BTRY4830_Lab5

Olivia Lang

2/28/2019

1. Pseudo Random Numbers
2. paste() function
3. Speeding up your code
4. Exercise - The Sampling Distribution of the Mean and P values

1. Pseudo Random Numbers

- Generating random numbers may sound like a fairly simple task, but if you think about it and actually try to implement a method for it you will quickly notice that it is a quite difficult problem.
- The basic mechanism is that the functions starts at a certain number (seed), and generates a sequence of numbers that look "random enough".
- Believing your numbers are random when there is in fact some pattern can become a problem if you are trying to make a super secret code (RAND book of 1,000,000 random numbers, diceware)
- We have already seen a random number generator in action with the rnorm() function which generates random numbers drawn from a normal distribution.
- Today we will take a look at the more general function sample(). This function allows you to generate random numbers from a sequence with assigned probabilities.
- The following function call will generate 3 numbers drawn from (1,3,5,7,9) without replacement.

```
sample(x = c(1,3,5,7,9),
       size= 3,
       replace = FALSE)
```

```
[1] 7 1 3
```

- When you set replace = TRUE, the same number can appear multiple times in the sample.

```
for( index in 1:5){
  sample.temp <- sample(x= c(1,3,5,7,9),
                        size= 3,
                        replace = TRUE)
  cat("Sample #",index,"=",sample.temp, "\n")
}
```

```
Sample # 1 = 3 9 1
Sample # 2 = 7 7 5
Sample # 3 = 1 7 9
Sample # 4 = 5 1 5
Sample # 5 = 3 5 7
```

- You can also set specific probabilities for the numbers if you are interested in running a casino.

```
for( index in 1:5){
  sample.temp <- sample(x= c(1,3,5,7,9),
                        size= 3,
                        replace = TRUE,
```

```
                          prob = c(0.1,0.1,0.1,0.1,0.6))
  cat("Sample #",index,"=",sample.temp, "\n")
}
```

```
Sample # 1 = 9 5 7
Sample # 2 = 5 9 3
Sample # 3 = 9 5 5
Sample # 4 = 9 3 9
Sample # 5 = 9 1 3
```

- Note that the sample function also works for strings as well. This will come in handy when we have to simulate genotype values later in the course.

```
sample(x = c("A","T","G","C"),
       size= 10,
       replace = TRUE,
       prob = c(0.3,0.3,0.2,0.2))
```

```
 [1] "T" "T" "C" "T" "A" "T" "A" "C" "G" "G"
```

- It is sometimes very useful to generate the exact same results from processes that involve random numbers. You can check if your code is doing the right thing, and if not you can probably find out where it is wrong. We can set a seed, which is used as a reference point to generate random numbers, to generate identical not-so-random-anymore numbers.

```
set.seed(2018)
sample(x = 1:100,size = 10, replace = FALSE)
```

```
 [1] 34 46  6 20 99 29 58 13 89 50
```

```
set.seed(2018)
sample(x = 1:100,size = 10, replace = FALSE)
```

```
 [1] 34 46  6 20 99 29 58 13 89 50
```

- The same is true for the random sampling functions from specific distributions.

```
set.seed(2002)
rnorm(10)
```

```
 [1] -0.10659101  1.61444509  1.38826285  0.70277776  2.25750025
 [6] -0.19649959  1.69685022 -2.35790439 -0.08141469 -0.68264759
```

```
set.seed(2002)
rnorm(10)
```

```
 [1] -0.10659101  1.61444509  1.38826285  0.70277776  2.25750025
 [6] -0.19649959  1.69685022 -2.35790439 -0.08141469 -0.68264759
```

2. paste() function

- paste() is a very useful function when you have to combine constant and changing information.
- For example, if you want to generate multiple files with the same prefix you can use the paste function like this:

```
for( i in 1:5){
  file.name <- paste("QG18","Lab5","file",i,sep = "_")
  # you can add as many elements as you want within the parentheses, the sep option specifies the separ
```

```
  file.name <- paste(file.name, "txt", sep = ".")
  cat(file.name, "\n")
}
```

```
QG18_Lab5_file_1.txt
QG18_Lab5_file_2.txt
QG18_Lab5_file_3.txt
QG18_Lab5_file_4.txt
QG18_Lab5_file_5.txt
```

3. Speeding up your code

- Say you want to do make some calculation 1,000 times on 1,000 different samples. One way to collect these answers is shown below

```
answer <- c()

for(i in 1:30000){
  mySample <- rnorm(1000)
  answer <- c(answer,mean(mySample))
}
print(mean(answer))
```

```
[1] -0.0001035967
```

- Wow! That took a while, to time exactly how long use

```
answer <- c()
startTime <- proc.time()  #
for(i in 1:30000){
  mySample <- rnorm(1000)
  answer <- c(answer,mean(mySample))
}
print(mean(answer))
```

```
[1] -0.0003910237
```

```
endTime <- proc.time()    #
print(endTime-startTime)  #
```

```
   user  system elapsed
  6.105   0.827   6.951
```

- A better way to collect answers is to preset the vector you will store the answer in

```
answer <- numeric(length = 1000)   #
startTime <- proc.time()
for(i in 1:30000){
  mySample <- rnorm(1000)
  answer[i] <- mean(mySample)
}
print(mean(answer))
```

```
[1] 0.0001584234
```

```
endTime <- proc.time()
print(endTime-startTime)
```

```
   user  system elapsed
  3.345   0.010   3.371
```

- By presetting your vector your debugging will be easier too, since any error that occurs in a preset vector will produce a message whereas anything could happen with a vector c()
- To really supercharge this process let's try the apply function
- The apply function is like a one function for that can only be applied to a dataframe or matrix

```
myMatrix <- matrix(1:9,nrow=3)
for(i in 1:nrow(myMatrix)){
  rowMean <- mean(myMatrix[i,])
  print(rowMean)
}
```

```
[1] 4
[1] 5
[1] 6
```

- The first argument is the matrix that will be evaluated, the second argument is either 1 (to loop over rows) or 2 (to loop over columns), the third argument is the function that will be applied on each row or column

```
myMatrix
```

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

```
apply(myMatrix,1,mean)
```

```
## [1] 4 5 6
```

```
# apply(myMatrix,2,mean)    # what is the output of this line?
```

- Now let's compare to our previous example, but first we need to fill a matrix

```
startTime <- proc.time()
normMat <- matrix(rnorm(1000*30000),nrow=30000)
answer <- apply(normMat,1,mean)    #
endTime <- proc.time()
print(endTime-startTime)
```

```
##    user  system elapsed
##   4.522   0.704   5.405
```

- It's a little faster! The difference becomes more noticeable when the actual function becomes more complex

4. Exercise - The Sampling Distribution of the Mean and P values

Turn in the code as an Rmarkdown or Rscript file to CMS.

Part 1 - Density

```
# - Create 100 samples from a normal distribution with a sample size of 50, mean = 0, and sd = 1.
#

# - For each sample calculate the mean and save it to a vector.
```

```
#

# - Plot the density for the sample means (there should be 100 sample means) and calculate the mean and
```

Part 2 - P-Values

- Use *pnorm()* to calculate p-values for each sample mean using the sampling distribution of the mean, which should have slightly different parameters compared to the original distribution.
- Plot a histogram for the p-value distribution, and show the number of times where we reject the null hypothesis (mean = 0) with a threshold of alpha = 0.05 and print it out. Keep in mind that in this case we are testing if the mean is **different** from 0, not larger or smaller.

Part 3 - Functionize

- Generate a function based on the work you have done, so that you can change the number of samples, sample size, and the mean and standard deviation of the population that we are sampling from.
- The output of the function should look something like this.

```
set.seed(777)
sampling_function(sample_size = 100,
                  number_of_samples = 1000,
                  population_mean = 0,
                  population_sd = 1)
```

Let's do the first part together!

```
sampling_function <- function(sample_size, number_of_samples, population_mean, population_sd){
  normSamples <- matrix(0,nrow=number_of_samples,ncol=sample_size)
  for(i in 1:number_of_samples){
    normSamples[i,] <- rnorm(sample_size)
  }
  # param values to use in pnorm
  sampleMeans <- apply(normSamples, 1, mean)
  sampleSDs <- apply(normSamples, 1, sd)


  plotDf <- data.frame(sampleMeans)
  head(plotDf)

  # basicR plot
  hist(sampleMeans, probability = TRUE)
  line( density(sampleMeans) )

  # ggplot
  library(ggplot2)
  ggplot(plotDf,aes(sampleMeans))+geom_density()
  # abs value to flip negatives and calculate p-value from the right side
  p_values <- 2*pnorm(abs(sampleMeans), mean = population_mean, sd = population_sd/sqrt(sample_size), l
  # halve alpha to make it a two sided test
  alpha <- 0.025
  hist(p_values)
  reject_num <- sum(p_values < alpha)
  print(paste0('The null is rejected ', reject_num, ' times'))
```

```
}
set.seed(777)
sampling_function(sample_size = 50,
                  number_of_samples = 100,
                  population_mean = 0,
                  population_sd = 1)
```