Quiz 3

Lingjun Fu, Matthew Lenert, Lisa Lin, Ivo Violich

21 September, 2015

Question 1-2:

```r
# Play "Tenzee" with k dice -- Binomial solution
Rk <- function(k=10){
 matches <- max(c( rmultinom(n=1, size=k, prob=rep(1/6,6)) ))
 diceLeft <- k - matches
 rolls <- 1
 while(diceLeft != 0){
   matches <- rbinom(n=1, size=diceLeft, prob=1/6)
   diceLeft <- diceLeft - matches
   rolls <- rolls + 1
 }
 return(rolls)
}
# Optimize

#Simultate a game of Tenzie
simul_tenzie <- function(parrallelSims=10, n,meanOrSD){
 x_bar <- c()
 #run multiple simulations in parallel
 for (i in 1:parrallelSims){
   d0 <- c()

   #Each simulation is a sample of n games.
   #N dynamically increaes over time by an increment set in steps parameter.
   for (games in 1:n) {
     d0[games] <- Rk(k=10)
   }

   #Code can be used for both mean and SD problems
   if (meanOrSD=="SD"){
    x_bar[i] <- sd(d0,na.rm = TRUE)
   }
   else{
     x_bar[i] <- mean(d0,na.rm=TRUE)
   }
 }
 return(list(n=n, x_bar=x_bar))
}

#  This function runs a variable number of parallel samples of Tenzie If the mean or SD of each simulat
# is within the margain parameter of the mean of all simulations,
# then the stop condtion counter is incremented. The function will continue until
# the stop condition counter reaches the threshold parameter.
# This means that enough simulations are within the margain of the mean of all simultaion.
# This function seeks to find the minimum needed of simulation in both cases for effeciency sake.
# Parameters
# parrallelSims - Number of simulations to run in parrallel
# steps - Amount sample size of every simulation will increase by each loop
# stopMargain - Margain a simultaion must be from mean of all simulation to count toward the stop thres
# stopThreshold - The number of simultation needed within the margain to end the function
# meanOrSD - String input of ''mean'' or ''SD'' to indicate which problem this is being used to solve
# defaultStartPoint - Default sample size to start with for all simulations
repeat_simul <- function(parrallelSims=10,steps=10000,stopMargain=0.01,stopThreshold=8,meanOrSD="mean",
 stopCount=0
 series<-defaultStartPoint
 meanOfXBar=0

 while (stopThreshold > stopCount){

   #simulate samples of Tenzie
   x <- simul_tenzie(parrallelSims,n=series,meanOrSD = meanOrSD)

   #take mean of all simulations
   meanOfXBar=mean(x$x_bar)

   stopCount=0

   #count how many simulations are within margain from mean of all simulations
   for(iterate in 1:parrallelSims){
     if(abs(meanOfXBar-x$x_bar[iterate])<stopMargain){
       stopCount=stopCount+1
     }
   }

   #print information as it processes so you know its progress
   print(cat(series," games with ",stopCount,"simulations out of ",parrallelSims," within stop margain.
   series=series+steps
 }
 return(cat("It took a sample of ",series-steps," to achieve a stable ",meanOrSD," of ",meanOfXBar))
 }

repeat_simul(10,steps = 10000, stopMargain = 0.01, stopThreshold = 8, meanOrSD = "mean",80000)
repeat_simul(10,steps = 1, stopMargain = 0.1, stopThreshold = 9, meanOrSD = "SD",1000)
```

Here are the simulation results:

It took ten samples of 210000 to achieve a stable mean of 15.35186

It took ten samples of 1023 to achieve a stable SD of 6.619448

Set up the functions for Q3-Q8:

```r
R10 <- function(){
  matches <- max(c( rmultinom(n=1, size=10, prob=rep(1/6,6)) ))
  diceLeft <- 10 - matches
  rolls <- 1
  while(diceLeft != 0){
    matches <- rbinom(n=1, size=diceLeft, prob=1/6)
    diceLeft <- diceLeft - matches
    rolls <- rolls + 1
  }
  return(rolls)
}

# Play "Tenzee" with k dice -- Binomial solution
Rk <- function(k=10){
  matches <- max(c( rmultinom(n=1, size=k, prob=rep(1/6,6)) ))
  diceLeft <- k - matches
  rolls <- 1
  while(diceLeft != 0){
    matches <- rbinom(n=1, size=diceLeft, prob=1/6)
    diceLeft <- diceLeft - matches
    rolls <- rolls + 1
  }
  return(rolls)
}


# Take a sample range from 1 to some large number
# set.seed(7)
samples <- 100
d0 <- rep(NA, samples)
var_R10 <- rep(NA, samples)
E_R10 <- rep(NA, samples)
for(i in 1:samples){
  d0[i] <- Rk(k=10)
}
```

Question 3:

We can use the mean value of the 100 samples as an estimation of $E[R_{10}]$. In the following code, we sample 100 rounds of Tenzee each time and repeat it 1000 times. We see that the standard deviation (sd(d0)) in one experiment is quite stable and close to the stable value we got in question 2 (6.619448), while the expected mean is not stable among different experiments and has a standard error (sd(mean)). Roughly, sd(mean) is about one tenth of sd(d0). This agrees with the formula: $\sigma_M = \frac{\sigma}{\sqrt{N}}$, here $\sqrt{N} = \sqrt{100} = 10$.

```
test <- 1000   ### try 10 times, each time roll 100 rounds of Tenzee
mean <- rep(NA, test) ### record mean value of each time
for(i in 1:test){
    samples <- 100
    d0 <- rep(NA, samples)
    for(j in 1:samples){
        d0[j] <- Rk(k=10)
    }
    mean[i] <- mean(d0)
```

```r
}
sd(mean)
```

```
## [1] 0.6923198
```

```r
sd(d0)
```

```
## [1] 7.792064
```

Question 4:

We propose a 1-sigma interval estimate for $E[R_{10}]$ based on a sample of only 100 rounds of Tenzee: $E[R_{10}] \sim [\mu_M - \sigma_M, \mu_M + \sigma_M]$. In other words, there is a probability of around 68% that the real $E[R_{10}]$ will fall in this range. To find μ_M and σ_M, we use the following method:

1. Find the mean and standard deviation of the 100 rounds data, denoted by μ and σ respectively.
2. $\mu_M = \mu$.
3. $\sigma_M = \frac{\sigma}{\sqrt{N}}$, here $N = 100$ is the sample size.

Our method is based on the theory of sampling distribution of the mean (see reference at http://onlinestatbook.com/2/sampling_distributions/samp_dist_mean.html). A simple check of our method would be: assume an extreme case where we are investigating an sample of infinite size ($N = \infty$), then $\sigma_M = 0$. This is because our sample can completely represent the population, and therefore the mean of the sample is just the real population mean.

Question 5:

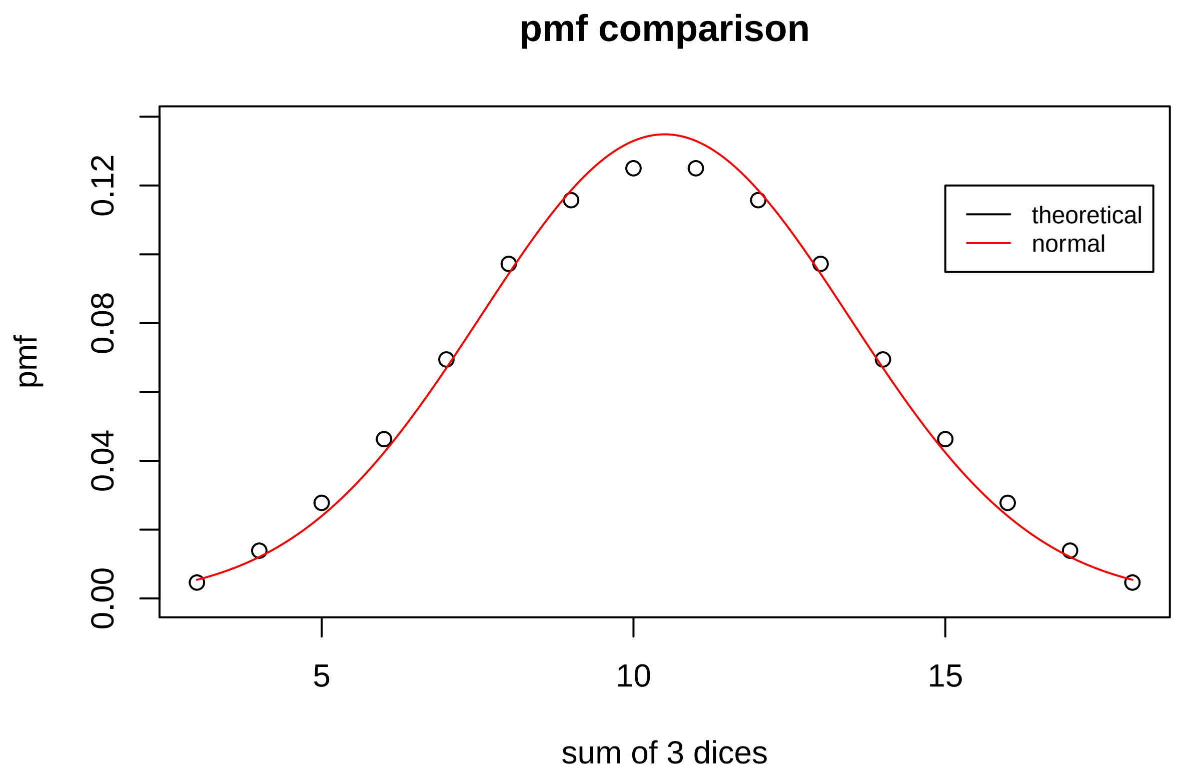
As is seen from our formula, $\mu_M = \mu$ does not change when the sample size changes, while σ_M gets smaller when the sample size gets larger. In other words, a larger sample size reduces the length of the estimated interval.

Question 6:

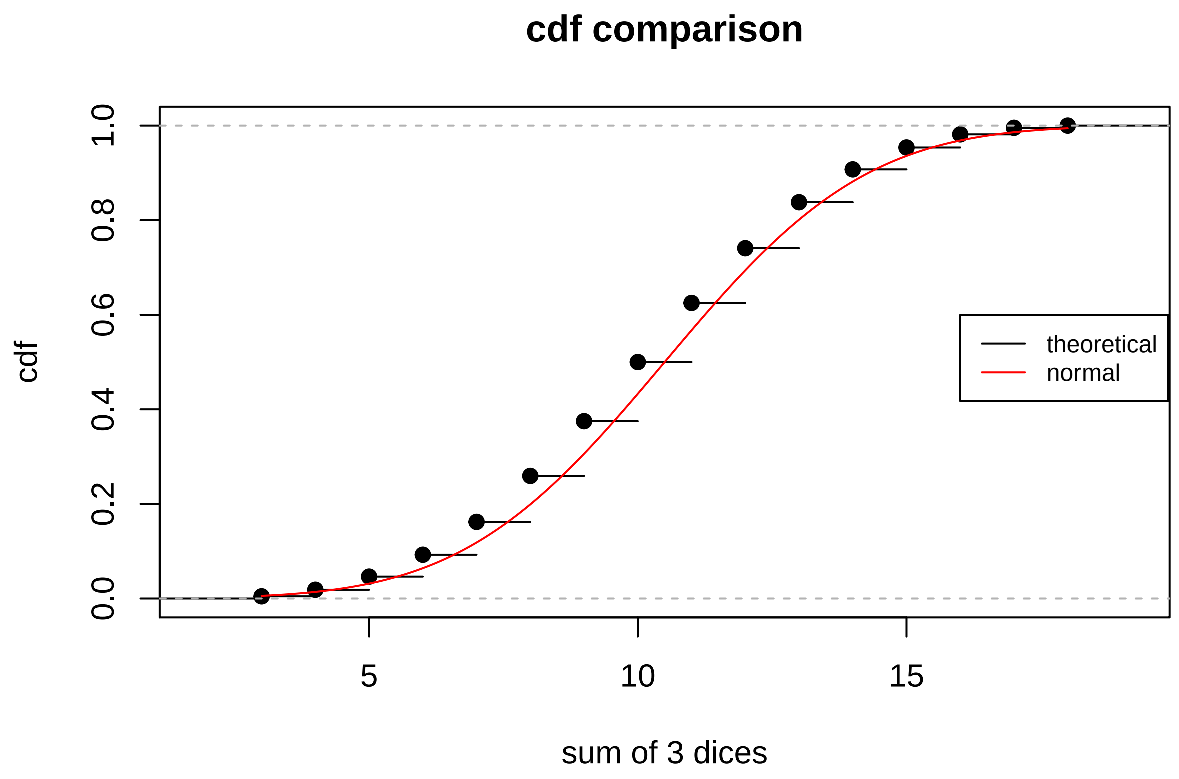
We use the "dice" package to generate theoretical pmf of the sum of 3 dices. Ideally, this is a geometry distribution. Then, we use the mean value and standard deviation of this geometry pmf as two parameters to generate a normal distribution.

```r
# Reference: https://cran.r-project.org/web/packages/dice/dice.pdf
library(gtools)  # install.packages("gtools")
library(dice)  # install.packages("dice")

###pmf for both geometry and normal distribution
```

```r
pmf_X3 <- getSumProbs(ndicePerRoll = 3, nsidesPerDie = 6, dropLowest = FALSE)$probabilities[,2]
sample_X3 <- c(3:18)
mean_X3 <- getSumProbs(ndicePerRoll = 3, nsidesPerDie = 6, dropLowest = FALSE)$average
sd_X3 <- (3*35/12)^0.5
# http://math.stackexchange.com/questions/406192/probability-distribution-of-rolling-multiple-dice
# tells us that rolling n six-sided dices will yield an approximately normal Distribution
# whose mean is n*3.5 and variance is n*35/12.
x<- seq(3,18,length=1000)
pmf_X3norm <- dnorm(x, mean=mean_X3, sd=sd_X3)
plot(sample_X3, pmf_X3, xlim=c(3,18), ylim=c(0,1.1*max(pmf_X3)), type='p',xlab="sum of 3 dices",
ylab="pmf", main="pmf comparison")
lines(x, pmf_X3norm, col="red")
legend(15,0.12,c("theoretical","normal"),lty=c(1,1),lwd=c(1,1),col=c("black","red"),cex=0.75)
```



```r
###cdf for both geometry and normal distribution
b <- c(3:18)
d <- c(1,3,6,10,15,21,25,27,27,25,21,15,10,6,3,1)
freq <- rep(b,d)
plot.ecdf(freq, xlab="sum of 3 dices", ylab="cdf", main="cdf comparison")
cdf_X3norm <- pnorm(x, mean=mean_X3, sd=sd_X3)
lines(x, cdf_X3norm, col="red")
legend(16,0.6,c("theoretical","normal"),lty=c(1,1),lwd=c(1,1),col=c("black","red"),cex=0.75)
```



As we can see from the pmf figure, a Normal distribution (denoted by curve) can approximate the distribution of X_3 (denoted by discrete points) quite well on the two sides, but deviates obviously in the middle. This deviation can be validated in the cdf figure as well. Therefore, we conclude that normal distribution is not a good approximation of the distribution of X_3.

Question 7:

Rather than plotting the pmf and cdf figures in question 6, we propose a quantitative criteria here. Our criteria is that the distribution of X_k should behave as follows:
1. No obvious deviation from the normal distribution curve.
2. About 68.3% of possible values fall within 1 standard deviation.
3. About 95.5% of possible values fall within 2 standard deviation.
4. About 99.7% of possible values fall within 3 standard deviation.

The central limit theorem implicates that the above criteria will be satisfied better when we increase k. We start from k=3, and then increase k until the above criteria is satisfied to a sufficient resolution (0.005). To save some computation time, we set the criteria as: the difference of the ratio of data located in 1, 2, and 3 sigma ranges between normal and theoretical distribution.

```
# Reference: https://cran.r-project.org/web/packages/dice/dice.pdf
library(gtools)
library(dice)
k <- 3
while(1){
```

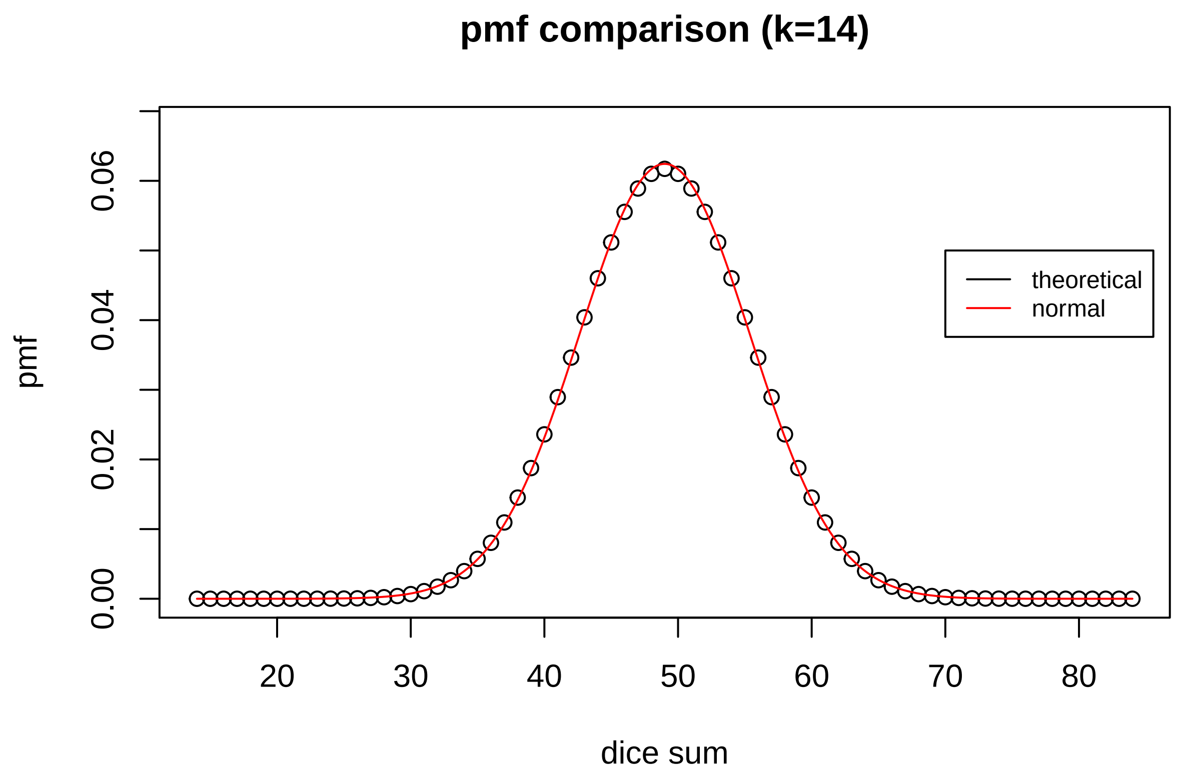
```r
pmf_Xk <- getSumProbs(ndicePerRoll = k, nsidesPerDie = 6, dropLowest = FALSE)$probabilities[,2]
sample_Xk <- c(k:(6*k))
mean_Xk <- getSumProbs(ndicePerRoll = k, nsidesPerDie = 6, dropLowest = FALSE)$average
sd_Xk <- (k*35/12)^0.5
# http://math.stackexchange.com/questions/406192/probability-distribution-of-rolling-multiple-dice

# Reference tells us that rolling n six-sided dices will yield an approximately

# Normal Distribution whose mean is n*3.5 and variance is n*35/12.

x<- seq(k,(6*k),length=1000)
pmf_Xknorm <- dnorm(x, mean=mean_Xk, sd=sd_Xk)

P_sigma <- rep(0, 3)
for(j in 1:3){
    for(i in seq_along(sample_Xk)){
    if((sample_Xk[i] + j*sd_Xk >= mean_Xk) & (sample_Xk[i] <= mean_Xk + j*sd_Xk)){
        P_sigma[j] = P_sigma[j] + pmf_Xk[i]
    }
}
}
if((abs(P_sigma[1] - 0.683) < 0.005)
   & (abs(P_sigma[2] - 0.955) < 0.005)
   & (abs(P_sigma[3] - 0.997) < 0.005)){
    break
}
k  <- k + 1
}
P_sigma  #To compare with 0.683, 0.955, 0.997
```

```
## [1] 0.6877476 0.9505452 0.9983110
```

```r
k
```

```
## [1] 14
```

```r
plot(sample_Xk, pmf_Xk, xlim=c(k,(6*k)), ylim=c(0,(1.1*max(pmf_Xk))),
     type='p',xlab="dice sum", ylab="pmf", main="pmf comparison (k=14)")
lines(x, pmf_Xknorm, col="red")
legend(70,0.05,c("theoretical","normal"),lty=c(1,1),lwd=c(1,1),col=c("black","red"),cex=0.75)
```

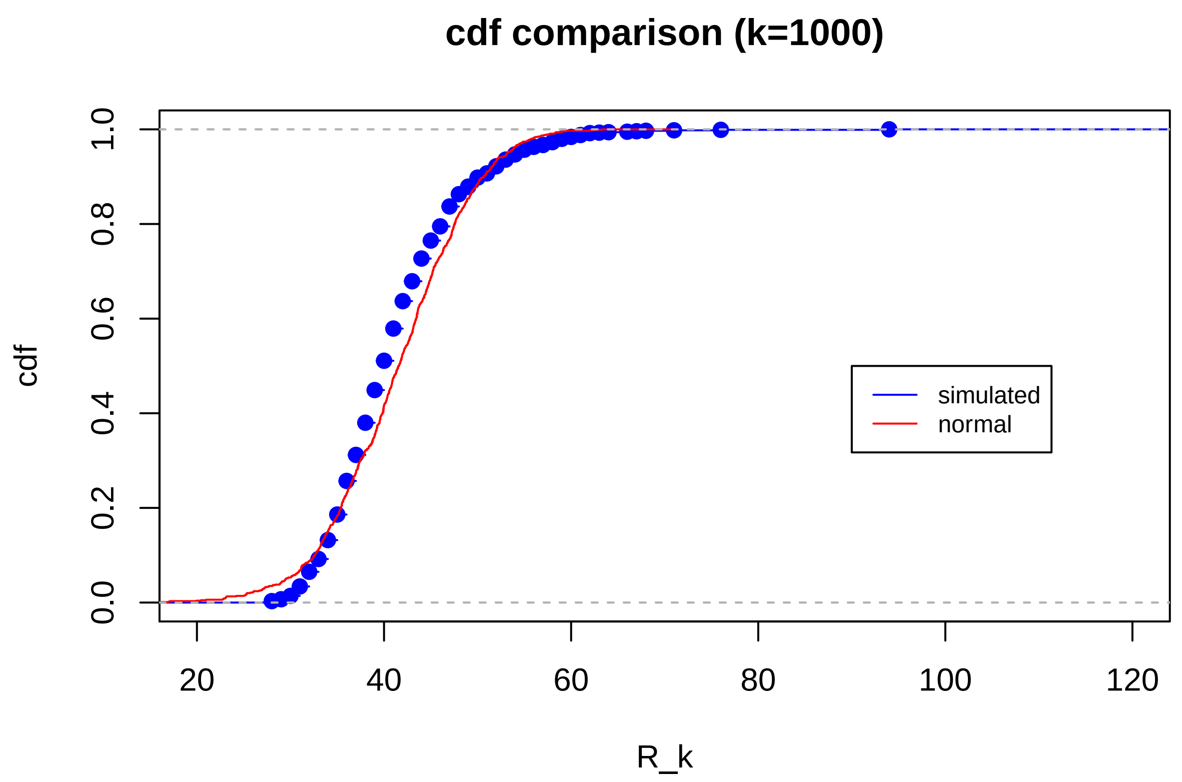


When we run the code on our laptop, we make a plot for each k and clearly see that the approximation become better in the dynamic process. Eventually, our code stops at k=14 where our criteria is satisfied for the first time.

Question 8:

We suspect that it is impossible to find a k such that a Normal distribution approximates the distribution of R_k satisfactorily. In general, the central limit theorem is applicable to a sufficiently large number of iterates of independent random variables. In this case, it is the number of rounds of rolling Tenzee, rather than the total number of dices k. Therefore, we cannot achieve a good normal distribution approximation of the pmf of R_k simply by increasing k.

```r
samples <- 1000
d0 <- rep(NA, samples)
var_R10 <- rep(NA, samples)
E_R10 <- rep(NA, samples)
for(i in 1:samples){
  d0[i] <- Rk(k=1000)
}
x<- seq(20,100,length=1000)
d0_norm <- rnorm(samples, mean=mean(d0), sd=sd(d0))
plot(ecdf(d0),xlim=c(20,120),ylim=c(0,1),,xlab="R_k",ylab="cdf", main="cdf comparison (k=1000)", col="b
lines(ecdf(d0_norm),col="red")
legend(90,0.5,c("simulated","normal"),lty=c(1,1),lwd=c(1,1),col=c("blue","red"), cex=0.75)
```

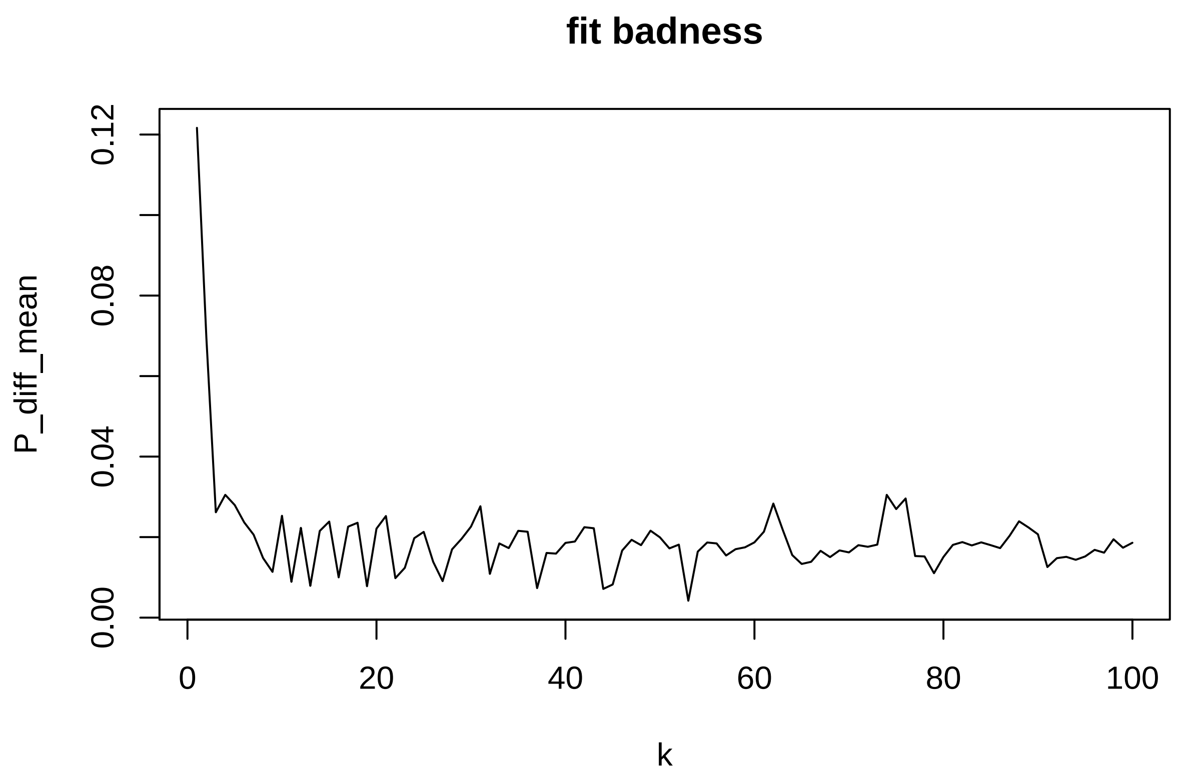


The above figure clearly tells that the cdf of simulated and normal distribution do not match well even we increase k to 1000. To see it more clearly, we use the same criteria in question 7 as a double check.

```
k <- 1
k_max <- 100
P_diff_mean <- rep(NA, k_max)
while(k <= k_max){
    samples <- 10000
    d0 <- rep(NA, samples)
    for(i in 1:samples){
        d0[i] <- Rk(k)
    }
    sd_d0 <- sd(d0)
    mean_d0 <- mean(d0)

    freq_sigma <- rep(0, 3)
    for(j in 1:3){
        for(i in seq_along(d0)){
            if((d0[i] + j*sd_d0 >= mean_d0) & (d0[i] <= mean_d0 + j*sd_d0)){
             freq_sigma[j] = freq_sigma[j] + 1
             }
        }
    }
    P_sigma <- freq_sigma/samples
    P_diff_mean[k] = (abs(P_sigma[1] - 0.683) + abs(P_sigma[2] - 0.955) + abs(P_sigma[3] - 0.997))/3
    # P_diff_mean is a variable to measure the fit badness
    if((abs(P_sigma[1] - 0.683) < 0.005)
       & (abs(P_sigma[2] - 0.955) < 0.005)
       & (abs(P_sigma[3] - 0.997) < 0.005)){
    break
}
    k <- k + 1
}
k
```

```
## [1] 101
```

```
plot(1:k_max, P_diff_mean, type="l", xlab="k", main="fit badness")
```



Again, we see that our criteria is not satisfied for k up to 100 (we did try k up to 1000 and it failed as well). In general, one can plot a figure of the fit badness as k increases. We define P_diff_mean as a variable to measure the deviation of the simulated distribution from normal. As we can see from the above picture, the value of P_diff_mean does not decrease monotonically as k increases. This supports our previous argument: we cannot achieve a good normal distribution approximation of R_k simply by increasing k.