

# lab block2

Group A20

## Statement Of Contribution:

Assignment 1: YAN Jin

Assignment 2: NI Dongwei

## PART ONE

### QUESTION ONE

The code is given in the appendix.

Results are given below.

```
# QUESTION ONE PART ONE
mis_rate1 <- c()
for(i in 1:1000){
  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabels<-as.factor(y)

  train_data <- as.data.frame(trdata)
  train_data <- dplyr::mutate(train_data,trlabels)
  Forest_1 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 1, nodesize = 25, keep.f
  pre <- predict(Forest_1,newdata = train_data)
  misclassification_rate <- mean(pre!=trlabels)
  mis_rate1 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate1 <- mean(mis_rate1)
var_mis_rate1 <- var(mis_rate1)

# the mean and variable for misclassification rate when B = 1
mean_mis_rate1

## [1] 0.15419
var_mis_rate1

## [1] 0.003783527
# QUESTION ONE PART TWO
mis_rate10 <- c()
for(i in 1:1000){
```

```

x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(x1<x2)
trlabels<-as.factor(y)

train_data <- as.data.frame(trdata)
train_data <- dplyr::mutate(train_data,trlabels)
Forest_10 <- randomForest::randomForest(trlabel~.,data = train_data, ntree = 10, nodesize = 25, keep
pre <- predict(Forest_1,newdata = train_data)
misclassification_rate <- mean(pre!=trlabels)
mis_rate10 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate10 <- mean(mis_rate10)
var_mis_rate10 <- var(mis_rate10)

# the mean and variable for misclassification rate when B = 10
mean_mis_rate10

## [1] 0.1542458
var_mis_rate10

## [1] 0.003782856
# QUESTION ONE PART THREE
mis_rate100 <- c()
for(i in 1:1000){

  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<x2)
  trlabel~as.factor(y)

  train_data <- as.data.frame(trdata)
  train_data <- dplyr::mutate(train_data,trlabels)
  Forest_100 <- randomForest::randomForest(trlabel~.,data = train_data, ntree = 100, nodesize = 25, ke
  pre <- predict(Forest_1,newdata = train_data)
  misclassification_rate <- mean(pre!=trlabels)
  mis_rate100 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate100 <- mean(mis_rate100)
var_mis_rate100 <- var(mis_rate100)

# the mean and variable for misclassification rate when B = 10
mean_mis_rate100

## [1] 0.1542657

```

```
var_mis_rate100
```

```
## [1] 0.003785485
```

## QUESTION TWO

The code is given in the appendix.

Results are given below.

```
# QUESTION TWO PART ONE
```

```
mis_rate1 <- c()
```

```
for(i in 1:1000){
```

```
  x1<-runif(100)
```

```
  x2<-runif(100)
```

```
  trdata<-cbind(x1,x2)
```

```
  y<-as.numeric(x1<0.5)
```

```
  trlabels<-as.factor(y)
```

```
  train_data <- as.data.frame(trdata)
```

```
  train_data <- dplyr::mutate(train_data,trlabels)
```

```
  Forest_1 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 1, nodesize = 25, keep.f
```

```
  pre <- predict(Forest_1,newdata = train_data)
```

```
  misclassification_rate <- mean(pre!=trlabels)
```

```
  mis_rate1 <- c(mis_rate1,misclassification_rate)
```

```
}
```

```
mean_mis_rate1 <- mean(mis_rate1)
```

```
var_mis_rate1 <- var(mis_rate1)
```

```
# the mean and variable for misclassification rate when B = 1
```

```
mean_mis_rate1
```

```
## [1] 0.07121
```

```
var_mis_rate1
```

```
## [1] 0.01237381
```

```
# QUESTION two PART TWO
```

```
mis_rate10 <- c()
```

```
for(i in 1:1000){
```

```
  x1<-runif(100)
```

```
  x2<-runif(100)
```

```
  trdata<-cbind(x1,x2)
```

```
  y<-as.numeric(x1<0.5)
```

```
  trlabels<-as.factor(y)
```

```
  train_data <- as.data.frame(trdata)
```

```
  train_data <- dplyr::mutate(train_data,trlabels)
```

```
  Forest_10 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 10, nodesize = 25, keep
```

```

pre <- predict(Forest_1,newdata = train_data)
misclassification_rate <- mean(pre!=trlabels)
mis_rate10 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate10 <- mean(mis_rate10)
var_mis_rate10 <- var(mis_rate10)
# the mean and variable for misclassification rate when B = 10
mean_mis_rate10

## [1] 0.07155844
var_mis_rate10

## [1] 0.01248297
# QUESTION TWO PART THREE
mis_rate100 <- c()
for(i in 1:1000){

  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(x1<0.5)
  trlabels<-as.factor(y)

  train_data <- as.data.frame(trdata)
  train_data <- dplyr::mutate(train_data,trlabels)
  Forest_100 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 100, nodesize = 25, ke
  pre <- predict(Forest_1,newdata = train_data)
  misclassification_rate <- mean(pre!=trlabels)
  mis_rate100 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate100 <- mean(mis_rate100)
var_mis_rate100 <- var(mis_rate100)

# the mean and variable for misclassification rate when B = 10
mean_mis_rate100

## [1] 0.07137862
var_mis_rate100

## [1] 0.0123899

```

## QUESTION THREE

The code is given in the appendix.

Results are given below.

```

# QUESTION THREE PART ONE
mis_rate1 <- c()
for(i in 1:1000){

```

```

x1<-runif(100)
x2<-runif(100)
trdata<-cbind(x1,x2)
y<-as.numeric(((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5)))
trlabels<-as.factor(y)

train_data <- as.data.frame(trdata)
train_data <- dplyr::mutate(train_data,trlabels)
Forest_1 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 1, nodesize = 12, keep.f
pre <- predict(Forest_1,newdata = train_data)
misclassification_rate <- mean(pre!=trlabels)
mis_rate1 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate1 <- mean(mis_rate1)
var_mis_rate1 <- var(mis_rate1)

# the mean and variable for misclassification rate when B = 1
mean_mis_rate1

## [1] 0.14995
var_mis_rate1

## [1] 0.006740238
# QUESTION THREE PART TWO
mis_rate10 <- c()
for(i in 1:1000){

  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5)))
  trlabels<-as.factor(y)

  train_data <- as.data.frame(trdata)
  train_data <- dplyr::mutate(train_data,trlabels)
  Forest_10 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 10, nodesize = 12, keep
  pre <- predict(Forest_1,newdata = train_data)
  misclassification_rate <- mean(pre!=trlabels)
  mis_rate10 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate10 <- mean(mis_rate10)
var_mis_rate10 <- var(mis_rate10)
# the mean and variable for misclassification rate when B = 10
mean_mis_rate10

## [1] 0.1500999

```

```

var_mis_rate10

## [1] 0.00675599
# QUESTION THREE PART THREE
mis_rate100 <- c()
for(i in 1:1000){

  x1<-runif(100)
  x2<-runif(100)
  trdata<-cbind(x1,x2)
  y<-as.numeric(((x1<0.5 & x2<0.5)|(x1>0.5 & x2>0.5)))
  trlabels<-as.factor(y)

  # QUESTION ONE
  train_data <- as.data.frame(trdata)
  train_data <- dplyr::mutate(train_data,trlabels)
  Forest_100 <- randomForest::randomForest(trlabels~.,data = train_data, ntree = 100, nodesize = 12, keep.forest = TRUE)
  pre <- predict(Forest_1,newdata = train_data)
  misclassification_rate <- mean(pre!=trlabels)
  mis_rate100 <- c(mis_rate1,misclassification_rate)
}

mean_mis_rate100 <- mean(mis_rate100)
var_mis_rate100 <- var(mis_rate100)

# the mean and variable for misclassification rate when B = 10
mean_mis_rate100

## [1] 0.1501598
var_mis_rate100

## [1] 0.006777574

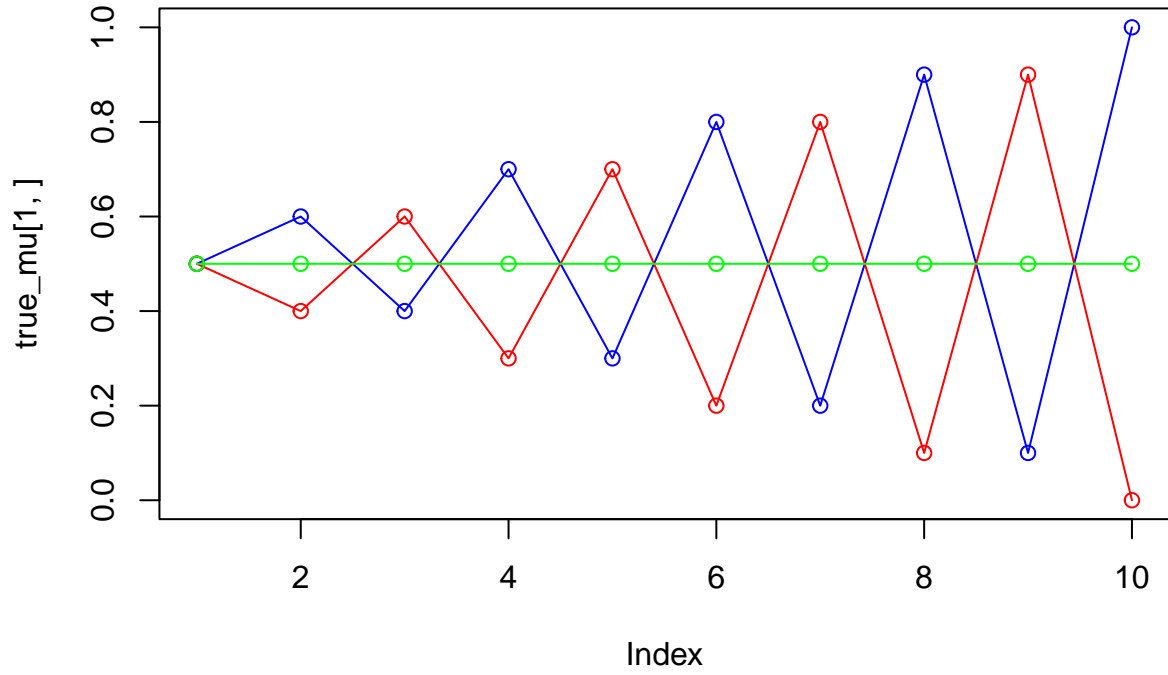
```

## QUESTION THREE

1.From the first two calculations, we can see the misclassification rate decreases first and then increases as the number of trees increases.

## 2. Mixture Models

For implementing the EM algorithm for Bernoulli mixture model First we generate a  $\pi$  and  $\mu$  of a Bernoulli mixture model

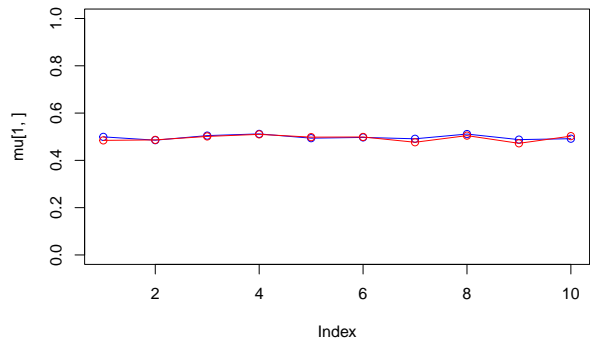
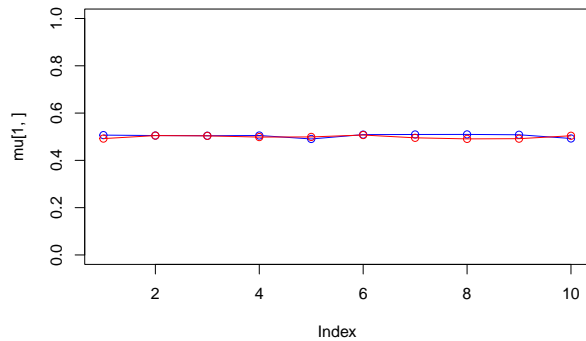


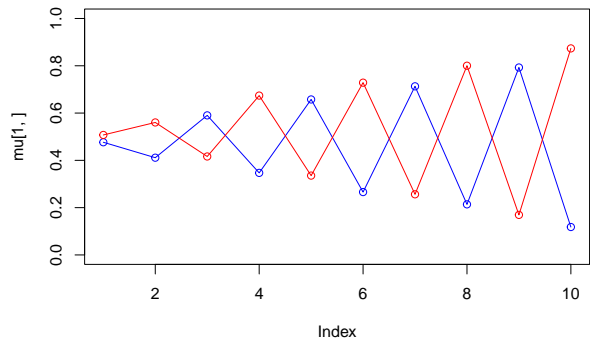
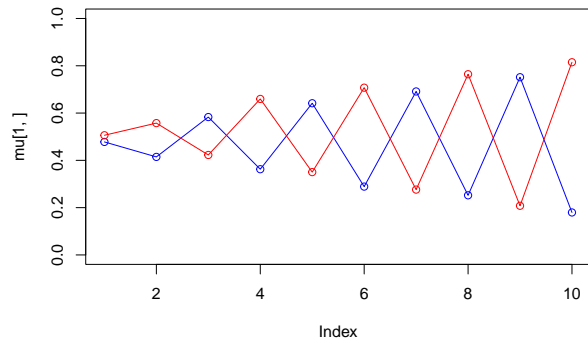
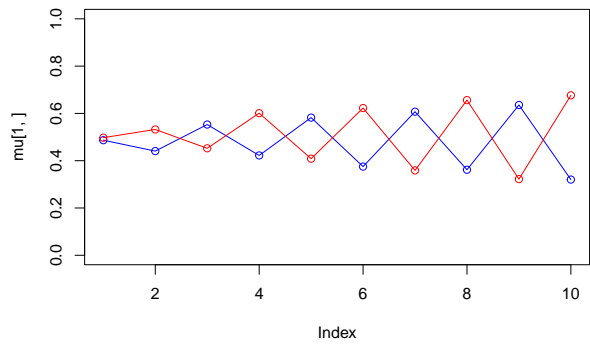
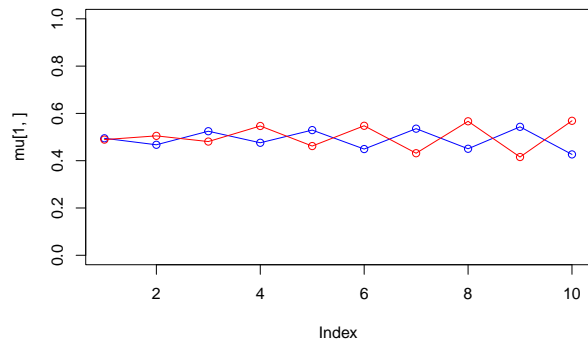
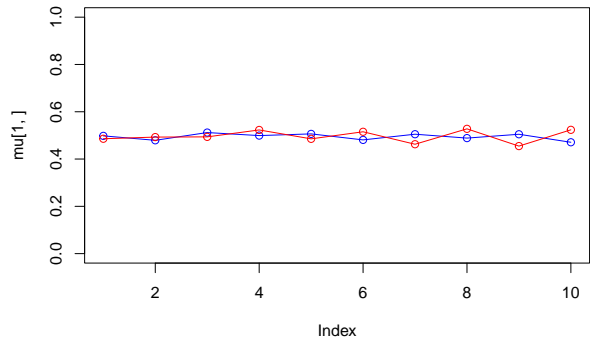
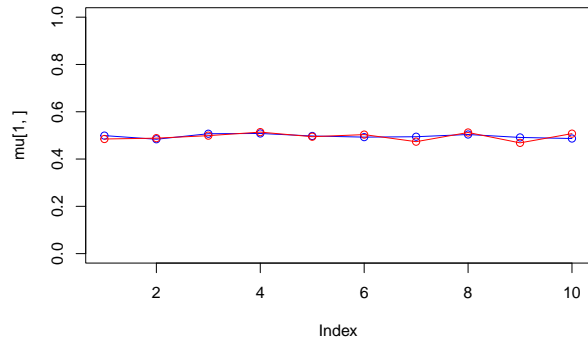
And the plot above shows the 3  $\mu$ s we are using to generate data set  $x$ , with a same  $\pi = \frac{1}{3}$  for each  $\mu$ , and of course with a  $Dimension = 10$ .

```
## [1] 0.5052178 0.4947822

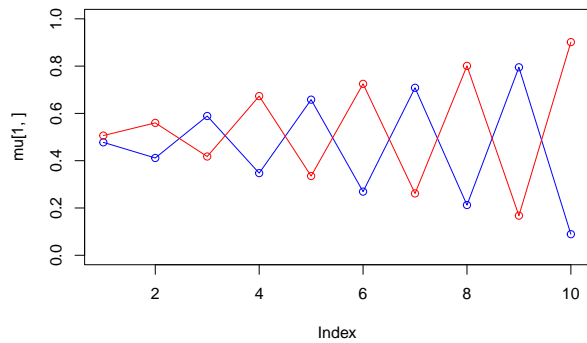
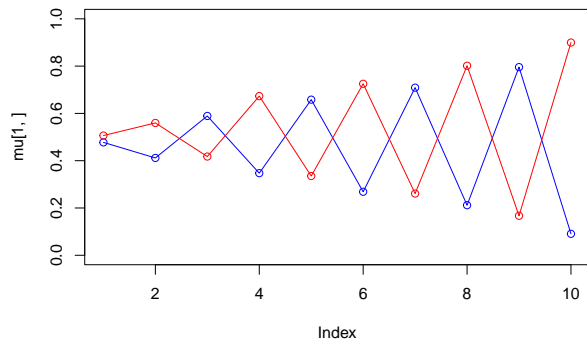
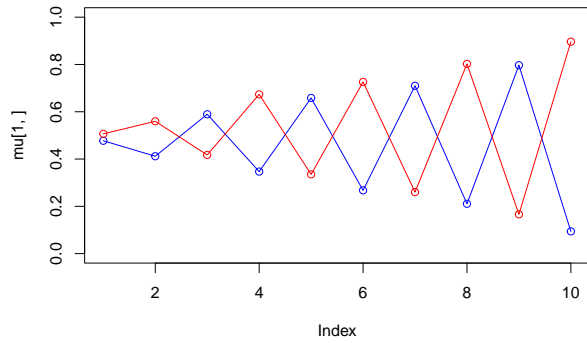
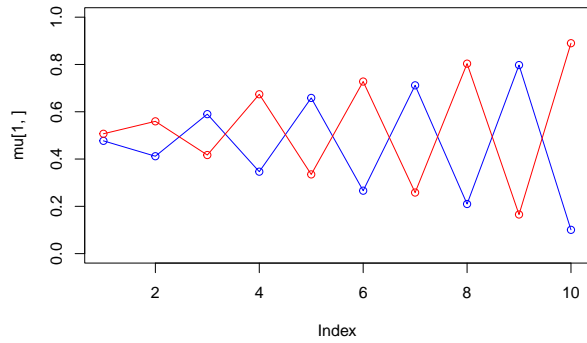
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5068040 0.5049455 0.5041509 0.5051382 0.4905578 0.5089228 0.509453
## [2,] 0.4921113 0.5048207 0.5036886 0.4985467 0.4991731 0.5071384 0.495380
##           [,8]      [,9]     [,10]
## [1,] 0.5097480 0.5082991 0.4926313
## [2,] 0.4908757 0.4917657 0.5040657
```

First we start with cluster  $M = 2$ . And we generate a  $\pi$  and  $\mu$  under  $M = 2$  as the start point for our model.





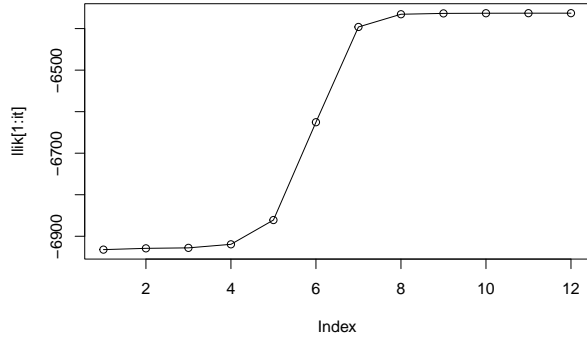




```
## [1] "iteration: 1 log likelihood: -6932.1625943363"
## [2] "iteration: 2 log likelihood: -6929.09536498845"
## [3] "iteration: 3 log likelihood: -6927.94380531483"
## [4] "iteration: 4 log likelihood: -6919.43478131895"
## [5] "iteration: 5 log likelihood: -6860.93456150249"
## [6] "iteration: 6 log likelihood: -6625.43370682142"
## [7] "iteration: 7 log likelihood: -6396.10695394879"
## [8] "iteration: 8 log likelihood: -6365.47503740155"
## [9] "iteration: 9 log likelihood: -6363.41093129657"
## [10] "iteration: 10 log likelihood: -6363.02364141854"
## [11] "iteration: 11 log likelihood: -6362.91755252171"
## [12] "iteration: 12 log likelihood: -6362.88538329492"

## [1] 0.4979156 0.5020844

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4776009 0.4114577 0.5889962 0.3474863 0.6581537 0.2691781 0.7083954
## [2,] 0.5062796 0.5599234 0.4177178 0.6731561 0.3351926 0.7249220 0.2614677
##           [,8]      [,9]     [,10]
## [1,] 0.2125293 0.7950371 0.08913449
## [2,] 0.8010174 0.1675785 0.90147909
```



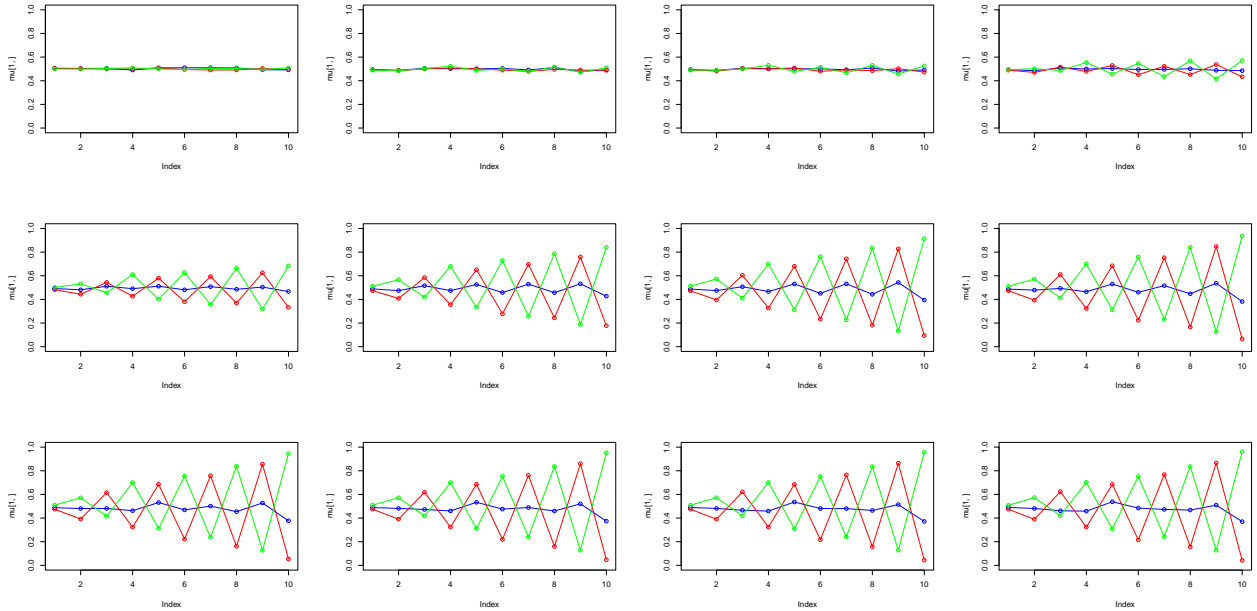
From the plots above. We can see that under this situation,  $\mu$  which is equal to 0.5 every dimension seems did not being reflect. It might be that the other 2  $\mu$ s are symmetrical to the 0.5 axis.

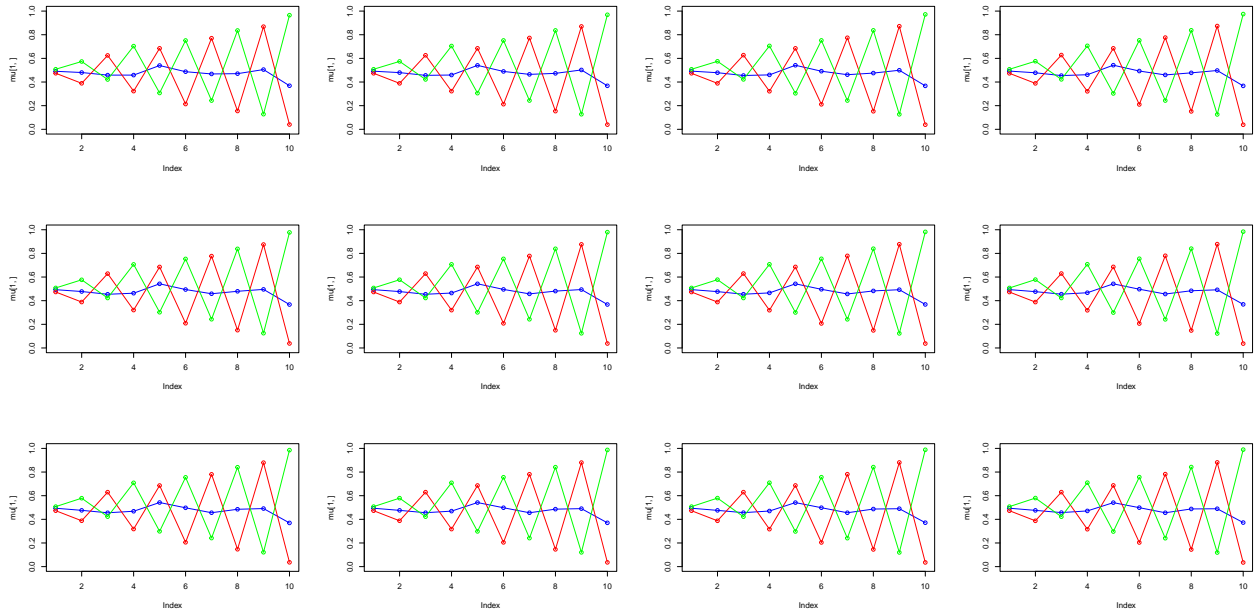
And for every iteration the predicted  $\mu$  is closer and closer to the *true*  $\mu$ . The rapidly raise of log-likelihood during iteration 5-7 is also reflect by the plot from significant changing of  $\mu$  in corresponding iterations. The final predicted  $\mu$  shows above is relatively close to the true value within around 10% differences.

```
## [1] 0.3359578 0.3290183 0.3350239

##          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5049455 0.5041509 0.5051382 0.4905578 0.5089228 0.5094530 0.5097480
## [2,] 0.5048207 0.5036886 0.4985467 0.4991731 0.5071384 0.4953800 0.4908757
## [3,] 0.4996279 0.4982070 0.5043346 0.5085042 0.4994862 0.4945702 0.5041462
##          [,8]      [,9]      [,10]
## [1,] 0.5082991 0.4926313 0.4921113
## [2,] 0.4917657 0.5040657 0.4956302
## [3,] 0.5040348 0.4955050 0.5088683
```

Then we goes to  $M = 3$ , again generate a  $\pi$  and  $\mu$  under  $M = 3$  as the start point for our model.



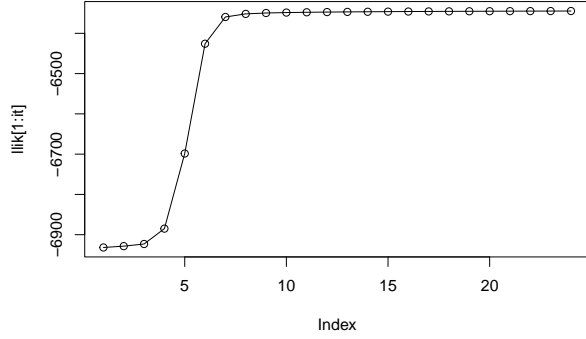


```
## [1] "iteration: 1 log likelihood: -6931.75122861544"
## [2] "iteration: 2 log likelihood: -6928.46695128332"
## [3] "iteration: 3 log likelihood: -6923.13091589768"
## [4] "iteration: 4 log likelihood: -6884.94468114899"
## [5] "iteration: 5 log likelihood: -6698.4237807576"
## [6] "iteration: 6 log likelihood: -6425.806406635"
## [7] "iteration: 7 log likelihood: -6359.4223002037"
## [8] "iteration: 8 log likelihood: -6351.74320723516"
## [9] "iteration: 9 log likelihood: -6349.54335699899"
## [10] "iteration: 10 log likelihood: -6348.4434579416"
## [11] "iteration: 11 log likelihood: -6347.73893891381"
## [12] "iteration: 12 log likelihood: -6347.21907132148"
## [13] "iteration: 13 log likelihood: -6346.8026280888"
## [14] "iteration: 14 log likelihood: -6346.45328398564"
## [15] "iteration: 15 log likelihood: -6346.15273441616"
## [16] "iteration: 16 log likelihood: -6345.89062787433"
## [17] "iteration: 17 log likelihood: -6345.66037682444"
## [18] "iteration: 18 log likelihood: -6345.45729529842"
## [19] "iteration: 19 log likelihood: -6345.2777404626"
## [20] "iteration: 20 log likelihood: -6345.11870927259"
## [21] "iteration: 21 log likelihood: -6344.97764443873"
## [22] "iteration: 22 log likelihood: -6344.852335347"
## [23] "iteration: 23 log likelihood: -6344.74086068395"
## [24] "iteration: 24 log likelihood: -6344.64154880195"
```

```
## [1] 0.2663361 0.3438780 0.3897860
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4938980 0.4758006 0.457040 0.4711376 0.5411178 0.4986316 0.4555200
## [2,] 0.4728756 0.3874569 0.630045 0.3163584 0.6869103 0.2039956 0.7823031
## [3,] 0.5075751 0.5799061 0.422322 0.7099547 0.2967462 0.7569457 0.2402904
##           [,8]      [,9]      [,10]
## [1,] 0.4878804 0.489488 0.37258194
## [2,] 0.1446699 0.881597 0.03501224
```

```
## [3,] 0.8422855 0.119219 0.98958938
```

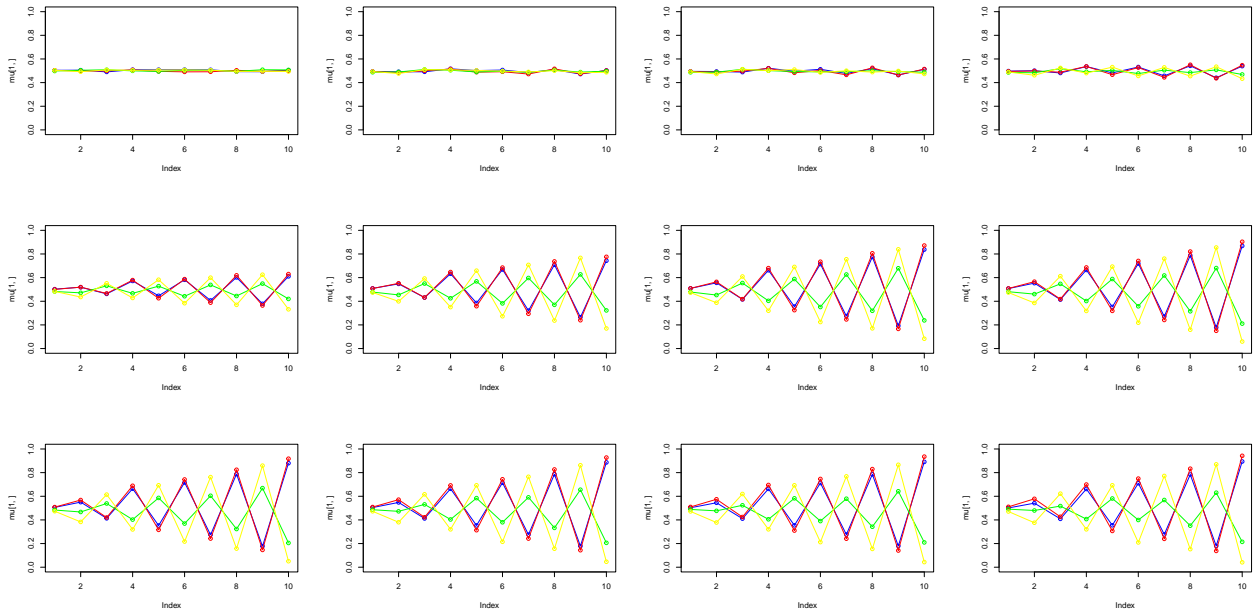


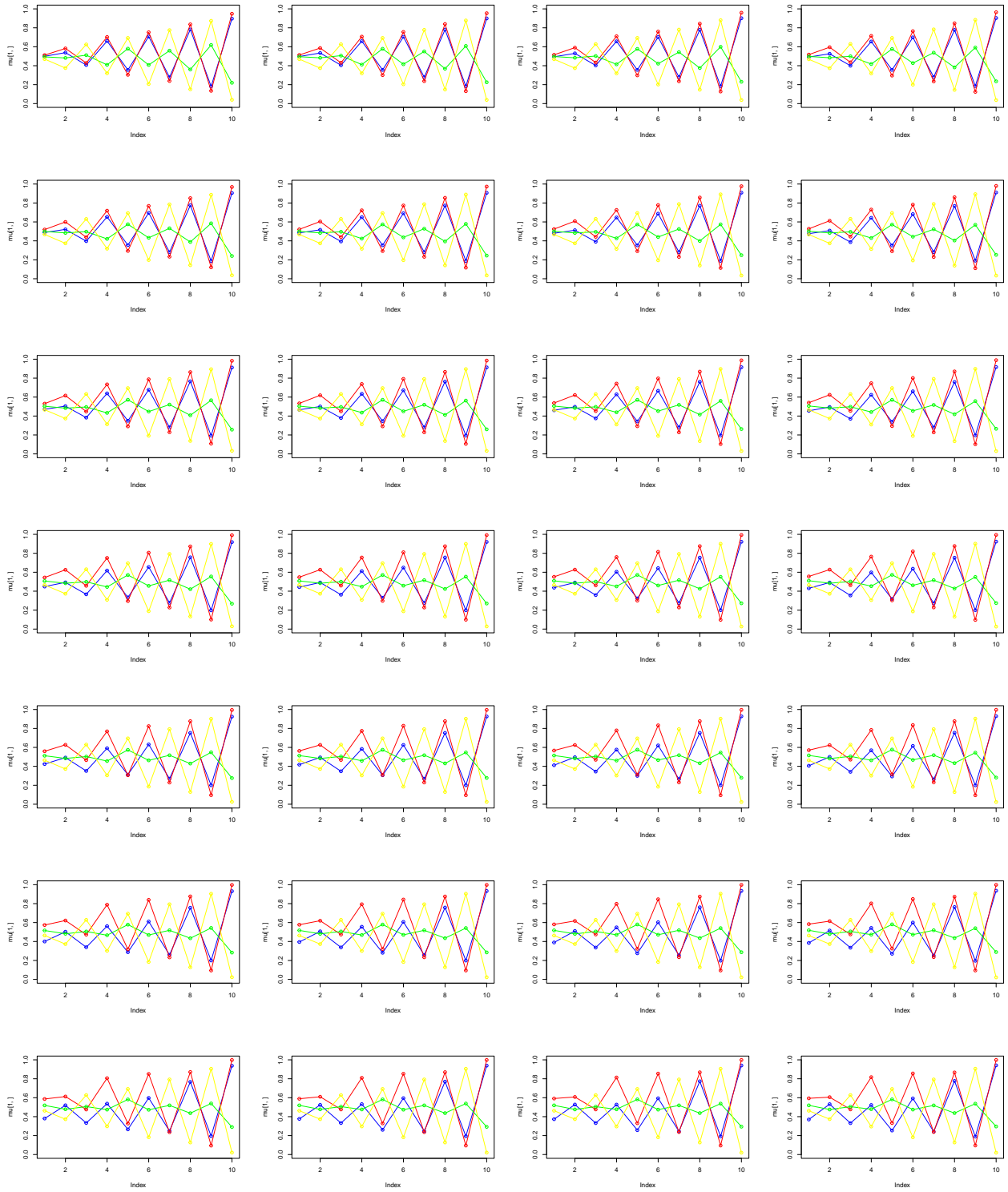
We can see that the log-likelihood rises rapidly during iteration 4-7, its also clearly shown by  $\mu$  values plot. Then from final result we can see that it predict the true values well, especially in comparison with following  $M = 4$ .

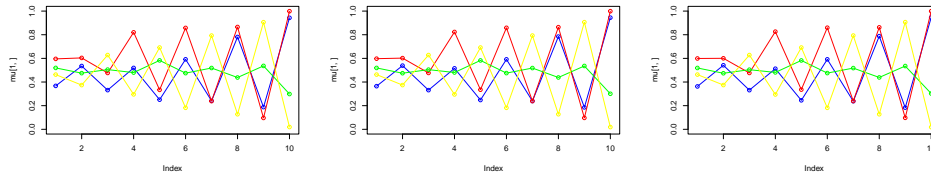
```
## [1] 0.2518811 0.2466783 0.2511809 0.2502598
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.5041509 0.5051382 0.4905578 0.5089228 0.5094530 0.5097480 0.5082991
## [2,] 0.5036886 0.4985467 0.4991731 0.5071384 0.4953800 0.4908757 0.4917657
## [3,] 0.4982070 0.5043346 0.5085042 0.4994862 0.4945702 0.5041462 0.5040348
## [4,] 0.5037389 0.4922173 0.5069624 0.5039756 0.5065369 0.5073122 0.5049473
##           [,8]      [,9]     [,10]
## [1,] 0.4926313 0.4921113 0.5048207
## [2,] 0.5040657 0.4956302 0.4996279
## [3,] 0.4955050 0.5088683 0.5072302
## [4,] 0.4943372 0.4951750 0.4940898
```

Then we change  $M$  to 4, again generate a  $\pi$  and  $\mu$  under  $M = 4$  as the start point for our model.





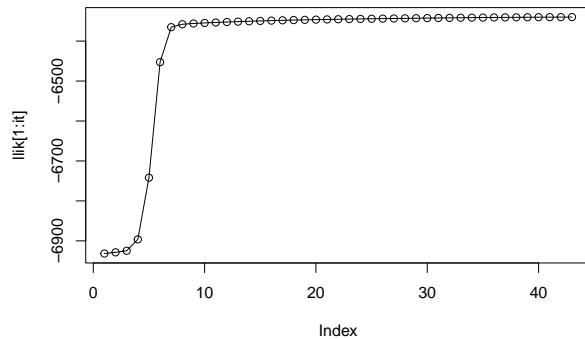


```
## [1] "iteration: 1 log likelihood: -6931.66006994864"
## [2] "iteration: 2 log likelihood: -6928.66727678128"
## [3] "iteration: 3 log likelihood: -6924.77054344575"
## [4] "iteration: 4 log likelihood: -6896.29508885115"
## [5] "iteration: 5 log likelihood: -6741.85787805056"
## [6] "iteration: 6 log likelihood: -6452.71193962151"
## [7] "iteration: 7 log likelihood: -6365.22915492218"
## [8] "iteration: 8 log likelihood: -6357.90724015069"
## [9] "iteration: 9 log likelihood: -6355.94977356183"
## [10] "iteration: 10 log likelihood: -6354.59564838264"
## [11] "iteration: 11 log likelihood: -6353.40988750751"
## [12] "iteration: 12 log likelihood: -6352.31517546734"
## [13] "iteration: 13 log likelihood: -6351.30092218428"
## [14] "iteration: 14 log likelihood: -6350.36818071176"
## [15] "iteration: 15 log likelihood: -6349.51652090179"
## [16] "iteration: 16 log likelihood: -6348.74223300225"
## [17] "iteration: 17 log likelihood: -6348.03936943375"
## [18] "iteration: 18 log likelihood: -6347.40094567819"
## [19] "iteration: 19 log likelihood: -6346.81968070248"
## [20] "iteration: 20 log likelihood: -6346.28834868353"
## [21] "iteration: 21 log likelihood: -6345.79994106786"
## [22] "iteration: 22 log likelihood: -6345.34778016045"
## [23] "iteration: 23 log likelihood: -6344.92564619259"
## [24] "iteration: 24 log likelihood: -6344.52792780162"
## [25] "iteration: 25 log likelihood: -6344.14978243089"
## [26] "iteration: 26 log likelihood: -6343.78728731843"
## [27] "iteration: 27 log likelihood: -6343.43756285041"
## [28] "iteration: 28 log likelihood: -6343.09885115178"
## [29] "iteration: 29 log likelihood: -6342.77053143643"
## [30] "iteration: 30 log likelihood: -6342.4530520589"
## [31] "iteration: 31 log likelihood: -6342.14776283848"
## [32] "iteration: 32 log likelihood: -6341.85664504623"
## [33] "iteration: 33 log likelihood: -6341.58196036309"
## [34] "iteration: 34 log likelihood: -6341.32586665114"
## [35] "iteration: 35 log likelihood: -6341.09006544297"
## [36] "iteration: 36 log likelihood: -6340.87554389922"
## [37] "iteration: 37 log likelihood: -6340.68245157758"
## [38] "iteration: 38 log likelihood: -6340.51011833165"
## [39] "iteration: 39 log likelihood: -6340.35718728741"
## [40] "iteration: 40 log likelihood: -6340.22181690354"
## [41] "iteration: 41 log likelihood: -6340.10190243326"
## [42] "iteration: 42 log likelihood: -6339.99527661232"
## [43] "iteration: 43 log likelihood: -6339.89986543919"
```

```
## [1] 0.1838186 0.2308466 0.2874187 0.2979162
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
```

```
## [1,] 0.3627106 0.5416076 0.3311785 0.5125500 0.2451860 0.5918027 0.2376350
## [2,] 0.5991155 0.6005747 0.4768083 0.8260669 0.3357255 0.8593412 0.2392769
## [3,] 0.5188276 0.4740528 0.5042778 0.4812818 0.5823257 0.4755847 0.5177702
## [4,] 0.4628904 0.3744350 0.6280787 0.2945785 0.6916641 0.1817551 0.7930593
##      [,8]      [,9]     [,10]
## [1,] 0.7878580 0.18327423 0.94572455
## [2,] 0.8624625 0.09719753 0.99920005
## [3,] 0.4387190 0.53515949 0.30265960
## [4,] 0.1275007 0.90649071 0.01848283
```



We can see that after the rapid rising of log-likelihood, the iteration 8-15 seems still have a reasonable  $\mu$  value, but then it falls into overfitting and the result  $\mu$  and  $\pi$  values seems not so favorable.

*# Code Appendix for part 2*

```
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log lik between two consecutive iterations
n=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=n, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients ; p(y)
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions ; p(xcol | y = Row)
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# Producing the training data x
for(i in 1:n) {
  m <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[i,d] <- rbinom(1,1,true_mu[m,d])
  }
}

set.seed(1234567890)
M <- 2 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights ; p(y = m | xi, thetaHat)
pi <- vector(length = M) # mixing coefficients
```

```

mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the parameters
pi <- runif(M,0.49,0.51)
pi <- pi / sum(pi)
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}
pi
mu

set.seed(1234567890)
iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
  # points(mu[3,], type="o", col="green")
  # points(mu[4,], type="o", col="yellow")
  Sys.sleep(0.5)
  # E-step: Computation of the weights

  for (i in 1:n) {
    # px <- 0
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
        # print(paste(i,m,d,bernXMum))
      }
      pxi <- pxi + pi[m] * bernXMum
      # print(paste(i,m,d,pxi))
    }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
        # print(paste(i,m,d,bernXMum))
      }
      w[i,m] <- (bernXMum * pi[m]) / pxi
    }
    w[i, ] <- w[i,] /sum(w[i,])
  }

  # Your code here
  #Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {

```



```

    bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
  }
  pxi <- pxi + pi[m] * bernXMum
}
llik[it] <- llik[it] + log(pxi)
}

# Your code here
iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])
flush.console()
# Stop if the log likelihood has not changed significantly
stopFlag <- it > 1 && (llik[it] - llik[it - 1]) < min_change
if(stopFlag) break
#M-step: ML parameter estimation from the data and weights
# pi mu
pi <- apply(w, 2, mean)
mu <- t(w) %*% x / colSums(w)
# Your code here
}
print(iterLog[1:it])
pi
mu
plot(llik[1:it], type="o")

set.seed(1234567890)
M <- 3 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights ; p(y = m | xi, thetaHat)
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the parameters
pi <- runif(M,0.49,0.51)
pi <- pi / sum(pi)
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}
pi
mu

iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  # plotChoose <- c(1, 5, 6, 12, 24)
  # if (it == any(plotChoose)) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  # points(mu[4,], type="o", col="yellow")
  # }
  Sys.sleep(0.5)
  # E-step: Computation of the weights

```

```

for (i in 1:n) {
  # px <- 0
  pxi <- 0
  for (m in 1:M) {
    bernXMum <- 1
    for (d in 1:D) {
      bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
      # print(paste(i,m,d,bernXMum))
    }
    pxi <- pxi + pi[m] * bernXMum
    # print(paste(i,m,d,pxi))
  }
  for (m in 1:M) {
    bernXMum <- 1
    for (d in 1:D) {
      bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
      # print(paste(i,m,d,bernXMum))
    }
    w[i,m] <- (bernXMum * pi[m]) / pxi
  }
  w[i, ] <- w[i,] / sum(w[i,])
}

# Your code here
#Log likelihood computation.
llik[it] <- 0
for (i in 1:n) {
  pxi <- 0
  for (m in 1:M) {
    bernXMum <- 1
    for (d in 1:D) {
      bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
    }
    pxi <- pxi + pi[m] * bernXMum
  }
  llik[it] <- llik[it] + log(pxi)
}

# Your code here
iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])
flush.console()
# Stop if the log likelihood has not changed significantly
stopFlag <- it > 1 && (llik[it] - llik[it - 1]) < min_change
if(stopFlag) break
#M-step: ML parameter estimation from the data and weights
# pi mu
pi <- apply(w, 2, mean)
mu <- t(w) %*% x / colSums(w)
# Your code here
}

```

```

print(iterLog[1:it])
pi
mu
plot(llik[1:it], type="o")

set.seed(1234567890)
M <- 4 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights ;  $p(y = m \mid x_i, \theta_{\text{hat}})$ 
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the parameters
pi <- runif(M, 0.49, 0.51)
pi <- pi / sum(pi)
for(m in 1:M) {
  mu[m,] <- runif(D, 0.49, 0.51)
}
pi
mu

iterLog <- vector(length = max_it)
for(it in 1:max_it) {
  # plotChoose <- c(1, 15, 43)
  # if (it == any(plotChoose)) {
  plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  points(mu[2,], type="o", col="red")
  points(mu[3,], type="o", col="green")
  points(mu[4,], type="o", col="yellow")
  # }
  Sys.sleep(0.5)
  # E-step: Computation of the weights

  for (i in 1:n) {
    # px <- 0
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
        # print(paste(i,m,d,bernXMum))
      }
      pxi <- pxi + pi[m] * bernXMum
      # print(paste(i,m,d,pxi))
    }
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
        # print(paste(i,m,d,bernXMum))
      }
      w[i,m] <- (bernXMum * pi[m]) / pxi
    }
  }
}

```

```

    w[i, ] <- w[i,] /sum(w[i,])
  }

  # Your code here
  #Log likelihood computation.
  llik[it] <- 0
  for (i in 1:n) {
    pxi <- 0
    for (m in 1:M) {
      bernXMum <- 1
      for (d in 1:D) {
        bernXMum <- bernXMum * (mu[m,d]^x[i,d]) * (1-mu[m,d])^(1-x[i,d])
      }
      pxi <- pxi + pi[m] * bernXMum
    }
    llik[it] <- llik[it] + log(pxi)
  }

  # Your code here
  iterLog[it] <- paste("iteration: ", it, "log likelihood: ", llik[it])
  flush.console()
  # Stop if the log likelihood has not changed significantly
  stopFlag <- it > 1 && (llik[it] - llik[it - 1]) < min_change
  if(stopFlag) break
  #M-step: ML parameter estimation from the data and weights
  # pi mu
  pi <- apply(w, 2, mean)
  mu <- t(w) %*% x / colSums(w)
  # Your code here
}

print(iterLog[1:it])
pi
mu
plot(llik[1:it], type="o")

```