Quiz 4

Lingjun Fu, Michael Greer, Richard Rosenow, Zhijun Yin

28 September, 2015

Construct code GroupsMaker (copied from Robert)

```
## Group randomizer using a restricted randomization
# This code restricts the randomization to
# put no more 2 students from the same program
# into a single group.

# embed the randomizer in a function
groupMaker.restricted <- function(theSeed=NA){
  # select a random seed if none is given
#   if(is.na(theSeed)) theSeed = sample(1:10^6, 1)
#   set.seed(theSeed)
  # Identify who is here and not here today.
  groupBios <- c('Michael Ackerman (BIOS)',
                 'Lingjun Fu (BIOS)',
                 'Lisa Lin (BIOS)',
                 'Molly Olson (BIOS)',
                 'Brooklyn Stanley (BIOS)',
                 'Nick Strayer (BIOS)',
                 'Hannah Weeks (BIOS)',
                 'Zhijun Yin (BIOS)'
  )
  groupBmif <- c('Alex Cheng (BMIF)',
                 'Kevin Dufendach (BMIF)' ,
                 'Michael Greer (BMIF)',
                 'Matthew Lenert (BMIF)',
                 'Sara Savage (BMIF)'
  )
  groupMMM <- c('Richard Rosenow (MGT)',
                 'Alexandra Sundermann (MSTP)',
                 'Ivo Violich (MLI)'
  )
  # Initialize groups matrix
  groups <- matrix( rep('', 4*4 ), nrow=4 )
  colnames(groups) <- c("Group1","Group2","Group3","Group4")
  # Shuffle and assemble groups
  shuffledBios <- sample(groupBios)
  shuffledBmif <- sample(groupBmif)
  shuffledMMM <- sample(groupMMM)
  # Stack into the groups matrix satisfying the
  # restriction conditions
  groups[1,1:4] <- shuffledBios[1:4]
  groups[2,1:4] <- shuffledBios[5:8]
  groups[3,1:4] <- shuffledBmif[1:4]
  groups[4,1:4] <- c(shuffledBmif[5],shuffledMMM)
  return(groups)
```

```
}


# Example, run the randomizer allowing a random seed
theGroups <- groupMaker.restricted(4)
print( noquote(theGroups) )
```

```
##      Group1               Group2
## [1,] Molly Olson (BIOS)   Lingjun Fu (BIOS)
## [2,] Zhijun Yin (BIOS)    Lisa Lin (BIOS)
## [3,] Sara Savage (BMIF)   Alex Cheng (BMIF)
## [4,] Michael Greer (BMIF) Alexandra Sundermann (MSTP)
##      Group3                  Group4
## [1,] Nick Strayer (BIOS)     Hannah Weeks (BIOS)
## [2,] Brooklyn Stanley (BIOS) Michael Ackerman (BIOS)
## [3,] Kevin Dufendach (BMIF)  Matthew Lenert (BMIF)
## [4,] Ivo Violich (MLI)       Richard Rosenow (MGT)
```

Question 1:

We construct simulation code as follows. Note that in the code we use two tricks to fix the inner and inter group duplication problems (motivated from discussions with Xianwen Shen). In other words, the inner duplication refers to the fact that the order of members in each group is unimortant: "ABCD" is the same as "BDCA". The inter duplication refers to the fact that group name is unimportant: "ABCD" in group 1 and "EFGH" in group 2 is the same as "EFGH" in group 1 and "ABCD" in group 2. Therefore, given sufficient iteration time our simulation result should reflect the true value without any manual calculation.

```
## rewrite Robert's code by replacing students' name by letters
groupMaker.old <- function(theSeed=NA){
  # select a random seed if none is given
  # if(is.na(theSeed)) theSeed = sample(1:10^6, 1)
  # set.seed(theSeed)
  # Identify who is here and not here today.
  groupBios <- c('A',
                 'B',
                 'C',
                 'D',
                 'E',
                 'F',
                 'G',
                 'H'
  )
  groupBmif <- c('I',
                 'J' ,
                 'K',
                 'L',
                 'M'
  )
  groupMMM <- c('N',
                 'O',
```

```
                'P'
  )
  # Initialize groups matrix
  groups <- matrix( rep('', 4*4 ), nrow=4 )
  colnames(groups) <- c("Group1","Group2","Group3","Group4")
  # Shuffle and assemble groups
  shuffled <- sample(c(groupBios, groupBmif, groupMMM))
  # Stack into the groups matrix satisfying the
  # restriction conditions
  groups[1,1:4] <- shuffled[1:4]
  groups[2,1:4] <- shuffled[5:8]
  groups[3,1:4] <- shuffled[9:12]
  groups[4,1:4] <- shuffled[13:16]
  return(groups)
}

iteration <- 2*2627625  # should be larger than 2627625!
test <- 5000 # should choose test=iteration, choose 5000 for efficiency
res <- rep(NA, test)
for(i in 1:test){
    temp = groupMaker.old()
    temp[,1]=sort(temp[,1])
    temp[,2]=sort(temp[,2])
    temp[,3]=sort(temp[,3])
    temp[,4]=sort(temp[,4]) # fix the inner group duplication
    newtemp=temp[,names(sort(temp[1,]))] # fix the inter group duplication
    res[i] = paste(newtemp, sep="", collapse ="")
    if(i == 1){
        x <- res[i]
    }
    if(i != 1){
        x <- c(x, res[i])
    }
}

length(unique(x)) # note that this is only for iteration=5000
```

```
## [1] 4990
```

```
q1_theory <- factorial(16)/factorial(4)^5 # q1_theory=2627625
q1_theory
```

```
## [1] 2627625
```

For the simulation code, the displayed number here is an example with iterations of 5000. In fact, Zhijun ran it for iterations of 10^8 and get a result around $2 * 10^6$, which has the same magnitude order of our theoretical result 2627625. We argue that given sufficient simulation time our code will eventually give a result converging to the theoretical value.

For the theoretical formula, we got hint from http://math.stackexchange.com/questions/393591/how-many-options-are-there-for-15-student-to-divide-into-3-equal-sized-groups. In the denominator, the first four 4! takes care of the inner group duplication (four groups with four members in each group). The last 4! takes care of the inter group duplication (4! ways to order the groups). We also generalize the formula

to: $\frac{\binom{16}{4}*\binom{12}{4}*\binom{8}{4}*\binom{4}{4}}{4!}$, and we will use this again in Q3. We can understand this formula as follows: choose 4 people from the student pool each time and repeat this 4 times without replacement. Then, we divide it by 4! to account for the group name duplication. Note that in this approach we don't need to worry about the inner duplication because we are not assigning students into specific positions in the group.

Question 2:

```
sim_12 <- 0
test <- 1000
pivot <- c(1:4,1:4,1:4,1:4)
for(i in 1:test){
    temp = sample(pivot, 16, replace=F) # random shuffle the group numbers
    if(temp[1] == temp[2]){  # treat 1st and 2nd student as two L students
        sim_12 = sim_12 + 1
    }
}
sim_12/test #to be compared with theoretical solution: 0.2
```

```
## [1] 0.196
```

```
q2_theory <- choose(4,1)*choose(4,1)*choose(3,1)*factorial(14)/
factorial(16)
q2_theory
```

```
## [1] 0.2
```

This question is straight forward to solve by hand. Assume the probability is equal to $\frac{A}{B}$. A is the number of ways to have Lingjun and Lisa in the same group. B is the number of ways to send all students into 4 groups. We note that we are ignoring inner and inter group duplication here since it won't affect the ratio $\frac{A}{B}$ (big advantage when you only want to know probability). Therefore, B is equal to 16!. A can be derived as following: we have four choices regarding which group to send Lingjun and Lisa (choose(4,1)), then Lingjun and Lisa have $4 * 3$ choices to choose which location to stay in that particular group, finally we send all the left 14 students to fill the left 14 slots (14!).

An intuitive way to think about the answer is: the probability that we send Lingjun to one group is 1. After that, there are $16 - 1 = 15$ slots for Lisa to choose from, among which $4 - 1 = 3$ slots are within the same group with Lingjun. Therefore: $\frac{3}{15} = 0.2$.

In the simulation, we simulate 1000 iterations of and assume the first two students as Lingjun and Lisa. Then we count how many times Lingjun and Lisa has the same group assignment. Since we are simulating large random samples, we assume that the ratio of the number of these two events can be used to estimate the probability. Note that our iteration times here 1000 is far less than the one in Q1. We still get a pretty precise value though.

Question 3:

We construct simulation code in a similar way as what we did in Q1. Note that in the code we use two tricks to fix the inner and inter group duplication problems. In other words, the inner duplication refers to the fact that the order of members in each group is unimortant: "ABCD" is the same as "BDCA". The inter duplication refers to the fact that group name is unimportant: "ABCD" in group 1 and "EFGH" in group 2 is the same as "EFGH" in group 1 and "ABCD" in group 2. Therefore, given sufficient iteration time our simulation result should reflect the true value without any manual calculation.

```
## rewrite Robert's code a little bit by replacing students' names by lettrs
groupMaker.new <- function(theSeed=NA){
  # select a random seed if none is given
  # if(is.na(theSeed)) theSeed = sample(1:10^6, 1)
  # set.seed(theSeed)
  # Identify who is here and not here today.
  groupBios <- c('A',
                 'B',
                 'C',
                 'D',
                 'E',
                 'F',
                 'G',
                 'H'
  )
  groupBmif <- c('I',
                 'J' ,
                 'K',
                 'L',
                 'M'
  )
  groupMMM <- c('N',
                 'O',
                 'P'
  )
  # Initialize groups matrix
  groups <- matrix( rep('', 4*4 ), nrow=4 )
  colnames(groups) <- c("Group1","Group2","Group3","Group4")
  # Shuffle and assemble groups
  shuffledBios <- sample(groupBios)
  shuffledBmif <- sample(groupBmif)
  shuffledMMM <- sample(groupMMM)
  # Stack into the groups matrix satisfying the
  # restriction conditions
  groups[1,1:4] <- shuffledBios[1:4]
  groups[2,1:4] <- shuffledBios[5:8]
  groups[3,1:4] <- shuffledBmif[1:4]
  groups[4,1:4] <- c(shuffledBmif[5],shuffledMMM)
  return(groups)
}

## simulation starts

iteration <- 2*151200  # should be larger than 151200
test <- 50000 # should choose test=iteration, choose 2000 for efficiency
```

```
res <- rep(NA, test)
for(i in 1:test){
    temp = groupMaker.new()
    temp[,1]=sort(temp[,1])
    temp[,2]=sort(temp[,2])
    temp[,3]=sort(temp[,3])
    temp[,4]=sort(temp[,4])  # fix the inner group duplication
    newtemp=temp[,names(sort(temp[1,]))] # fix the inter group duplication
    res[i] = paste(newtemp, sep="", collapse ="")
    if(i == 1){
        x <- res[i]
    }
    if(i != 1){
        x <- c(x, res[i])
    }
}

length(unique(x)) # simulation result to be compared with q3_theory=151200
```

```
## [1] 42665
```

```
#theoretical value
q3_theory <- choose(8,2) * choose(6,2) * choose(4,2) * choose(2,2) *
choose(5,2) * choose(3,1) * choose(2,1) * choose(3,1) * choose(2,1) /
(factorial(1)*factorial(3))
q3_theory
```

```
## [1] 151200
```

For the simulation code, the displayed number here is an example with iterations of 50000. In fact, Zhijun ran it for iterations of $2 * 10^6$ and get a result exactly the same as theoretical value: 151200! We argue that our code has given a precise result since it's converging to 151200 after running a iteration of $1.5 * 151200$. Note that different from Q1, we get a precise result through simulation, though it took a whole night or so.

The theoretical value for this question is 151200. The formula is hinted from Q1. In the numerator, we use all the "choose" terms to calculate how many possibilities there are if we need to fill all the 16 slots with the restriction. Then, we divide it by 1! and 3! to take care of group name duplication. Note that we are not using 4! here because we have two kinds of groups now: 1 group with 2 Bioinformatics students and 3 groups with 1 Bioinformatics student in each. In other words, the "symmetry" is broken.

Question 4:

For this question, we just use Robert's code to randomly generate 4 groups and count how many times Lisa and Lingjun are in the same group. Note that we use "match" function to find the index of the student which is a number between 1 and 16 and "ceiling" function to find the student's group number. For example, a student with index 5 is located in group ceiling(5/4)=2.

```
sim_12 <- 0
test <- 10000
for(i in 1:test){
    temp = groupMaker.restricted()
    Lisa = match("Lisa Lin (BIOS)",temp) # give a number between 1 and 16
    Lingjun = match("Lingjun Fu (BIOS)",temp) # number between 1 and 16
    if(ceiling(Lisa/4) == ceiling(Lingjun/4)){
        #ceiling finds thegroup number
        sim_12 = sim_12 + 1
    }
}
sim_12/test  # 0.1392 for 100000 simulations, to be compared with 1/7
```

```
## [1] 0.1404
```

```
q4_theory <- 1/7
q4_theory
```

```
## [1] 0.1428571
```

This theoretical value is straight forward if you use some symmetry aguments to solve by hand. Due to the constraint that no more than 2 members from the same program in one group, we must send two Bios students to each group. That said, every Bios student has $2 * 4 = 8$ slots to choose from. Now focus on the group where Lingjun is located: every other Bios student has a equal probability of 1/(8-1) to be assigned to the left one slot in this particular group. So does Lisa.

Question 5:

```
z_score <- qnorm(0.88+(1-0.88)/2, mean=0, sd=1)
z_score
```

```
## [1] 1.554774
```

In this formula, LB is:

$$0 - 1.555 * \frac{1}{\sqrt{10}} = -0.492$$

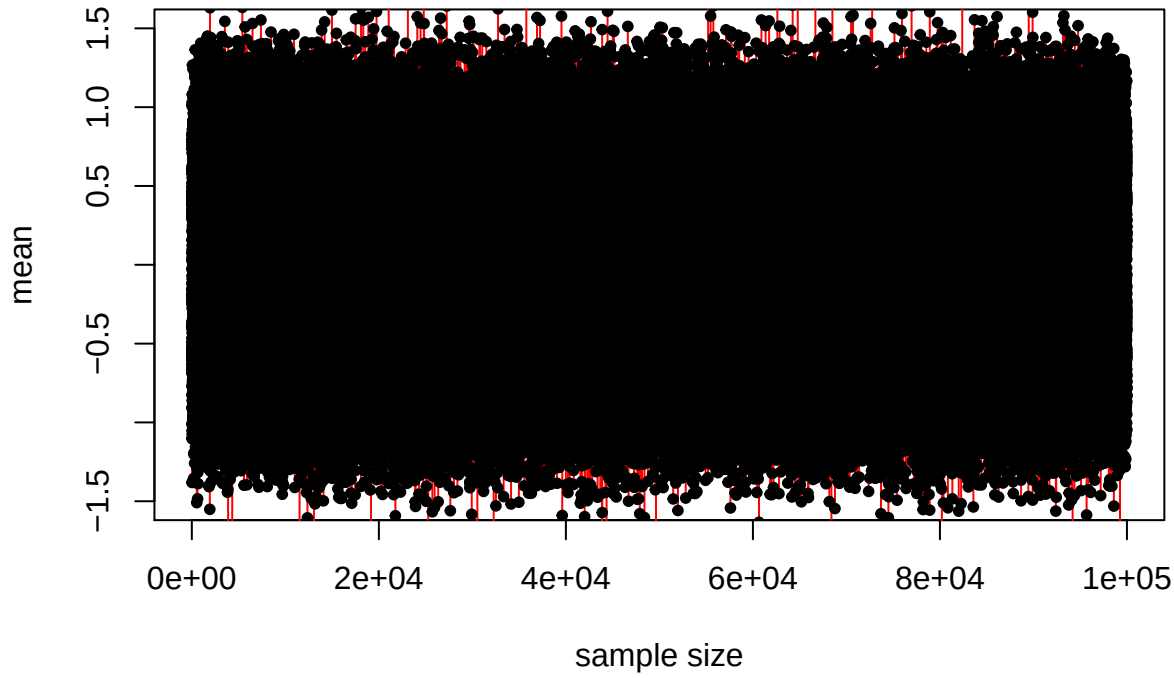
,
UB is:

$$0 + 1.555 * \frac{1}{\sqrt{10}} = 0.492$$

.

Question 6:

```r
plotInV6 <- function(size=100, mean=0,sigma=1) {
  ints <- seq(1:size)
  p1 <- seq(1:size)
  p2 <- seq(1:size)
  for (i in 1:size) {
    x_bar <- mean(rnorm(10, mean=0, sd=1))
    # calculate the upper bound and lower bound
    p1[i] <- x_bar -1.555 * sigma / sqrt(10)
    p2[i] <- x_bar + 1.555 * sigma / sqrt(10)
  }

  # plot the true mean
  plot(seq(1:size), rep(mean, size), type="l", xlab="sample size", ylab="mean", ylim=c(-1.5, 1.5))

  col <- rep(1, size)
  # index those CIs that failed to cover the true mean and plot them with red color
  index <- p1 > mean | p2 < mean
  col[index] <- 2

  # plot the CI segments and points
  segments(ints, p1, ints, p2, col=col)
  points(ints, p1, pch=20)
  points(ints, p2, pch=20)

  return((size-sum(index))/size)
}

plotInV6(size=10^5)
```



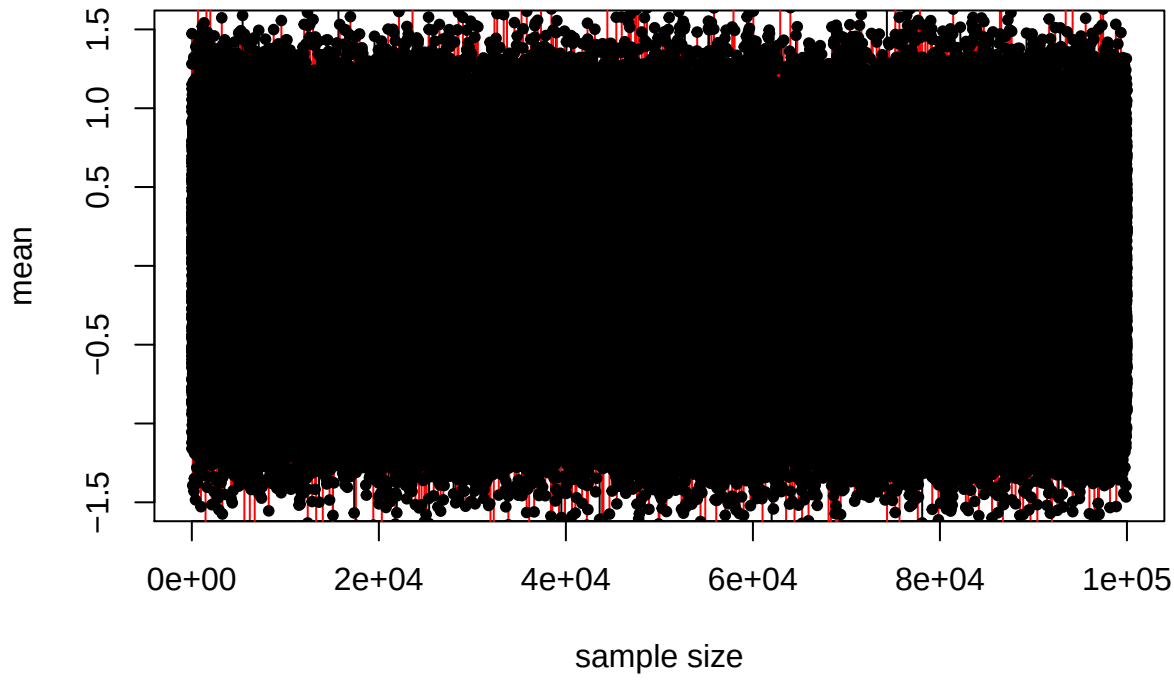

```
## [1] 0.88081
```

Our simulation result is 0.88154 (suppose the size of 10^5 is big enough (by LLN)), which is quite close to the designed 0.88. This is because x_bar is also a normal distribution with mean of 0 and sd of $1\sqrt{(10)}$. Once we know the true σ, the result of question 6 makes sure that we can obtain a quite close cover probability by LLN.

Question 7:

```r
plotInV7 <- function(size=10^6, mean=0,sigma=1) {
  ints <- seq(1:size)
  p1 <- seq(1:size)
  p2 <- seq(1:size)
  for (i in 1:size) {
    samples <- rnorm(10, mean=0, sd=1)
    x_bar <- mean(samples)
    sigma <- sd(samples)
    p1[i] <- x_bar -1.555 * sigma / sqrt(10)
    p2[i] <- x_bar + 1.555 * sigma / sqrt(10)
  }
```

```
  plot(seq(1:size), rep(0, size), type="l", xlab="sample size", ylab="mean", ylim=c(-1.5, 1.5))

  col <- rep(1, size)
  # index those CIs that failed to cover the true mean and plot them with red color
  index <- p1 > mean | p2 < mean
  col[index] <- 2
  segments(ints, p1, ints, p2, col=col)
  points(ints, p1, pch=20)
  points(ints, p2, pch=20)

  return((size-sum(index))/size)
}

plotInV7(size=10^5)
```

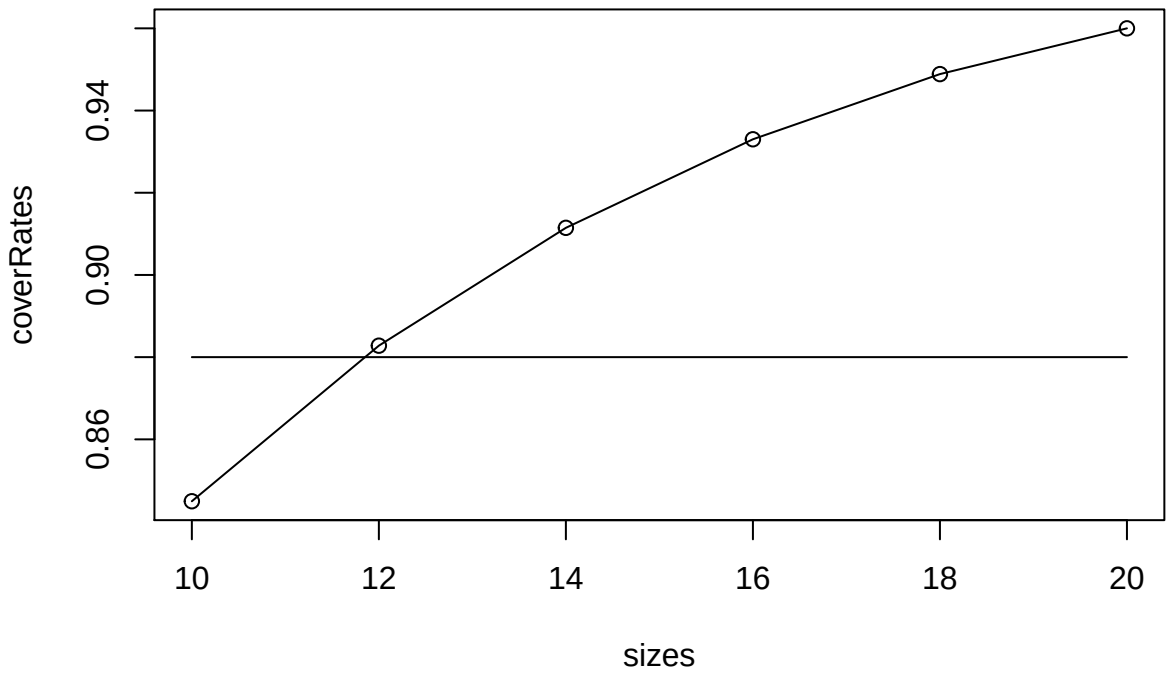



```
## [1] 0.84414
```

The result is 0.8472 which is a little far from the true cover probability 0.88. This is because we have a bad estimation of σ with only 10 samples.

Question 8:

```
plotInV8 <- function(size=10^5, sampleSize=10, mean=0,sigma=1) {
  ints <- seq(1:size)
  p1 <- seq(1:size)
  p2 <- seq(1:size)
  for (i in 1:size) {
    samples <- rnorm(sampleSize, mean=0, sd=1)
    x_bar <- mean(samples)
    sigma <- sd(samples)
    p1[i] <- x_bar -1.555 * sigma / sqrt(10)
    p2[i] <- x_bar + 1.555 * sigma / sqrt(10)
  }

  # plot(seq(1:size), rep(0, size), type="l", xlab="sample size", ylab="mean", ylim=c(-1.5, 1.5))
  # col <- rep(1, size)
  index <- p1 > mean | p2 < mean
  # col[index] <- 2
  # segments(ints, p1, ints, p2, col=col)
  # points(ints, p1, pch=20)
  # points(ints, p2, pch=20)

  return((size-sum(index))/size)
}

sizes = seq(10, 20, 2)
coverRates <- rep(0, length(sizes))
for (i in 1:length(sizes)) {
  coverRates[i] <- plotInV8(sampleSize=sizes[i])
}

plot(sizes, coverRates, type="o")
lines(sizes, rep(0.88, length(sizes)), type="l")
```



By the analysis of question 7, a good estimation of σ can promise a good cover probability of CI. For instance, we can increase the sample size in order to otain a better estimation of σ. The figure shows that as the samples size increases, we tends to obtain a better CI. When sample size equals to 12, we can obtain a CI with cover probability close to 0.88. Actually, there are several other ways to improve the cover probability of CI, such as enlarging the interval, or choosing a more reasonable model to make a good estimation on σ with small sample size, e.g., student's T-distribution in this case.
