STA 141A
Fundamentals of Statistical Data Science

Fall 2016

Instructor: Debashis Paul

Lecture 5

for loop vs. sapply( )

- We can use for loop to repeat a task by cycling over the elements of a vector

```
x = c(4.1,2.3,-2.2,3.1,4.0,3.3); y = numeric(0)
for(i in 1:length(x)){
  y[i] = sum(x[1:i]^2)   # sum of x[1:i]^2
}
```

- We can use sapply( ) to avoid using the for loop in the above example

```
y = sapply(1:length(x), function(i) {sum(x[1:i]^2)})  # note we have defined a function inside sapply
```

- We can get the same answer as above by using the following command:

```
cumsum(x^2) # computes cumulative sum of x^2
```

Logical control: while, if, else

- We can control the flow of logical statements by using **while** (as opposed to for) **if** and **else** statements
- Example: A random walk hitting a barrier

```
x = rnorm(100) # generate 100 i.i.d. samples from of N(0,1)
m = 1; hits = numeric(0)
while(m <= 30){
  if(sum(x[1:m]) < -1)  {hits[m] = -1} # if sum of x[1] … x[m] is < -1 set the value of hits[m] to -1
  else if(sum(x[1:m]) > 1) {hits[m] = 1} # if sum of x[1] … x[m] is < -1 set the value of hits[m] to 1
  else {hits[m] = 0}
  m = m+1
}
```

ifelse( )

- We can vectorize if-then-else statements using ifelse( ) function

```
u = c("and", "read","sleep","where","next")
v = ifelse(u == "read" | u == "sleep", T,F)  # if an element of u is either "read" or
#"sleep" set the corresponding element of v to TRUE, otherwise to FALSE
u %in% c("read", "sleep") # returns the same result as v
```

- Implement the random walk example using ifelse( )

```
x = rnorm(100)
y = cumsum(x) # cumulative sum of x
hits = ifelse(y < -1, -1, ifelse(y > 1,1,0))
```

Factors

- Factors are used typically to represent *nominal* or *categorical* variables.
- An R factor may simply be viewed as a numeric or a character vector, but with extra information, named **levels** of the factor.

```
x = c(5,12,10,5,12,13)

xf = factor(x) # a factor with four levels: 5,10,12,13, i.e., the distinct values of x

str(xf)  or unclass(xf) # reveals the structure of the factor xf
```

- We can manipulate the levels of a factor

```
xflong = factor(x,levels=c(5,10,12,13,15))  # creating an additional level "15"
```

tapply( ) and split( )

- tapply( ) splits a vector (first argument) into groups, according to the levels of the second argument (a factor) and applies a function to the groups
- split( ) only splits a vector into groups according to the levels of the second argument

```
age = c(19,20,21,21,21,20,19,20)
GPA = c(2.9,3.7,3.4,3.6, 2.5,3.1,4.0,3.2)
gender = c("M", "F","M", "M","F", "F", "F","M")
tapply(GPA,age,mean) # split GPA according to age and then compute mean
tapply(GPA,gender,sd) # split GPA according to gender and then compute s.d.
split(GPA,gender) # returns a list with two fields: "M" and "F"
split(GPA,list(age,gender)) # splits the vector GPA into groups according to age and gender
```

Tables

- Tables are effective data structures for representing categorical variables.
- We can use function table( ) to create a frequency distribution in the form of a contingency table

```
x = c(5,10,12,14,10,5,12,8)
table(x) # returns the distinct values of vector x together with their frequencies
y = table(data.frame(gender,age)) # creates two-way contingency table
```

- We can treat tables like matrices or arrays and apply the corresponding operations (remember Titanic)

```
dimnames(y) # returns the list consisting of levels of fields gender an age
apply(y,2,sum) # returns column sums (frequencies for different levels of age)
```

Example: Iris

- Data set iris contains multivariate measurements on three species of iris flowers. We want to compare their features.

```
names(iris) # returns c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")
attach(iris) # use during the current R session (remember to close with detach(iris) )
table(Species) # frequency distribution of Species
tapply(Petal.Length,Species,summary)  # summary of Petal.Length categorized by Species
par(mfrow=c(1,3)); tapply(Petal.Length,Species,hist)  # Histogram of Petal.Length by Species
iris1and2.by5 = split(iris[,1:2],iris[,5]) # extract data on Sepal.Length and Sepal.Width by Species
par(mfrow=c(1,3)); sapply(iris1and2.by5,plot) # plots of Sepal.Width vs. Sepal.Length by Species
par(mfrow=c(1,3)); sapply(1:3,function(i) {plot(iris1and2.by5[[i]],main=names(iris1and2.by5)[i])})
```

Working with random variables

- R uses 4 types of functions: (i) to generate random variables; (ii) to compute pdf/pmf of their distributions; (iii) to compute probabilities; (iv) to compute quantiles of the distributions.
- Uniform distribution: **runif( ), duinf( ), punif( ), qunif( )**
- Normal distribution: **rnorm( ), dnorm( ), pnorm( ), qnorm( )**
- Other examples: **rbinom( ), rpois( ), rexp( ), rchisq( ), rf( ), rgamma( )**
- For **reproducibility** of the results, we would like to set the random seed to a specified value

```
rnorm(10) # generate a sample of 10 i.i.d. N(0,1) random variables  (repeat a few times)
set.seed(171); rnorm(10)  # set random seed to 171, then generate 10 i.i.d. N(0,1) r.v. (repeat)
```