36 ...
##  $ gonysW : num  2.68 2.36 1.93 2.04 2.22 ...
```

```r
# need a vector of column names
colNames <- names(geospiza)
colNames  #note that this is a vector of characters
```

```
## [1] "wingL"   "tarsusL" "culmenL" "beakD"   "gonysW"
```

```r
# setup output
output <- rep(NA, length = length(colNames))
# still need an iterator
j <- 1

for (i in colNames) {
    tempRange <- max(geospiza[i]) - min(geospiza[i])
    output[j] <- tempRange
    j <- j + 1
}

output
```

```
## [1] 0.4443 0.4640 0.7502 1.6325 1.2748
```

2.3 Nested loops

In the above example, we used two different iteration variables (i and j) to attack our problem. We can also use two different iteration variables to help navigate and perform calculations on a matrix. Often we nest loops within loops when we want to include an element of simulation as well. Let's build off last week's idea of generating random walks by using a nested loop to generate multiple random walks with one block of code. Also, instead of using i and j, we'll use more descriptive variables in our loops to prevent confusion as our code grows.

```r
# setup our output to be a matrix, each row representing a
# different simulation
numberOfSims <- 10
time <- 10

output <- matrix(data = NA, nrow = numberOfSims, ncol = time)

for (simNum in 1:numberOfSims) {
    #for 10 simulations
    # code for a random walk should go in here
    x <- 1:time
    y <- rep(0, length = time)
```

```
    for (t in 2:time) {
        y[t] <- y[t - 1] + sample(x = c(-1, 1), size = 1)
    }
    # save our result to our output
    output[simNum, ] <- y
}

output
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    1    0    1    2    1    0    1    2     3
##  [2,]    0    1    2    1    2    3    2    1    2     1
##  [3,]    0    1    0    1    2    3    4    3    2     1
##  [4,]    0   -1   -2   -3   -2   -1   -2   -3   -2    -1
##  [5,]    0   -1    0   -1   -2   -1    0    1    2     1
##  [6,]    0   -1    0    1    0   -1    0    1    2     1
##  [7,]    0   -1   -2   -3   -2   -1   -2   -3   -2    -1
##  [8,]    0   -1   -2   -3   -2   -3   -4   -3   -2    -1
##  [9,]    0   -1    0    1    0    1    2    1    0     1
## [10,]    0    1    0    1    0   -1    0    1    2     1
```

3 Exercises

1. Using a double-nested loop, create a 10x10 multiplication table.
2. Convert our looped-random walk code into a function with inputs numberOfSims and time.