Control Structures and Random Numbers

Jordan Bird and Daniel Schwartz

04 January, 2019

Overview

In Dante's satire of hell each layer contains a new iterative task for the tortured to endure for eternity. Much of the power of computing stems from the efficiency of computers in performing small tasks in an iterative manner. This lesson introducts the concepts of control structures and random numbers. We will use these tools to understand the diversity of jellybeans flavors, but these basic tools are useful for a numbers of tasks in computing and will help you automate many of your future work flows.

Control structures allow you to control the flow of execution of a script. Today, we will focus on the "for loop", but other common control structures include: "if, else" "while" and "next"

A quick tutorial of common control structures in R can be found at https://ramnathv.github.io/pycon2014-r/learn/controls.html

Before we get started lets clear up our session.

```
# clear up the environment
rm(list = ls())
#clear the console
cat("\f")
```

the if and else logical tests

Short intro: when and why to use 1. syntax 2. example 3. "if" accepts logical values; can accept numbers but not strings. 4. "if" can be complemented by an "else" 5. ifelse as shorthand 6. if-else can be strung to create hirearchy of tests

The for loop

Let's start by introducing the "for loop" as a control structure.

Syntax The specific syntax of for loops will look different in other language but basics syntax remains the same. 1. Start a loop with for 2. Declare variable and number of iteration 3. Tasks to perform in code block

```
                    1.      2.          3.
```

In R this looks like: for(i in 1:10){ print(i) }

```
for(i in 1:10){
  print(i)
}
```

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

```
## [1] 9
## [1] 10
```

In this sturcture i is a variable that will hold each interger from 1 to 10 during each successive pass through the loop.

You can use this structure to move through each item of vector and perfrom any number of tasks to each element.

Like printing out a list of fruits.

```
x <- c("apples", "oranges", "bananas", "strawberries")

for (i in seq(x)) {
    print(x[i])
}
```

```
## [1] "apples"
## [1] "oranges"
## [1] "bananas"
## [1] "strawberries"
```

Note that a for loop can iterate over any type of vector: logical, numerical, string. You can also iterate over vectors using indices (R counts from 1 and not form 0 like some other languages)

This chunk below seems to be going off topic with strings. That might need to be developed as an independrnt unit. Or printing out a list of fruit.

```
if (!require("stringr")) install.packages("stringr")
```

```
## Loading required package: stringr
```

```
library(stringr)

for (i in seq(x)) {
    print(str_sub(x[i],1,-2))
}
```

```
## [1] "apple"
## [1] "orange"
## [1] "banana"
## [1] "strawberrie"
```

Here we saw an example of how two control structures could be used together. If the first statement allowed use to check and see if the proper library for handling strings was loaded on to our R packages and the second gave us an inventive new spelling for the word strawberry.

Note on the use of for loops When possible try to avoid them. They take a long time to execute and are somewhat clumsy. Consider using apply type functions.

combining for and if

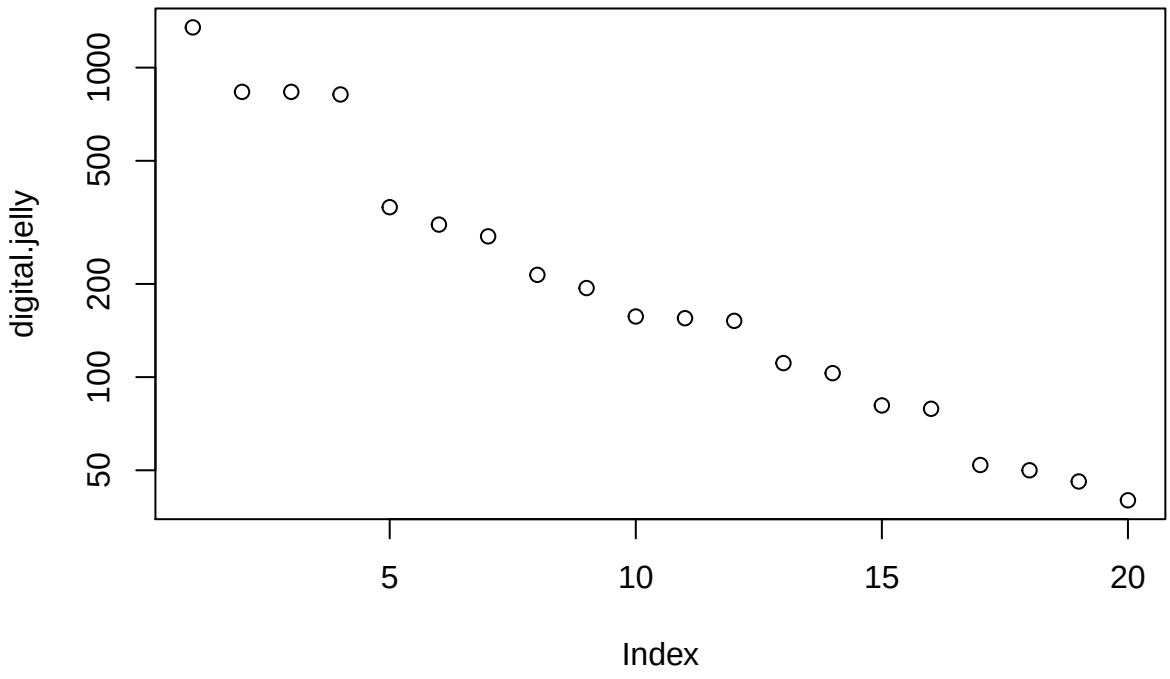
e.g. print only numbers that can be divided by 3

**mention that there are also 'while' loops. Need catioun to have a breaking point

Using loops and ifs to simulate Jelly bean exersice

1. Generate a community by drawing from lognormal distribution

```r
set.seed(6)
digital.jelly <- round(rlnorm(n = 20, meanlog = 5 ))
digital.jelly <- sort(digital.jelly,decreasing = T)
names(digital.jelly) <- paste0("species.",LETTERS[seq(digital.jelly)])
plot(digital.jelly, log='y')
```



Explain the details of commands above

2. Sample community Function to generate a single sample

```r
sample.community <- function (x,n){
  survey <- sample(seq(x),size = n, replace = T,prob = x)
  survey.sum <- rep(0, length(x))
  for (i in survey){
    survey.sum[i]<-survey.sum[i]+1
  }
  return(survey.sum)
}
```

3. Calculate for alpha diversity parameters for single samples and for source community

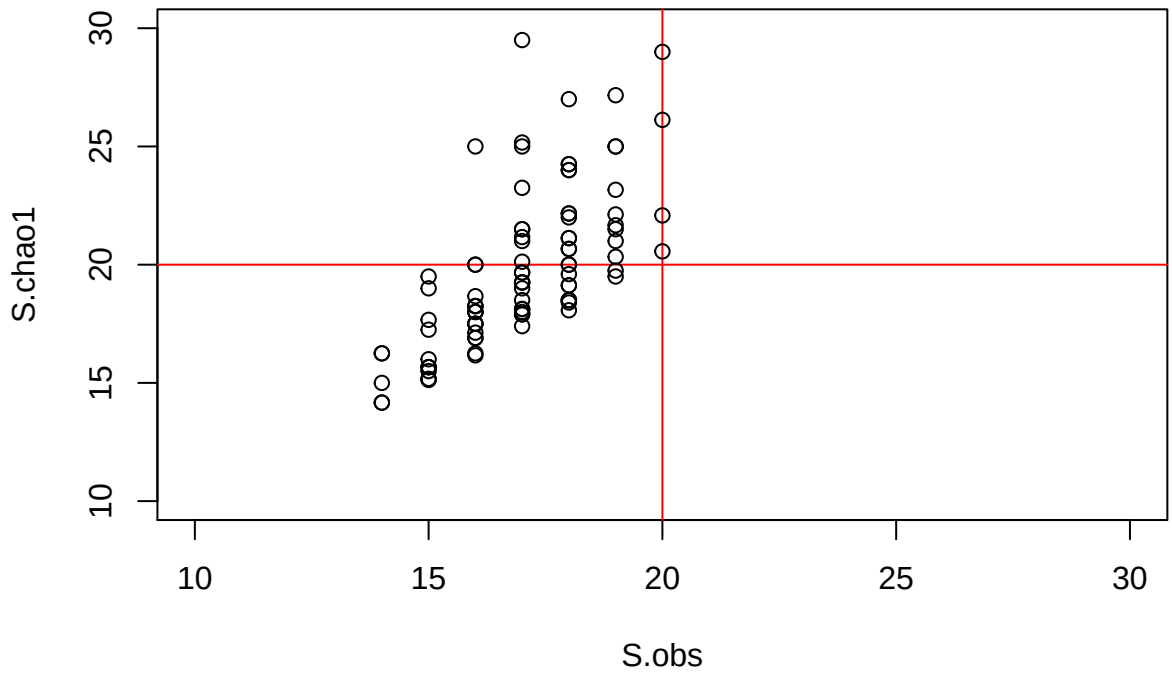
```r
# Functions from alpha diversity handout
S.obs <- function(x = ""){
  rowSums(x > 0) * 1
}

S.chao1 <- function(x = ""){
  S.obs(x) + (sum(x == 1)^2) / (2 * sum(x == 2))
```

```
}
```

4. repeat sampling and calculations 100 times using for loop and look at estimated prameter distribution along with real value. **The code below works well as script but not in markdown... *

```
S.true <- length(digital.jelly)
plot(0, t="n", xlim=c(S.true/2, 1.5*S.true), ylim=c(S.true/2, 1.5*S.true),
     xlab="S.obs", ylab="S.chao1")
abline(h=S.true,v = S.true, col="red")
N <- 100
for ( i in 1:100){
  sample.dj <- sample.community(digital.jelly,N)
  x<-S.obs(t(sample.dj))
  y<-S.chao1(t(sample.dj))
  points(x,y)
}
```



Play around with the sample size (by changing the value of `N`). How does sampe size affect observed and estimated fitness?

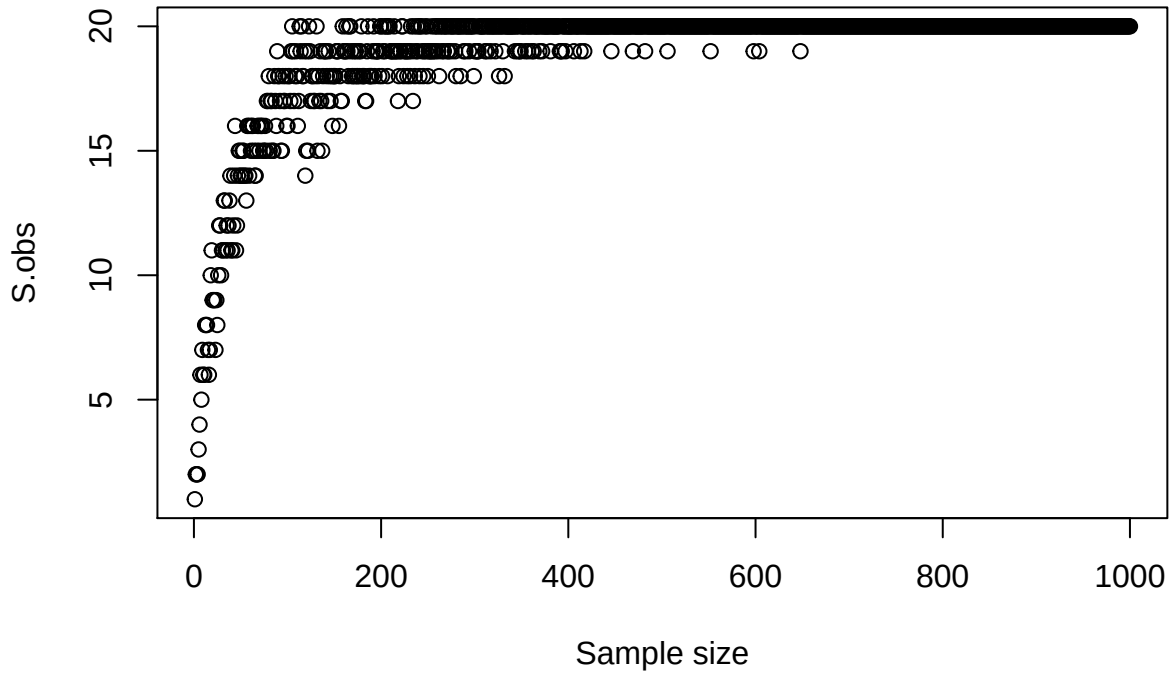
4B. Collectors curve

S.observed as a function of N. First take a single sample of size N from the source population. Subsample you sample with sampe sizes increasing from 1 to N and note the observed richness for each subsample.

```
#first take a single sample of size N from the source population
N<-1000
sample.dj <- sample.community(digital.jelly,n = N)
```

```r
#generate vector to store results
S.collectors <- rep(NA, N)
#loop through 1:N and check S.obs
for (i in 1:N){
  current <-  sample.community(sample.dj,n = i)
  S.collectors[i] <- S.obs(t(current))
}

plot(1:N, S.collectors, xlab = "Sample size",ylab = "S.obs")
```



What is the samllest sampe size to indicate the true richness? What is the largest sample size to underestimate richness? What sample size wiuld you recommend for the next survey of ths community?

5. using for loop generate multiple communities and repeat 2-4 with beta diversity