ASSIGNMENT 3

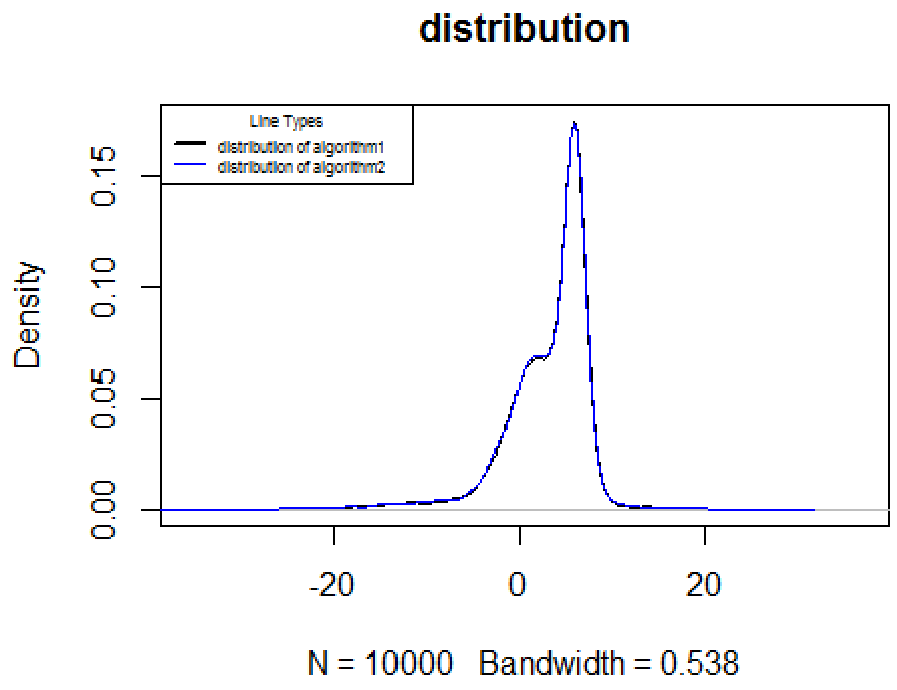
Yibing Luo

912491862

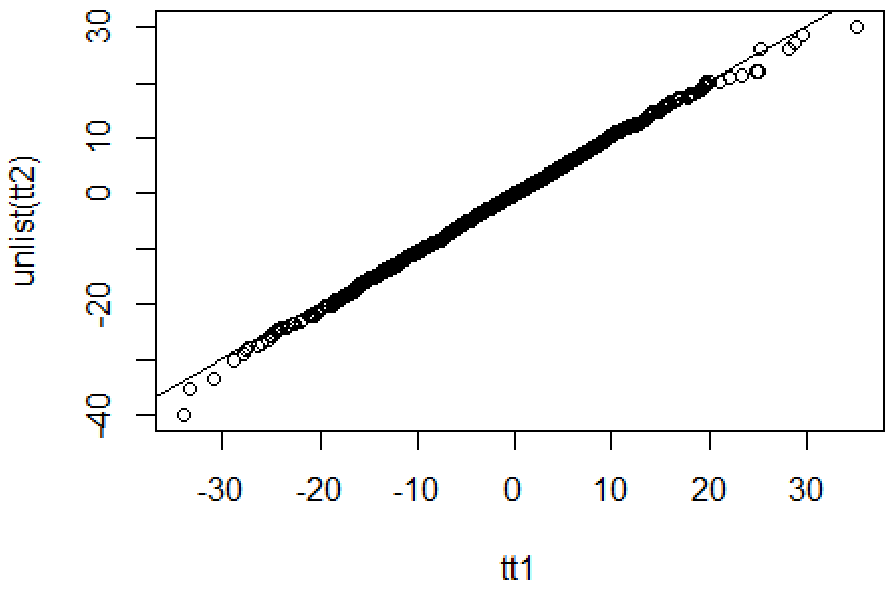
The first question

```
#the first algorithm to generate sample
#inputs are the numbers of sample, probilities of events,
#and mean and standard of normal distributions, function
#return is numSample random numbers

mixture1 = function (numSample , prob , meanSd){
  #use sample to choose distribution from k norems
  knorm=sample(c(1:length(prob)) , size = numSample , replace = TRUE ,
prob = prob)

  #use sapply to get the distrbution
  sapply(1:numSample , function(p) rnorm( 1, meanSd[1 , knorm[p]] , mea
nSd[2 , knorm[p]]))
}

#the second algorithm to generate sample
#inputs are the numbers of sample, probilities of events,
#and mean and standard of normal distributions, function
#return is numSample random numbers

select = function( n = 10000, probs )
{
  CDF = c(0, cumsum( probs ))
  u = runif(n)

  g = cut(u, CDF)
  names(probs)[g]
}


mixture2 = function (numSample , probs , meanSd){
  gg=select(numSample, probs)


  #Let all events show up in gg, so we can use sapply to manipulate
  gg = c(gg , names(probs))
  num = table(gg)

  sapply(1:length(probs), function(p) rnorm(as.vector(num[p]-1) , meanS
d[1,p] , meanSd[2,p]))
}
```

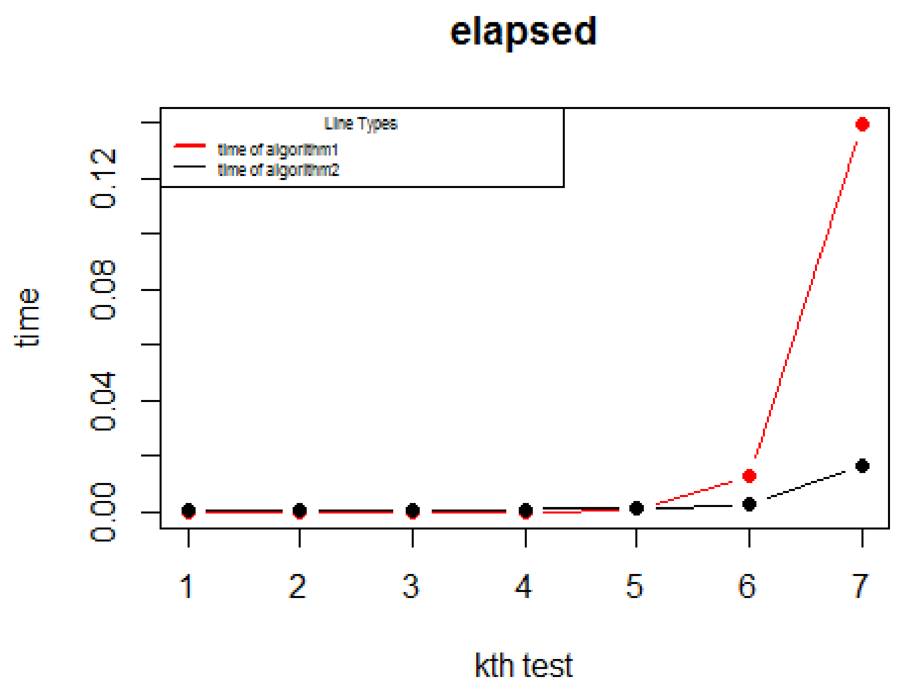
```r
#give the k normal distributions
meanSd = matrix( c(-3,10,6,1,2,3), nrow = 2, ncol = 3, byrow = F)

#obtain the sample
tt1 = mixture1(10000 , c(a=.1,b=.4,c=.5) , meanSd)
tt2 = mixture2(10000 , c(a=.1,b=.4,c=.5) , meanSd)

#plot the distribution we obtain to figure out whether the two results
are the same
plot( density(tt1) , main = "distribution" )
lines( density(unlist(tt2)) , col = "blue" )
legend( "topleft" , legend = c("distribution of algorithm1","distributi
on of algorithm2"),
       col = c("black","blue") , cex = .5, lty = c(1,1) , lwd = c(2,.7)
 , title = "Line Types",
       xjust = 0 , yjust = 0,text.width=20)
```



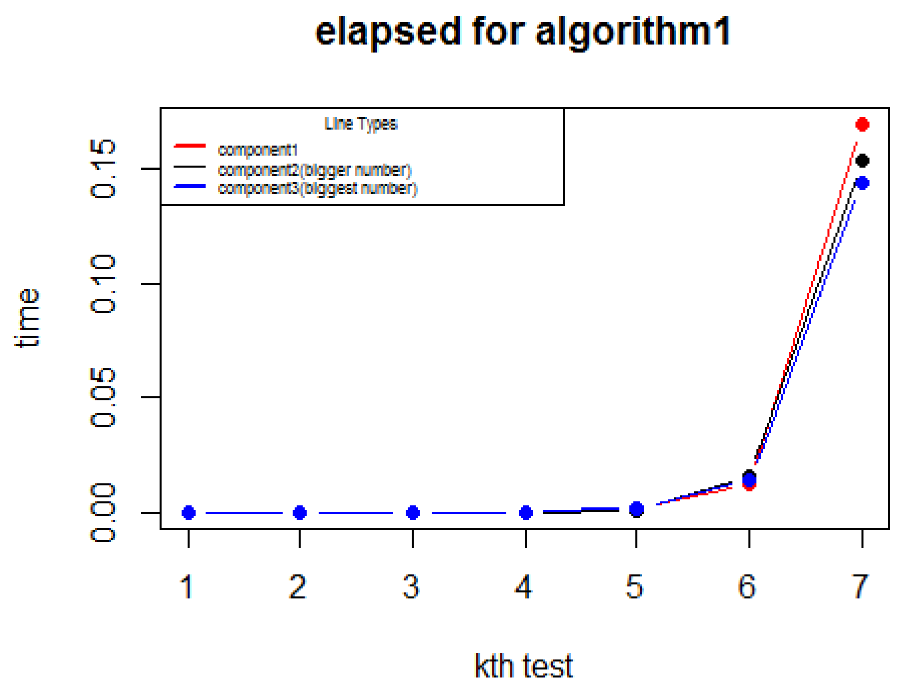
```r
#use qqplot to identify
qqplot(tt1,unlist(tt2))
abline(a = 0,b = 1)
```



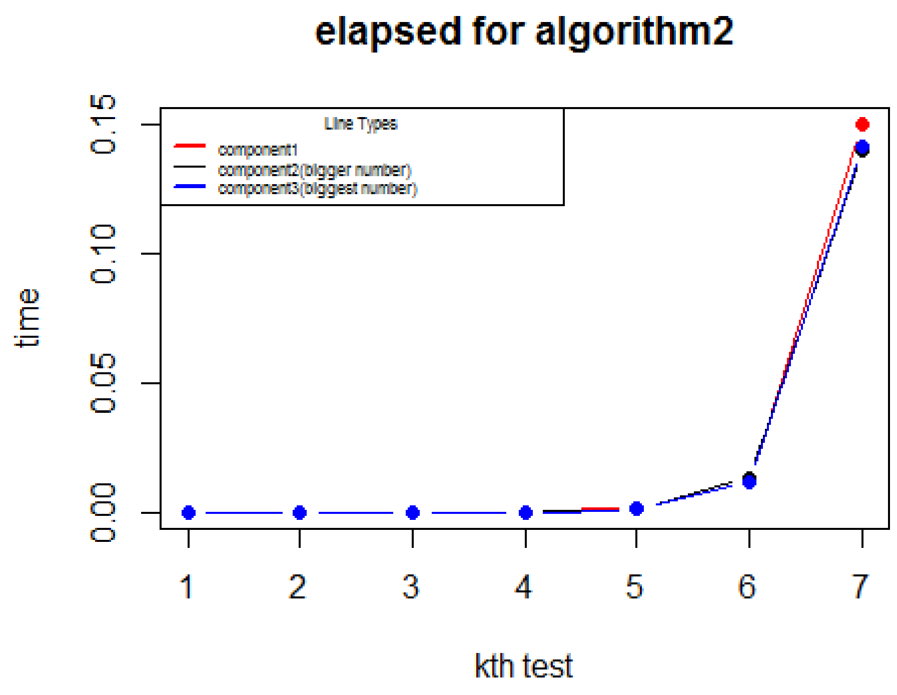
```
#plot the time to compare algorithm1 and algorithm2
x=c(1, 2, 5, 10, 100, 1000, 10000)

time_for_1=sapply(1:length(x), function(p) system.time(replicate(100,mi
xture1(x[p],c(a=.1,b=.4,c=.5) , meanSd)))/100)
time_for_2=sapply(1:length(x), function(p) system.time(replicate(100,mi
xture2(x[p],c(a=.1,b=.4,c=.5) , meanSd)))/100)

plot( time_for_1[3,] , main=("elapsed"), xlab="kth test" , ylab="time" ,
 col='red', pch = 19,type="b")
points(time_for_2[3,] , pch = 19,type="b")

legend( "topleft" , legend = c("time of algorithm1","time of algorithm2
"), col = c("red","black") , cex = .5, lty = c(1,1) , lwd = c(2,.7) , t
itle = "Line Types", xjust = 0 , yjust = 0,text.width=3)
```



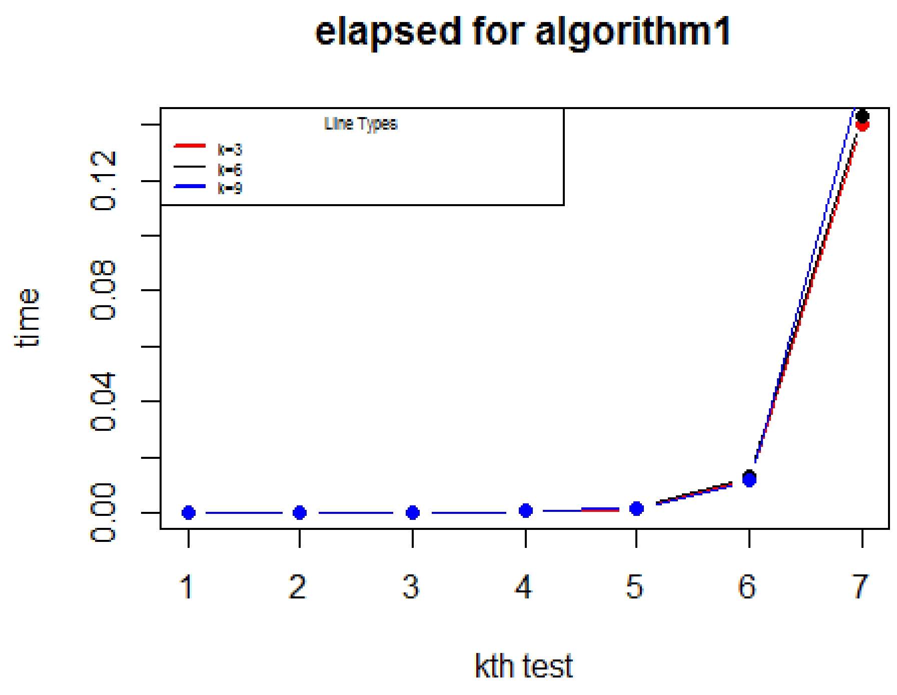
```
##########################################################
#consider about the parameters in the component densities
meanSd1 = matrix( c(-3,10,6,1,2,3), nrow = 2, ncol = 3, byrow = F)
meanSd2 = matrix( c(-9,100,36,1,4,9), nrow = 2, ncol = 3, byrow = F)
meanSd3 = matrix( c(-9^9,10^9,6^9,1,4^9,9^9), nrow = 2, ncol = 3, byrow
 = F)
time_for_1 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd1)))/100)
time_for_2 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd2)))/100)
time_for_3 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd3)))/100)

plot( time_for_1[3,] , main = ("elapsed for algorithm1"), xlab = "kth t
est" , ylab = "time" , col = "red", pch = 19, type = "b")
points(time_for_2[3,] , pch = 19,type = "b")
points(time_for_3[3,] , pch = 19,type = "b" ,col = "blue")

legend( "topleft" , legend = c("component1","component2(bigger number)",
"component3(biggest number)"), col = c("red","black","blue") , cex = .5,
 lty = c(1,1) , lwd = c(2,.7) , title = "Line Types", xjust = 0 , yjust
 = 0,text.width=3)
```



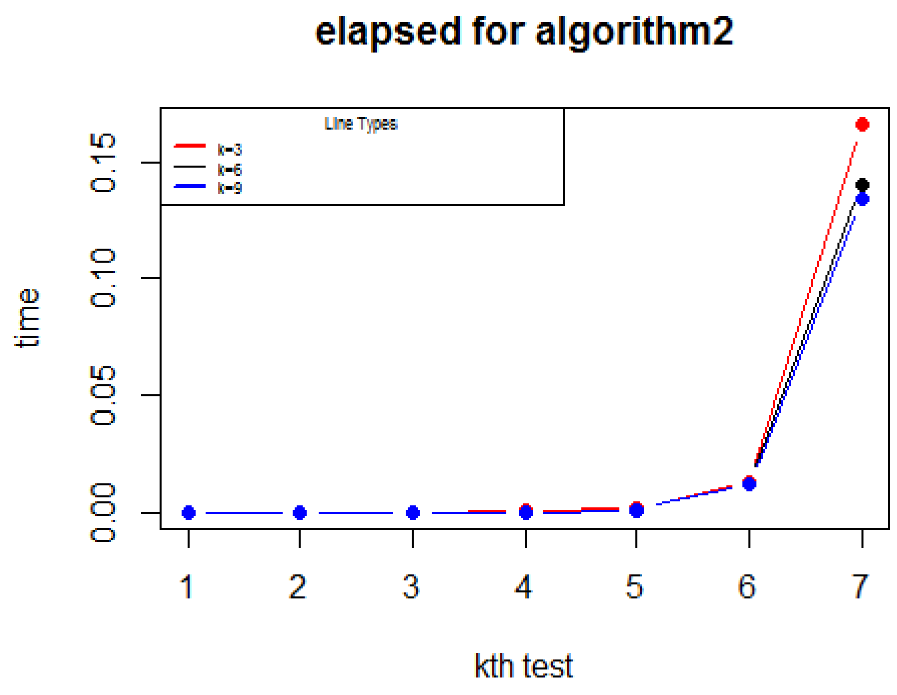
```
#the sencond algorith
meanSd1 = matrix( c(-3,10,6,1,2,3), nrow = 2, ncol = 3, byrow = F)
meanSd2 = matrix( c(-9,100,36,1,4,9), nrow = 2, ncol = 3, byrow = F)
meanSd3 = matrix( c(-9^9,10^9,6^9,1,4^9,9^9), nrow = 2, ncol = 3, byrow
 = F)
time_for_1 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd1))) / 100)
time_for_2 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd2))) / 100)
time_for_3 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd3))) / 100)

plot( time_for_1[3,] , main = ("elapsed for algorithm2"), xlab = "kth t
est" , ylab = "time" , col = 'red', pch = 19,type = "b")
points(time_for_2[3,] , pch = 19, type = "b")
points(time_for_3[3,] , pch = 19,type = "b" ,col = "blue")

legend( "topleft" , legend = c("component1","component2(bigger number)",
"component3(biggest number)"),col = c("red","black","blue") , cex = .5,
 lty = c(1,1) , lwd = c(2,.7) , title = "Line Types", xjust = 0 , yjust
 = 0,text.width=3)
```



```
#########################################################
#consider the number of components in the mixture
meanSd1 = matrix( c(-3,10,6,1,2,3), nrow = 2, ncol = 3, byrow = F)
meanSd2 = matrix( c(-3,10,6,5,7,9,-1,3,9,1,2,3), nrow = 2, ncol = 6, by
row = F)
meanSd3 = matrix( c(-3,10,6,5,7,9,20,11,2,7,13,10,-1,3,9,1,2,3), nrow =
 2, ncol = 9, byrow = F)

time_for_1 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd1)))/100)
time_for_2 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd2)))/100)
time_for_3 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd3)))/100)

plot( time_for_1[3,] , main = ("elapsed for algorithm1"), xlab = "kth t
est" , ylab = "time" , col = 'red', pch = 19,type = "b")
points(time_for_2[3,] , pch = 19,type="b")
points(time_for_3[3,] , pch = 19,type="b" ,col="blue")

legend( "topleft" , legend = c("k=3","k=6","k=9"),
        col = c("red","black","blue") , cex = .5, lty = c(1,1) , lwd =
c(2,.7) , title = "Line Types",
        xjust = 0 , yjust = 0,text.width=3)
```



```
#the second algorithm
meanSd1 = matrix( c(-3,10,6,1,2,3), nrow = 2, ncol = 3, byrow = F)
meanSd2 = matrix( c(-3,10,6,5,7,9,-1,3,9,1,2,3), nrow = 2, ncol = 6, by
row = F)
meanSd3 = matrix( c(-3,10,6,5,7,9,20,11,2,7,13,10,-1,3,9,1,2,3), nrow =
 2, ncol = 9, byrow = F)

time_for_1 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd1)))/100)
time_for_2 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd2)))/100)
time_for_3 = sapply(1:length(x), function(p) system.time(replicate(100,
mixture1(x[p],c(a =.1,b =.4,c =.5) , meanSd3)))/100)

plot( time_for_1[3,] , main = ("elapsed for algorithm2"), xlab = "kth t
est" , ylab = "time" , col = 'red', pch = 19,type = "b")
points(time_for_2[3,] , pch = 19,type = "b")
points(time_for_3[3,] , pch = 19,type = "b" ,col = "blue")

legend( "topleft" , legend = c("k=3","k=6","k=9"),col = c("red","black",
"blue") , cex = .5, lty = c(1,1) , lwd = c(2,.7) , title = "Line Types",
 xjust = 0 , yjust = 0,text.width=3)
```



Report:

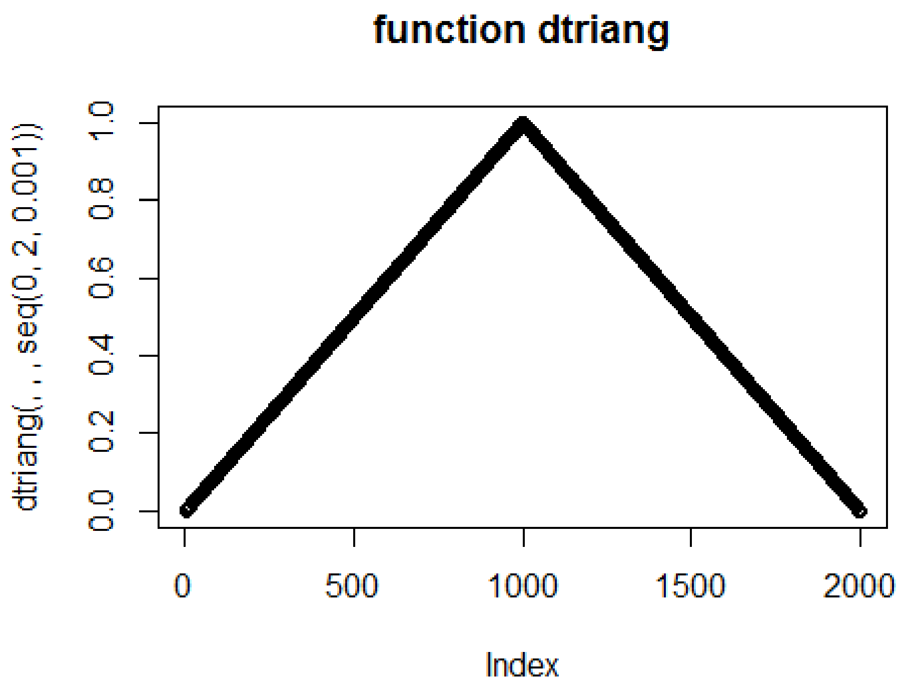
The algorithm 2 is significantly fast compared with algorithm 1 by drawing their plots especially when the k is big.

The time of each function using is depend on sample size but I don't think it depend either parameters in the component densities or number of k.

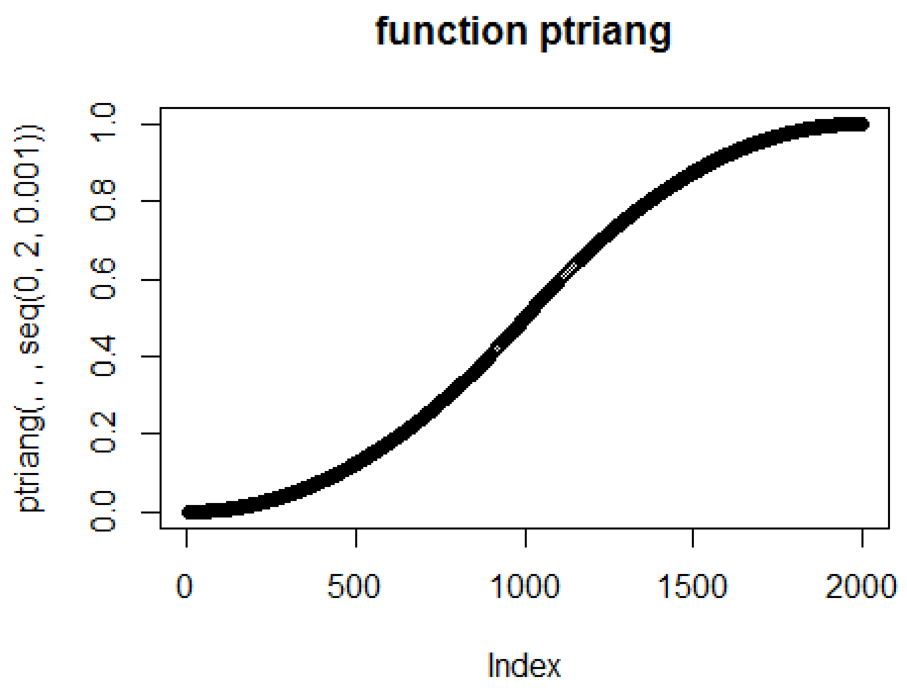
the second question the first part

```
#All these three functions are based on what we derive before.

#function dtriang()

dtriang = function(a = 0 , b = 2 , c=1 , X){
  fun = c( 1:length(X))
  for( i in 1:length(X) ){
    if (X[i] >= a & X[i] < c)
      fun[i] = 2*(X[i] - a) / ( (b - a) * (c - a) )
    else if(X[i] >= c & X[i] <= b)
      fun[i] = 2*(b - X[i]) / ( (b - a) * (b - c) )
    else
      fun[i] = 0
  }
  fun
}
```

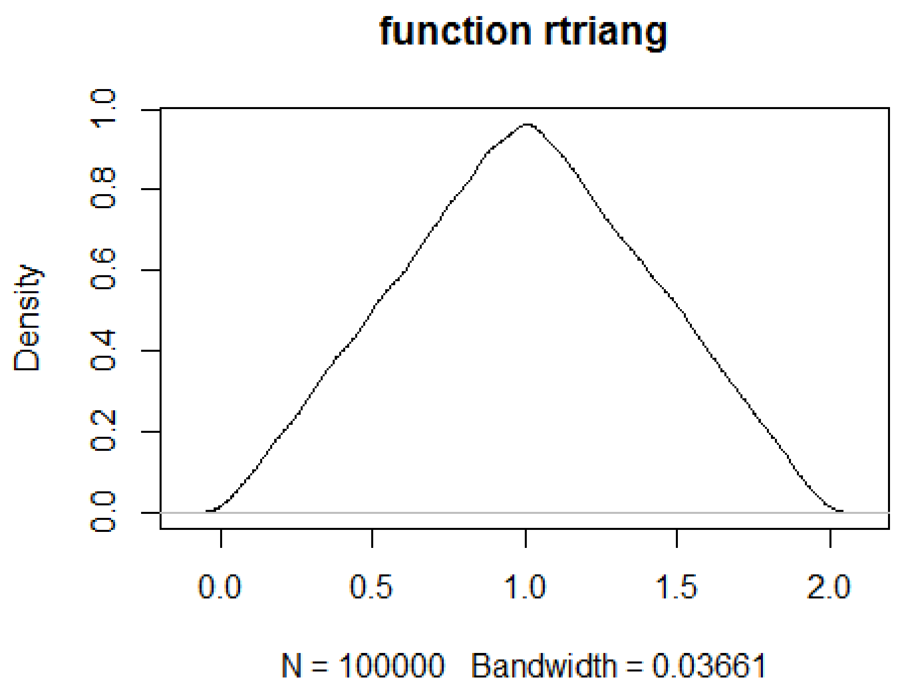
```r
plot(dtriang(,,,seq(0,2,0.001)), main = ("function dtriang"))
```



```r
#function ptriang()


ptriang = function(a = 0 ,b = 2 ,c = 1 , X){
  Fun=c(1:length(X))
  for( i in 1:length(X) ){
    if( X[i] <= a)
      Fun[i] = 0
    else if (X[i] >= a & X[i] < c)
      Fun[i] = (X[i] - a)*dtriang(a = a, b = b, c = c, X = X[i]) / 2
    else if(X[i] >= c & X[i] <= b)
      Fun[i] = (b - a )*dtriang(a = a, b = b, c =c , X = c) / 2 - (b -
X[i]) *
        dtriang(a = a, b = b, c = c, X = X[i]) / 2
    else
      Fun[i] = 1
  }
  Fun
}

plot(ptriang(,,,seq(0,2,0.001)), main = ("function ptriang"))
```



```
#function rtriang()

#function inverse CDF
inCDF = function (a = 0 , b = 2 , c = 1, X){
  inF = c(1:length(X))
  for( i in 1:length(X) ) {
    if( X[i] < 0)  return(sprintf( "x<%f" , a ))
    else if(X[i] >= ptriang(a = a, b = b, c = c, X = a) & X[i] < ptrian
g(a = a, b = b, c = c, X = c))
      inF[i] = a + sqrt( (b - a) * (c - a) * X[i])
    else if(X[i] >= ptriang(a = a, b = b, c = c, X = c) & X[i] <= ptria
ng(a = a, b = b, c = c, X = b))
      inF[i] = b - sqrt( (b - a) * (b - c) * (1 - X[i]))
    else if(X[i] > 1) return(sprintf( "x>%f" , b ))
  }
  inF
}

#use inCDF to samply from a Triangular distribution
rtriang = function (n = 1, a = 0, b = 2,c = 1){
  x = runif(n,0,1)
  inCDF(a = a, b = b, c = c, X = x)
}

plot(density(rtriang(100000)), main = ("function rtriang"))
```



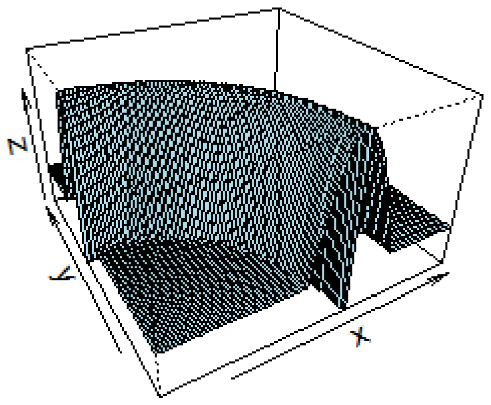
Report:

I use the plot as evidence and I think it's relatively good to support that the function s I derive are right.

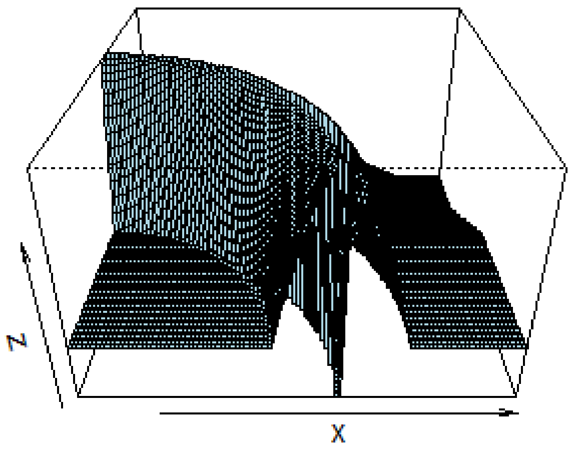
The formula of CDF and inverse CDF I write down on paper is handed in with the co py of my code.

the second question part 2

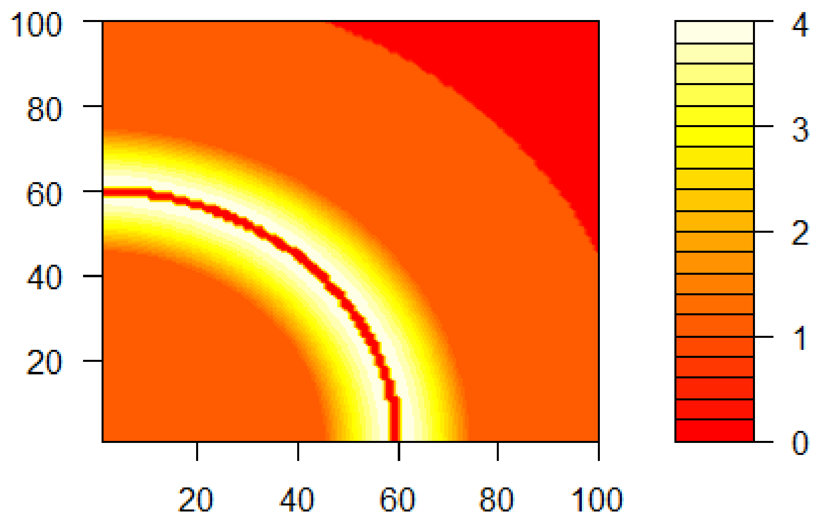
```
source(url("http://eeyore.ucdavis.edu/stat141/homework/nodeDensity.R"))

#get the plot of nodeDensity
x <- y <-seq(1,100,1)
z = outer(x,y,nodeDensity)


persp(x,y,z,theta=-30,phi=30,expand=0.6,col="lightblue")
```



```
persp(x,y,z,theta=0,phi=30,expand=0.6,col="lightblue")
```



```
filled.contour(x, y, z,color.palette=heat.colors)
```

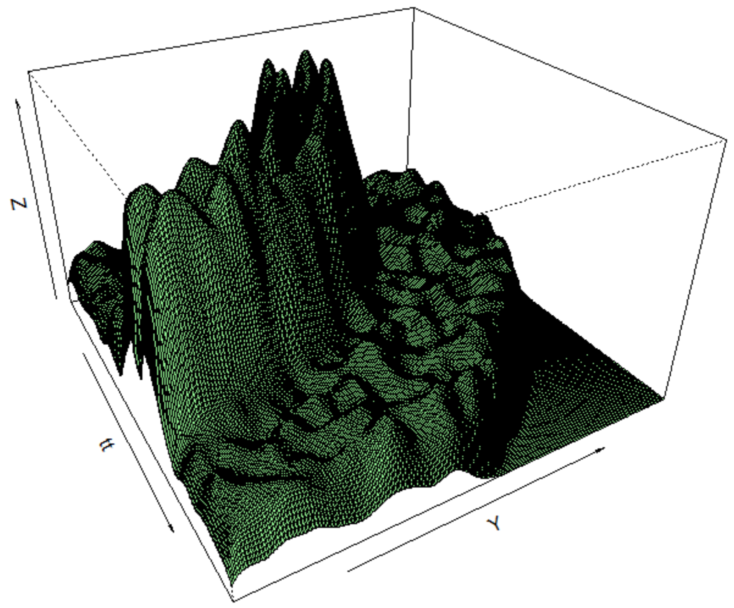


```
#two way to find the max number in the density
##############
#the first one
nodeDensity
```

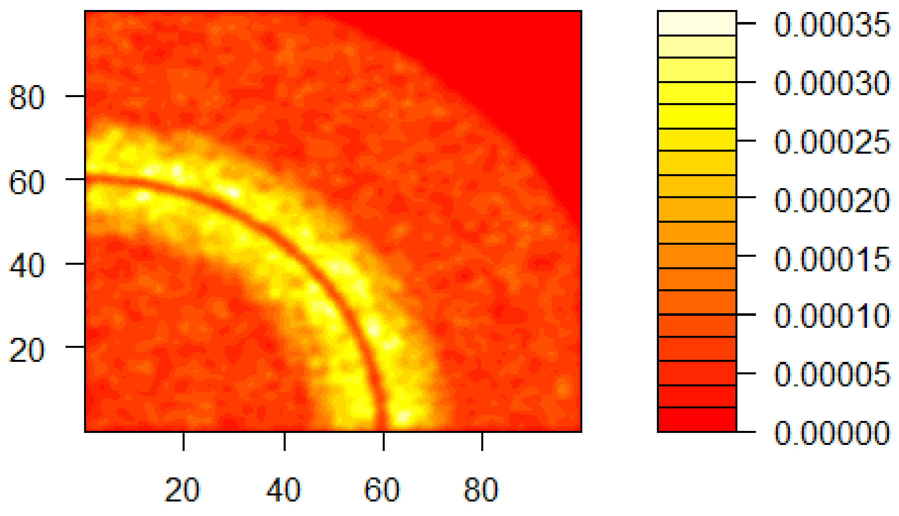
```
## function (x, y, a = 3, b = 1)
## {
##     if (mode(x) != "numeric" | mode(y) != "numeric")
##         stop("x and y must be numeric")
##     if (length(x) != length(y))
##         stop("x and y must be same length")
##     band = 15
##     bank = 1
##     inBoundary = (0 <= x & x <= 100) & (0 <= y & y <= 100 & y <
##         sqrt(110^2 - x^2))
##     river = abs(sqrt(x^2 + y^2) - 60)
##     hiArea = river > bank & river < band & inBoundary
##     hiDensity = a * cos(river[hiArea] * pi/(2 * band)) + b
##     z = b * inBoundary
##     z[hiArea] = hiDensity
##     z[river <= bank] = 0
##     z
## }
```

```
#then we will find hiDensity = a * cos(river[hiArea] * pi / (2 * band))
 + b,
#and obviously it's at most 4
```

```
mx=4
###############
#the second way
nDensity = function(x,y) {
  z = - nodeDensity(x,y)
  return(z)
  }

nlm(nDensity , p=50 , y =40)

## $minimum
## [1] -3.983563
##
## $estimate
## [1] 46.05441
##
## $gradient
## [1] 0.02479501
##
## $code
## [1] 2
##
## $iterations
## [1] 24

mx=round(-nlm(nDensity , p=50 , y =40)$minimum)     #take the value a b
it more than what we get

#sampleAR is the function which uses Acceptance/Reject to sample
sampleAR = function( fun , n ){

  samples = matrix(c(1:(2*n)) , nrow=2)                        #initial
ize parameters
  acceptnum = 0
  totalnum = 0

  while(acceptnum < n){
    totalnum = totalnum + 1

    x = runif(1 ,0 ,100 )
    y = runif(1 ,0 ,100 )
    u = runif(1 ,0 ,mx )
    if( u <= fun(x,y)){
      acceptnum = acceptnum + 1
      samples[1 , acceptnum] = x
      samples[2 , acceptnum] = y
    }
  }

    return(list(totalnum , samples) )
}


ff = sampleAR(nodeDensity, 30000)

#efficiency
eff = dim(ff[[2]])[2] / ff[[1]]
eff

## [1] 0.3376059

#plot for samples
library(MASS)

##
## Attaching package: 'MASS'
##
## The following object is masked _by_ '.GlobalEnv':
##
##     select

tt=kde2d(ff[[2]][1,],ff[[2]][2,],n=300,h=8)
persp(tt,theta=60,phi=30,expand=0.6,col="lightgreen")
```



```
tt=kde2d(ff[[2]][1,],ff[[2]][2,],n=300,h=4)
filled.contour(tt,,color.palette=heat.colors)
```



Report:

From the plots we can find the sample is pretty fine. But there is still weakness, such as the density in river is not zero. I really don't know how to correct it.

The efficiency is 33.79%, which is very low in my opinion. I try to use triangle distribution, but fail in generating 3 dimension triangle p.d.f. I am trying to find other way to make it better.