Introduction to Computer Programming with R (FOR 6934)

Qing Zhao (School of Forest Resources & Conservation, qing.zhao@ufl.edu)
Daijiang Li (Department of Wildlife Ecology & Conservation, dli1@ufl.edu)
Denis Valle (School of Forest Resources & Conservation, drvalle@ufl.edu)

Class fourteen

- Create a function to summarize tree plot data
- Create a function to conduct power analysis
- Some general comments about programing

Sample code

Simulate number of trees in plots

```
x <- rep(1:10, times=10)
y <- rep(1:10, each=10)
z <- rpois(100, 2.5)
dat <- cbind(x, y, z)
head(dat, n=3)
```

```
##      x y z
## [1,] 1 1 3
## [2,] 2 1 2
## [3,] 3 1 0
```

Sample code

Plot the simulated data

```
par(mar=c(1,1,1,1))
plot(x, y, type='n', xlim=c(0,11), ylim=c(0,11), axes=F, xlab='', ylab='')
text(x, y, z)
for (i in 1:11) {
  lines(x=c(i,i)-.5, y=c(0,10)+.5)
  lines(x=c(0,10)+.5, y=c(i,i)-.5)
}
```

1	1	1	2	2	4	3	2	1	4
3	2	3	3	2	2	3	2	1	6
3	2	1	2	4	1	1	4	1	7
0	4	1	3	2	1	3	5	4	3
1	3	6	2	3	1	2	1	3	1
1	2	4	2	2	3	2	1	3	3
2	3	1	4	5	0	2	0	5	1
3	0	2	2	0	0	4	2	3	3
1	3	2	1	2	1	1	3	2	1
3	2	0	1	3	3	4	1	0	2

I want to summarize the data into larger plots

1	1	1	2	2	4	3	2	1	4
3	2	3	3	2	2	3	2	1	6
3	2	1	2	4	1	1	4	1	7
0	4	1	3	2	1	3	5	4	3
1	3	6	2	3	1	2	1	3	1
1	2	4	2	2	3	2	1	3	3
2	3	1	4	5	0	2	0	5	1
3	0	2	2	0	0	4	2	3	3
1	3	2	1	2	1	1	3	2	1
3	2	0	1	3	3	4	1	0	2

Sample code

Create new coordinates

```
newx <- ceiling(x / 2)
newy <- ceiling(y / 2)
id <- paste(newx, newy, sep='-')
head(cbind(x, y, newx, newy, id), n=6)
```

```
##      x   y   newx newy id
## [1,] "1" "1" "1"  "1"  "1-1"
## [2,] "2" "1" "1"  "1"  "1-1"
## [3,] "3" "1" "2"  "1"  "2-1"
## [4,] "4" "1" "2"  "1"  "2-1"
## [5,] "5" "1" "3"  "1"  "3-1"
## [6,] "6" "1" "3"  "1"  "3-1"
```

Sample code

Sum data

```
out <- tapply(z, INDEX=id, FUN=sum)
out <- data.frame(names(out), cbind(out))
```

```r
names(out) <- c('id', 'newz')
out$newx <- as.numeric(substr(out$id, start=1, stop=1))
out$newy <- as.numeric(substr(out$id, start=3, stop=3))
head(out, n=3)
```

```
##      id newz newx newy
## 1-1 1-1    9    1    1
## 1-2 1-2    8    1    2
## 1-3 1-3    7    1    3
```

The results look like this:

1	1	1	2	2	4	3	2	1	4
3	2	3	3	2	2	3	2	1	6
3	2	1	2	4	1	1	4	1	7
0	4	1	3	2	1	3	5	4	3
1	3	6	2	3	1	2	1	3	1
1	2	4	2	2	3	2	1	3	3
2	3	1	4	5	0	2	0	5	1
3	0	2	2	0	0	4	2	3	3
1	3	2	1	2	1	1	3	2	1
3	2	0	1	3	3	4	1	0	2

7	9	10	10	12
9	7	8	13	15
7	14	9	6	10
8	9	5	8	12
9	4	9	9	5

Make a function to re-use the code

I want to make a customized function so that next time I can re-use the code to sum the numbers into plots of a different size, say 5 x 5 plots

```r
sum.plot <- function(x, y, z, nplot.to.combine) {
  # I simply copy-paste the above code with small (but important) changes
  newx <- ceiling(x / nplot.to.combine) # make the number of plots to combine a variable
  head(cbind(x, newx), n=6)
  newy <- ceiling(y / nplot.to.combine)
  id <- paste(newx, newy, sep='-')
  out <- tapply(z, INDEX=id, FUN=sum)
  out <- data.frame(names(out), cbind(out))
  names(out) <- c('id', 'newz')
  out$newx <- as.numeric(substr(out$id, start=1, stop=1))
  out$newy <- as.numeric(substr(out$id, start=3, stop=3))
  return(out) # remember to return the result
}
```

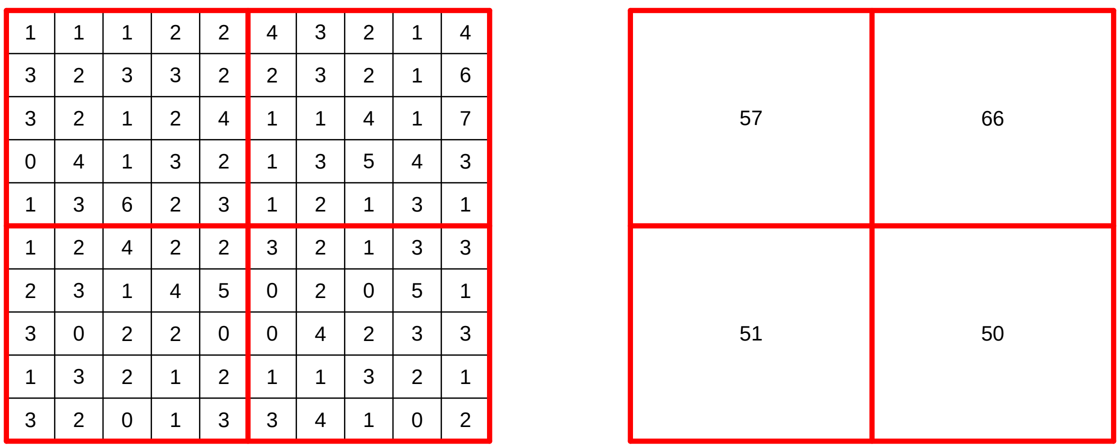
Sample code

Change the values

```
out <- sum.plot(x, y, z, nplot.to.combine=5)
out
```

```
##      id newz newx newy
## 1-1 1-1   51    1    1
## 1-2 1-2   57    1    2
## 2-1 2-1   50    2    1
## 2-2 2-2   66    2    2
```

The results look like this:



Review the code

You only need to use what we have learned in this course here

```
sum.plot
```

```
## function(x, y, z, nplot.to.combine) {
##   # I simply copy-paste the above code with small (but important) changes
##   newx <- ceiling(x / nplot.to.combine) # make the number of plots to combine a variable
##   head(cbind(x, newx), n=6)
##   newy <- ceiling(y / nplot.to.combine)
##   id <- paste(newx, newy, sep='-')
##   out <- tapply(z, INDEX=id, FUN=sum)
##   out <- data.frame(names(out), cbind(out))
##   names(out) <- c('id', 'newz')
##   out$newx <- as.numeric(substr(out$id, start=1, stop=1))
##   out$newy <- as.numeric(substr(out$id, start=3, stop=3))
##   return(out) # remember to return the result
## }
```

```
## <bytecode: 0x000000001807c850>
```

This concludes Class 14, Section 1

Please continue on to the next video

A brief review of type I and type II errors

- A type I error is the incorrect rejection of a true null hypothesis.
- A type II error is the failure to reject a false null hypothesis.

Example

$$X_1, ..., X_n \sim Normal(2, \sigma^2)$$

$$H_0 : \mu = 0$$

$$H_a : \mu \neq 0$$

A type II error here is to "believe" that the mean is 0. Power is 1 minus the probability of making a type II error.

Sample code

Simulate data

```
n <- 10
sd <- 5
x <- rnorm(n=n, mean=2, sd=sd)
x
```

```
##  [1]  6.87024490  2.01951642 -0.09119079  9.42929959 -8.60560845
##  [6] -5.34574651  1.71281934 -3.59585218  0.73448264 -3.63113373
```

Sample code

Conduct student's t test to test if the mean is different from 0

```
t.test(x, mu=0, two.sided=T)
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.028922, df = 9, p-value = 0.9776
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -3.985830  3.885197
## sample estimates:
##   mean of x
## -0.05031688
```

Sample code

Repeat the process for many times to calculate the probability of making a type II error

```
nsim <- 1000
p.value <- numeric(nsim)
for (i in 1:nsim) {
    x <- rnorm(n=n, mean=2, sd=sd)
    p.value[i] <- t.test(x, mu=0, two.sided=T)$p.value
}
power <- 1 - length(which(p.value > 0.05)) / nsim
power
```

```
## [1] 0.226
```

Sample code

Make a function for power analysis

```
power.analysis <- function(nsim, n, sd) {
    p.value <- numeric(nsim)
    for (i in 1:nsim) {
        x <- rnorm(n=n, mean=2, sd=sd)
        p.value[i] <- t.test(x, mu=0, two.sided=T)$p.value
    }
    power <- 1 - length(which(p.value > 0.05)) / nsim
    return(power)
}
```

Sample code

Power is related to sample size n

```
power.analysis(nsim=10000, n=10, sd=5)
```

```
## [1] 0.2054
```

```
power.analysis(nsim=10000, n=20, sd=5)
```

```
## [1] 0.3972
```

```
power.analysis(nsim=10000, n=100, sd=5)
```

```
## [1] 0.9761
```

Sample code

Power is also related to the standard deviation of the normal distribution (given that the mean is 2)

```
power.analysis(nsim=10000, n=10, sd=5)
```

```
## [1] 0.2047
```

```
power.analysis(nsim=10000, n=10, sd=2)
```

```
## [1] 0.8083
```

```r
power.analysis(nsim=10000, n=10, sd=1)
```

```
## [1] 0.9994
```

Sample code

Make another function to calculate power under different sample size and standard deviation

```r
power.analysis.multiple <- function(nsim, n.seq, sd.seq) {
    power.mat <- matrix(, length(n.seq), length(sd.seq))
    for (i in 1:length(n.seq)) {
        for (j in 1:length(sd.seq)) {
            power.mat[i,j] <- power.analysis(nsim, n=n.seq[i], sd=sd.seq[j])
        }
    }
    return(power.mat)
}
```

Sample code

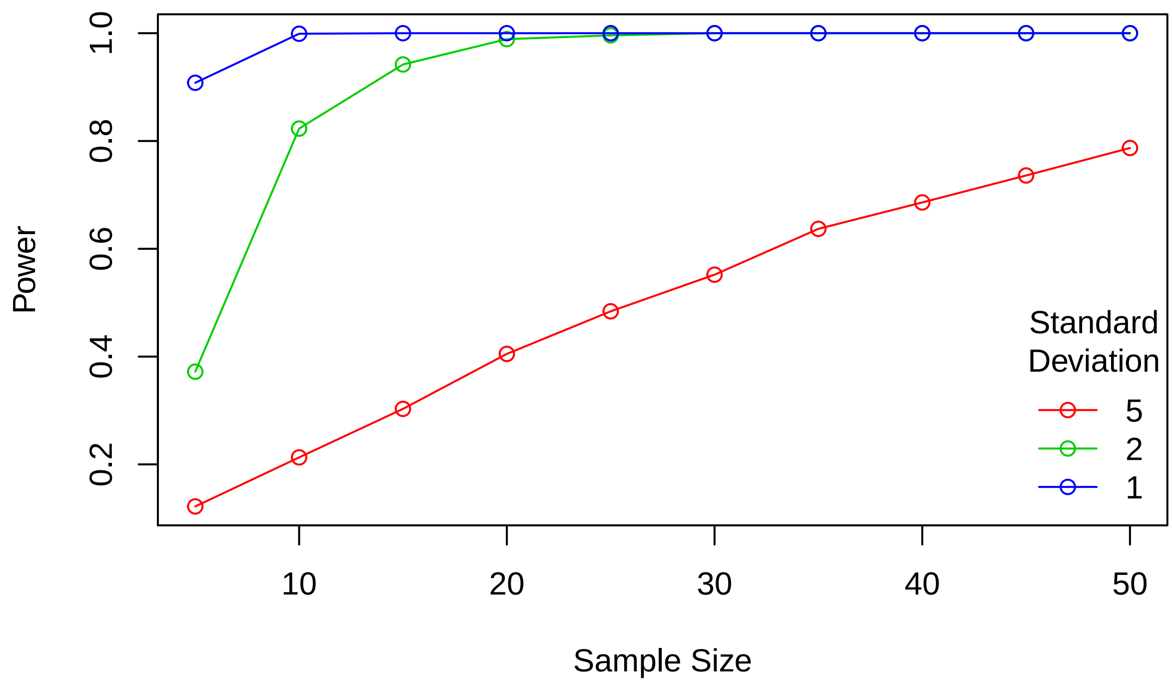
Use the function

```r
nsim <- 1000
n.seq <- seq(from=5, to=50, by=5)
sd.seq <- c(5, 2, 1)
power.mat <- power.analysis.multiple(nsim=nsim, n.seq=n.seq, sd.seq=sd.seq)
```

Sample code

Plot the results

```r
plot(power.mat[,1] ~ n.seq, type='n', ylim=range(power.mat),
     xlab='Sample Size', ylab='Power')
for (i in 1:length(sd.seq)) {
    lines(power.mat[,i] ~ n.seq, type='o', col=i+1)
}
legend('bottomright', pch=1, lty=1, col=2:4, legend=sd.seq,
       title='Standard\nDeviation', bty='n')
```



Review the code

Again, we only need to use what we have learned in this course here

```
power.analysis
```

```
## function(nsim, n, sd) {
##     p.value <- numeric(nsim)
##     for (i in 1:nsim) {
##         x <- rnorm(n=n, mean=2, sd=sd)
##         p.value[i] <- t.test(x, mu=0, two.sided=T)$p.value
##     }
##     power <- 1 - length(which(p.value > 0.05)) / nsim
##     return(power)
## }
## <bytecode: 0x00000000182340f8>
```

```
power.analysis.multiple
```

```
## function(nsim, n.seq, sd.seq) {
##     power.mat <- matrix(, length(n.seq), length(sd.seq))
```

```
##     for (i in 1:length(n.seq)) {
##         for (j in 1:length(sd.seq)) {
##             power.mat[i,j] <- power.analysis(nsim, n=n.seq[i], sd=sd.seq[j])
##         }
##     }
##     return(power.mat)
## }
## <bytecode: 0x0000000021ef0130>
```

This concludes Class 14, Section 2

Please continue on to the next video

Some general comments about computer programing

- Pick your own style and be consistent
  - Naming
- Write your code bit by bit
  - Debug and make sure each piece works, then put them together
- Always learn new things, and be creative
  - Learn by using, practicing

Thank you and enjoy computer programming